IMPLEMENTATION OF THE 'ICPL' GRAPHICS LANGUAGE ON THE PDP-15 COMPUTER

C.D. O'Brien
Faculty of Engineering
Carleton University
Ottawa, Ontario

Abstract

The 'ICPL' (Interactive Control Program Language) is a high level graphics language developed at the Communications Research Centre to facilitate the writing of interactive graphics software. This paper briefly discusses the 'ICPL' language and describes in detail its implementation on the PDP-15, VT-15 graphics system at Carleton University. Problems encountered in transferring graphics software from one machine to another are also discussed.

Abrégé

Le langage ICPL (Interactive Control Programming Language) est un langage graphique motivé qui a été conçu et réalisé au Centre de Recherches sur les Communications pour faciliter le develompent de programmerie graphique intéractive. Ce document définit brièvement ICPL et décrit en detail sa réalisation sur le système graphique PDP-15, VT-15 de l'Université Carleton. Les divers problèmes associés au transport de programmerie graphique d'une machine à une autre sont aussi présentés.

Introduction

The Interactive Control Program Language[1] is a high level graphics language developed at the Communications Research Centre to facilitate the writing of interactive graphic software. It completely removes the graphics programmer from interrupt level programming and forces him to think of displayable objects and actions resulting from identifier strikes upon these objects. This concept is common to most forms of graphics and makes 'ICPL' generally machine independent. The original implementation of the language was on a drum based graphics system on the PDP-9 which is very different from the core based graphics system on the PDP-15, VT-15 system.

The control program is written in the form of object - action pairs, with each action associated with an object. The program is parallel in nature and the object action pairs require no special ordering. A menu of light-buttons can easily be defined as a series of object statements and the associated action statements identify the code to be executed when an interrupt driven command such as a light-pen strike occurs. A simple 'ICPL' program is shown below.

```
CONTROL PROGRAM TEST
STRING(6) BUFF
* THIS SECTION OF CODE IS EXECUTED ON ENTRY
ENTRY
1 DISPLAY
  SEEK
* THIS OBJECT STATEMENT DEFINES THE LIGHT-
* BUTTON 'APPEND A' AND APPENDS IT TO THE
* DISPLAY FILE.
OBJECT
   TEXT 500,500/'APPEND A'
ACTION
   APPEND TO BUFF/'A'
   GOTO 1
OBJECT
   TEXT 500,400/'APPEND B'
ACTION
   APPEND TO BUFF/'B'
   GOTO 1
OBJECT
   TEXT 500,300/'CLEAR'
ACTION
   CLEAR BUFF
   GOTO 1
* THIS OBJECT DISPLAYS THE CONTENTS OF THE
* THE BUFFER AND UNDERLINES IT.
OBJECT
   LIST 600,450/BUFF
   TRACE 600,438/60,0
END
```

This program allows up to 6 characters, A or B, to be appended to a buffer, BUFF, in any order. A light-button 'CLEAR' is provided for emptying the buffer.

PDP-15 Implementation

The routines which generate the light-button menus and those that interpret the light-pen strikes and control the execution of the specialized functions are peculiar to each application and contain code that is highly dependent on the structure of the display and light-pen system. The object-action structure of 'ICPL' lends to portability because this structure is independent of the particular graphic hardware of a machine. The run-time environment of an 'ICPL' program provides the interface between standardized function calls and the particular machine hardware.

The task of implementing this language on a computer consists of:

i) transporting the translator for the language to the new machine.

ii) providing the run-time environment to allow 'ICPL' to interact with the graphics hardware.

The translator for the 'ICPL' language was written using the machine independent macro processor 'STAGE2' of the 'Mobile Programming System' of Prof. W.M. Waite[2], of the University of Colorado. 'STAGE2' is the second level of a macro processor bootstrap sequence which is easily implemented on any computer. It is a flexible powerful macro processor defined in itself. It is implemented on a pseudo machine called FLUB which is an idealized 24 bit machine defined as a series of macros translatable by 'STAGE2' and by 'SIMCMP' a much simpler macro processor written in less than 100 lines of Fortran. 'SIMPCMP' bootstraps 'STAGE2'.

Because of the high portability of 'STAGE2' it can be installed on a machine in about a weeks' work. Since 'ICPL' is defined as a set of macros for 'STAGE2' only the code emitting parts of 'ICPL' must be changed in order to install it on a machine. The PDP-15 and the PDP-9 have essentially identical processors so the 'ICPL' translator ran directly without modification.

The run-time environment of 'ICPL' consists of providing the routines to draw vectors and text on the screen and providing controls on intensity, relative or absolute coordinates, a tracking cross and a tagging mechanism. Providing the display dependent routines was the major task in implementing 'ICPL' on the PDP-15.

The overall software structure is described in Figure 1. The two routines ICPSR and BHA provide the mechanism whereby light-pen strikes are identified with the object and items described by the OBJECT statements in an 'ICPL' program. They are also concerned with the implementation of the character string variables. No changes were required in these routines because the assembly languages on both machines are almost identical.

The PDP-15, VT-15 graphic system is structured as a dual processor system sharing common core. Both the CPU and DPU (display processor unit) execute simultaneously and the DPU has its own set of special graphics instructions. A display file consists of an area of core containing graphics processor instructions which are stored there by the main CPU. The graphics system on the PDP-9 computer used at CRC is very different. The machine has a single processor and its display controller is like an I/O device. Incremental graphics codes consisting of short vectors are put on a drum by the CPU. On every drum rotation this display code is continually refreshing the screen. The drum has an enormous display space while core on the PDP-15 restricts display files.

The run-time environment on the PDP-15 was configured as an interrupt handler along with a group of subroutines to build and maintain display files. The object identification scheme requires that every entity is to be considered logically unique and separated by an identifier, a tag code. The system counts these tag codes from the beginning of the display file on every refresh. If a light-pen interrupt occurs the tag counter is stopped and the number it contains is the entity number of the object that caused the light-pen interrupt. This was largely done by hardware on the PDP-9 and had to be established on the PDP-15 by complicated interrupt processing. The graphics stop code is used as a tag and upon a stop code interrupt the CPU increments a counter if the light-pen hasn't interrupted, and then resumes the display. The tag mechanism is turned on and off by dynamic patching of stop codes.

A tracking cross is a graphical object designed to follow a light-pen. The tracking cross implemented on the PDP-15 consists of a hexagon of vectors surrounding an inactive marker. It was programmed so that the tracking cross captures the display and is multiply displayed while tracking so as to improve its tracking performance.

The display package is a modular package of system routines consisting of a specialized I/O handler for the display controller and a package of basic display generation routines.

The display generation package provides facilities for creating character and vector strings and is callable from FORTRAN or assembly-language programs. A description of the most important ones are as follows:

<u>Character Strings</u>

1) create a character string entity

```
TEXT (IX,IY,STRING(1),N)
```

where IX, IY are starting coordinates; STRING is an array containing the string as Hollerith constant; N(integer) is the number of characters in the string.

2) initialize a character string entity

```
CSTG(IX,IY)
```

to be followed by calls of type (3).

3) add a character to a string entity

```
CHAR(ICHAR)
```

where ICHAR is an integer representing the character code.

<u>Vector Strings</u>

4) create a vector string entity

```
VECT(IX,IY,IDX,IDY)
```

where IX, IY are starting coordinates; IDX,IDY are the projected lengths of a single vector.

5) initialize a vector string entity

    VSTG(IX,IY)

   to be followed by calls of type (6).

6) add a vector to a string entity

    DRAW(IDX,IDY)

   where IDX, IDY are the projected lengths of the vector.

<u>Mode Switching</u>

set absolute or relative mode for the origin of character or vector strings

7) MODE(M)

   where M is the numerical code for the desired setting mode, absolute or relative.

8) intensity setting

    INTY(M)

   where M is a numerical code for desired intensity.

9) tracking cross mode

    TCMODE(M)

   where M is a numerical code for the desired tracking cross restraint. (free,free in x, free in y,fixed, off)

10) set tracking cross position

    TCSET(IX,IY)

   where IX,IY are the tracking cross co-ordinates.

11) read tracking cross position

    TCLOC(IX,IY)

   where IX,IY are the tracking cross coordinates.

<u>Display File Administration</u>

12) tag

TAG

writes a tag in the current position of the display file.

13) start a new display file

DFST

14) close a display file

CLEAR

15) append new material to display file

APND

16) determine whether the light-pen is pointing at a display entity and set the entity number

%LPS

This routine is part of the interrupt handler and callable only by the ICPL service routines.

The display package routines are all written in assembly language and occupy 1540 words of core.

Conclusion

Since the 'ICPL' translator is written in the macro processor language 'STAGE2' and since it makes standardized requests upon a machine dependent run-time package, 'ICPL' is fairly portable graphics language. Some basic machine dependencies which can't be eliminated, such as the size of the display file and the refresh rate still exist. The PDP-9 and the PDP-15 are similar enough on these points to make the transfer of several applications programs from one machine to the other possible. One major incompatibility in transferring graphics software from one machine to another is that the screen coordinates are dependent on the particular hardware of the system. A simple scaling can't be done because the ratio of character size to screen increment size is not constant between systems. This is the problem encountered in transporting CRC's software to the PDP-15 and this type of problem will remain until the industry adopts standards.

This work was initiated while the author was employed at the Communications Research Centre and continued at Carleton University. I wish to thank Mr. H.G. Bown for his assistance in this work.

References

1. M.A. Maclean, *Designing a Language for Interactive Control Programs.* 2nd NRC Man-Computer Communications Seminar on Interactive Graphics. 31 May - 1 June 1971, pp. 30-39.

2. W.M. Waite, *The Mobile Programming System:STAGE2*'. ACM 13, No. 7, July 1970, pp. 415-421.

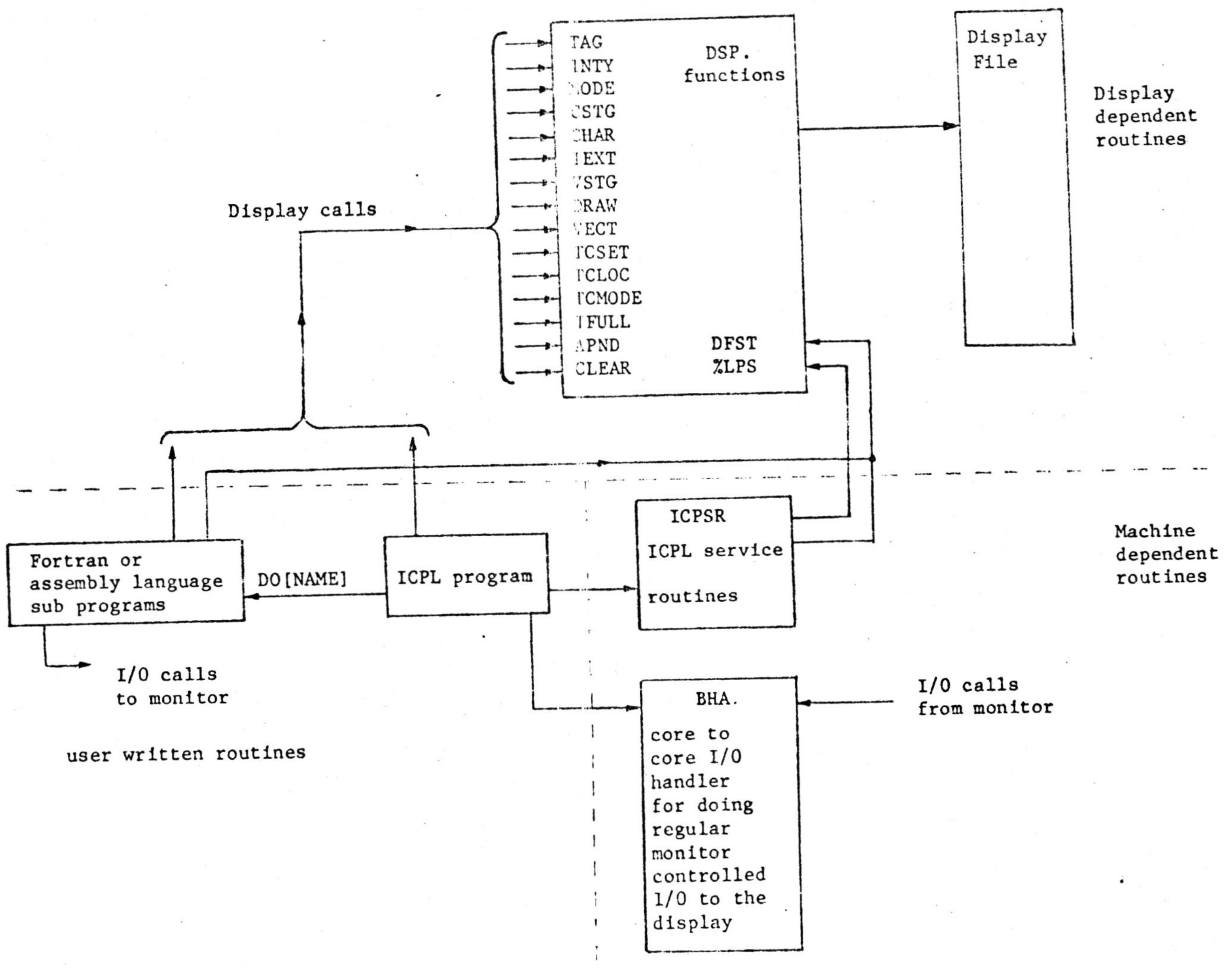

Figure 1 Overall Software Structure