MACHINE-INDEPENDENTMETACODE TRANSLATION

Thomas Wright
The National Center for Atmospheric Research†

Abstract:

Many systems implement plotter device-independent computer graphics by having a system plot package which outputs a plotter-independent code (here called metacode) and having a translating driver for each plotter which uses this code as input. The translator for this code can often be run with greatest efficiency on the computer which hosts the plotter. In NCAR's configuration, various computers will drive different plotters, making a portable metacode translator a desirable tool. Constructing a metacode translator which can drive the simplest devices and yet provide the potential to use sophisticated plotter hardware features is a stimulating challenge. The design and implementation of such a translator are described.

Keywords and Phrases: Device-independent graphics, Metacode translation, Portability

CR Categories: 8.2, 4.19

Introduction

Device-independent computer graphics are implemented at NCAR by using metacode [2]. This device-independent instruction set is produced on each of the computers in NCAR's network where user programs execute, such as the Cray-1, the Control Data 7600, the general purpose satellites, or the high performance graphics satellite (see figure 1).†† Translation of the metacode is done on the various computers where the individual graphics devices reside. The general purpose satellites translate metacode into instructions for a microfilm recorder. The Control Data 7600 uses metacode to produce instructions for its microfilm recorder. The high performance graphics satellite translates metacode into instructions for its display. Various computers can translate metacode, producing output tapes for off-line plotters.

† The National Center for Atmospheric Research is sponsored by the National Science Foundation.

†† At this writing, the hardware configuration is only partially implemented, but this introduction assumes the ultimate configuration.

Siggraph '77, July 20-22 San Jose, California

When enormous volumes of instructions are to be translated, an inflexible machine language program is used to translate the metacode. The high speed microfilm device on the 7600, for example, currently averages several hundred instructions per second around the clock. This makes a small, fast metacode translator very important. There is, however, a need for another type of translator.

A portable translator makes some efficiency sacrifices to achieve other goals. Because it is portable, the translator is easily moved from one host computer to another. Because it can reduce high level constructs to low level ones, the least sophisticated plotters can be supported. Because the code has clearly marked interface points, a new plotter can be added to the system with a small effort.

Achieving *Portability*

The translator is written in PFORT [1], a portable subset of FORTRAN. A verification program was used to check that the translator adheres to this standard. When moving the translator to a new computer, the implementor sets certain machine-dependent constants and supplies FORTRAN-callable functions for shifting and masking (IAND, IOR, ISHIFT). The translator assumes that positive integers are stored in binary and that the host's default length integer variables have 16 or more bits.

Perhaps a word on portable FORTRAN is in order. The syntactic constraints of PFORT tend to be an irritant rather than an obstacle. Standards which are little known to many FORTRAN users include:

- when initializing an array with a DATA statement, each element of the array must be individually listed before the first slash (as in DATA A(1),A(2),A(3),A(4),A(5)/5*0/).
- mixed modes of real and integer are not permitted (as in A = A+I).
- if a routine references two other routines which share a named COMMON block, the calling routine must also share that named common block.
- DATA statements referring to named common blocks can only appear in BLOCK DATA routines. These and other standards are checked using the PFORT verification program.

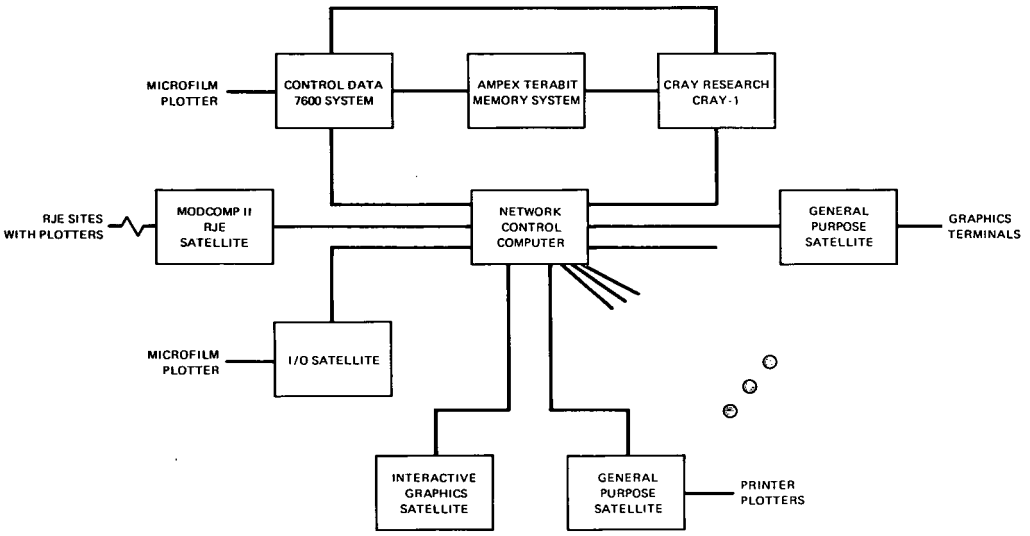

FIGURE 1
NCAR CONFIGURATION

The main program of the translator contains a location for the implementor to set various parameters, checks these parameters, and repeatedly invokes a routine which translates the metacode for one picture until all the metacode is exhausted. To prevent errors in transporting the translator, the parameters which are to be established by the implementor are originally initialized to disallowed values, and a run-time check is made to see that the implementor has set the parameters to reasonable and consistant values.

The routine which processes a picture's worth of metacode uses four lower level packages. Three of these--a character generator, a dashed line package, and the plotter interface--are discussed in later sections. The fourth supplies the metacode in chunks of a size which is easily digested by the translator. For the metacode in use at NCAR, these chunks are 16 bits long.

Portable extraction of 16-bit bytes from a bit stream must be carefully handled. The routine for this task assumes that a bit stream can be read into and accessed from an integer array completely filling each individual word except for possibly the last word used. This is true on nearly all machines (possible exceptions include 16-bit minicomputers reading from seven-track tape drives and long-word-length machines with short integer registers, such as a **Cray-1).** Further, the code assumes that default length integers are some multiple of four bits in length. Four packets of four bits each are extracted using shifts and masks from the current word being used from the buffer. For each packet, a test is made to see if all the bits have been obtained from the current word. If a new word is needed, a test is also made to see if an end-of-file is encountered. This must be done with compiler-dependent code. When four' packets have been obtained, they are shifted and OR'd together to form a 16-bit byte.

Simplifying High Level Constructs

The metacode used at NCAR has high level constructs which allow the use of important hardware features often available on sophisticated plotters. The two most important constructs are for drawing characters and for specifying dashed line patterns. For plotters with hardware characters and hardware dashed lines, instructions can be formed from the metacode which uses these features. For plotters without these capabilities, these constructs must be reduced to lower level constructs (pen movements) to emulate character generation and dashed line formation.

The portable generation of characters on a plotter is an interesting problem [3]. When translating metacode, the problem is simplified because the characters are in a known character code and are in a known position in the input string. In NCAR's metacode, characters are in ASCII and are stored in order, one per 8-bit byte. Each ASCII character is used to form an index to an array of pointers to the digitizations of the characters. Each digitization is used to form the individual strokes that make up each character. Even with only the 46 PFORT characters implemented, initializing the pointers and digitizations with standard DATA statements consumed about 250 statements.

The portable generation of software dashed lines is not a complicated problem when working with metacode because of the restricted nature of the input. A version of NCAR's simplest software dashed line package was easily integrated into the portable metacode translator without any special portability problems.

Other constructs, such as color and intensity, cannot be reduced to simple pen movements. To take advantage of these capabilities, extra code can be added by the implementor at places marked in the translation program to perform the desired function.

Plotter Interfacing

All plotting is handled through one routine. When establishing the parameters for the translator, the implementor specifies whether the translator should produce integer or floating point coordinates and the range for these coordinates. Two basic methods exist for interfacing to the plotter.

A small subset of the vendor-supplied software for the plotter can often be used to form the plotter's instructions. All scaling, labelling, and so on, will already be resolved, so only the lowest level line drawing routine need be referenced. This can provide a clean, efficient interface to the plotter if the vendor's software is highly modularized. Unfortunately, this is rarely the case.

Alternatively, a description of the plotter's hardware instruction set can be used as a basis for writing code to directly formulate instructions for the plotter. This is generally more work for the implementor than using vendor software, but often results in a smaller, more efficient interface.

For more sophisticated plotters, the forming of software characters and software dashed lines can be replaced by interfaces to hardware capabilities for these functions.

Conclusion

A portable metacode translator has been implemented and functions on several computers with a variety of plotters. The code is about 1100 statements, of which about one-third are comments. Various techniques for making the program easily transportable and flexible are described. The code has been implemented on IBM, Control Data, and PDP computers; plotters supported include Tektronix, CALCOMP, FR-80, and dd80.

References

[1] Ryder, B.G. "The PFORT Verifier," *Computer Graphics,* Vol. 4, No. 4 (1974), 359-377.

[2] Wright, T. "A Schizophrenic System Plot Package," *Computer Graphics,* Vol. 9, No. 1 (Spring 1975), 252-255.

[3] Wright, T. "SIGCHR--A Portable Character Generator," *Computer Graphics,* Vol. 10, No. 4 (Winter 1977).