NAPLPS Decoding in Forth

Dr. Billie Goldstein Stevens
CBS Technology Center
227 High Ridge Road
Stamford, Connecticut 06905

Abstract

The North American Presentation Level Protocol Syntax (NAPLPS) is a standard for encoding both textual and graphic information for videotex. Given a personal computer with hardware (either built in or added on) that can produce a display conforming to the standard, the actual decoding of NAPLPS-encoded data can be done in software. Such software decoders have been written both experimentally and as commercial products (primarily in Canada, where consumer videotex is more prevalent than in the U.S.). NAPLPS decoding is similar to recursive descent parsing, but NAPLPS code does not always have the clean structure of a context-free language. There are awkward jumps and unexpected escapes possible. Forth lends itself readily to the task of NAPLPS decoding, gracefully handling both the straightforward parsing and the sudden shifts. An experimental NAPLPS decoder in Forth has been developed at the CBS Technology Center. We briefly describe the NAPLPS coding techniques, then explain the design and implementation of the decoder. The suitability of Forth for this task is illustrated and emphasized.

1 INTRODUCTION

The North American Presentation Level Protocol Syntax [1] is a standard for encoding display data. There are two major components to a NAPLPS receiving device: a decoder and a display. The decoder may be either hardware or software. This paper describes a software NAPLPS decoder, written in Forth, that runs on an IBM PC with a colorboard. As we explain below, it would be a relatively easy task to port this decoder to different hardware.

Section 2 gives a greatly simplified description of the NAPLPS standard, explaining only as much as is necessary to understand what decoding entails. Section 3 describes the design and implementation of the Forth decoder, and discusses why Forth is particularly well-suited to NAPLPS decoding. This section elucidates why good programming practices, combined with the facililties provided by Forth, made writing the decoder an interesting and enjoyable project.

2 A SIMPLIFIED LOOK AT NAPLPS

As the heading implies, this section gives only an extremely brief overview of NAPLPS. The (literally) definitive work on NAPLPS is the North American PLPS standard [1]. An introduction to NAPLPS was presented several years ago by BYTE magazine [2].

NAPLPS is a standard formalism (i.e., language) for encoding, in a machine-independent manner, text and graphic display information for videotex. Each character in a NAPLPS stream is a token (word) of the language. NAPLPS has both a seven-bit and an eight-bit mode. For the sake of simplicity, this paper describes only the seven-bit mode; there is an eight-bit analog to each feature described below. A stream of NAPLPLS-encoded data is just a stream of ASCII characters, ranging in value from 0h to 7Fh. (Lower-case 'h' following a number indicates a hexadecimal value; all character values are represented this way.) Characters are usually interpreted as indices into a look-up table, called the

in-use table. The most significant hexadecimal digit of a character is the column number; the least significant digit is the row number. The action caused by a particular character depends on the contents of the table at the time the character is received.

The in-use table consists of two parts--a 32 element control set and a 96 element character set. The contents of the in-use table change when characters are received specifying a new control set or a new character set.

There are two control sets, known as C0 and C1, defined in NAPLPS. (The standard also leaves "hooks" for other control sets to be added.) The control set characters can be used to change cursor positions and text characteristics (normal, double-size, reverse video, etc.), as well as the contents of the in-use table.

There are six character sets: dynamically redefineable character set (DRCS), picture drawing instruction set (PDI), primary, supplementary, mosaic and macro sets. The primary, supplementary, mosaic, and DRCS sets are character sets in the conventional sense--that is, each character received when one of these character sets is in the in-use table causes one character to be drawn on the screen. The first three sets are fixed sets of characters, specified in the NAPLPS standard document. DRCS, as its name implies, allows the creator (encoder) of the information to define her/his own characters, but invocation of a character while the DRCS set is in use causes a particular (previously defined) character to be drawn. The PDI set is used primarily to draw basic geometric shapes--arcs, rectangles, points, polygons. The macro set is used to invoke previously defined streams of NAPLPS; that way, sequences that are used more than once need only be transmitted once.

We stated above that characters are "usually" interpreted as indices into the in-use table. Certain characters, in certain (control and character) sets introduce multi-character sequences. In these cases, subsequent characters are interpreted as part of a sequence, rather than as indices into the table. There are certain characters, however, that are always interpreted according to the table, even when they occur in what would otherwise be the middle of a sequence. The NAPLPS standard specifies what happens when a sequence is interrupted in this fashion. There are also characters that interrupt only some types of sequences.

All of these are aspects of NAPLPS that must be considered when designing a decoder. As stated above, there are many other features of NAPLPS; all them must be handled by the decoder. The features described here are only those necessary to understand the following sections.

3 THE FORTH DECODER

Forth decoding is similar to recursive descent parsing.* (For a detailed explanation of recursive descent parsing, consult any standard text on parsing or compilation, such as [3] or [4].) That is, the decoder, like a parser, inspects each token. In NAPLPS, each character is a token. (This minimizes transmission length as well as obviating the need for lexical analysis.) If the token initiates a particular type of sequence, the process that decodes (parses) that type of sequence is invoked. In programming languages that are parsed in this fashion, either a sequence-terminating token

* "Recursive descent interpreting" might be a more accurate phrase, since the decoding process acts on characters as they are received.

is received, or an error condition exists. In a NAPLPS stream, however, there are other possibilities.

Some sequences have fixed terminators--e.g., the END character (45h in the C1 set) marks the end of a macro, texture, or DRCS definition. Other sequences have a length that is known in advance by the sequence processor, such as the APS (active position set; 1Ch in the C0 set) cursor positioning character. In either of these cases, however, certain characters can cause termination of the sequence before either the terminator (in the first case) or the expected number of bytes (in the second) is received. What action is taken upon such "abnormal" termination is specified by the standard. In general, the characters that cause such termination also initiate other actions, so their values must be retained beyond the terminating process.

Additionally, there are sequences that have no "normal" termination--that is, sequences of indeterminate length with no special terminator. The four "draw polygon" commands (34h, 35h, 36h & 37h in the PDI set) are all in this category. In these cases, sequence processing continues until a character is received initiating some other action. At that point, processing of the current sequence must be completed by the decoder, and then processing of the new character must occur.

This aspect of NAPLPS can be viewed another way. That is, certain NAPLPS characters, occuring in certain contexts, serve a dual function-- they both terminate an ongoing sequence and initiate some other action. It is this feature that makes strict recursive descent algorithms inadequate for NAPLPS decoding.

The decoding process itself is similar to recursive descent, with "escape hatches" where necessary. Forth lends itself gracefully to this type of processing. The decoder was designed in a top-down fashion, with records kept of day-to-day progress. The first routine written was the highest-level decoder process; i.e., the routine that receives each character and decides what to do with it. This routine was subsequently revised three times before being frozen; each revision was necessitated by having encountered, while working on lower levels, some special case requiring a high-level escape hatch.

The design preceded from the top down; coding was done from the bottom up. Here again Forth made things very easy; each low-level routine could be tested and thoroughly debugged on its own. No linking or dummy routines are required for testing. Once the routines on one level were done, those on the next higher level could be worked on. Since the decoder was designed from the top down, this bottom-up implementation was always clearly directed.

Essentially, there are routines at five levels in the decoder. The lowest level includes all of those routines that actually change the display. These routines constitute a primitive graphics package for the PC; they are not strongly tied to NAPLPS in any way. All of the hardware-specific code is at this level.

At the next lowest level are the utility routines. The best example of this type of routine is the routine called OPS->XY; this routine takes an <X,Y> coordinate pair in the special format required by NAPLPS, and translates it into a normal integer pair. Another utility is the routine ?RANGE, which tests whether a given integer lies within a specified range.

The next level up includes all the routines that actually correspond to NAPLPS "commands." This level includes, but is not limited to, the following:

1. routines that draw individual geometric shapes, such as draw line, draw point, etc., each of which has four forms in NAPLPS (24h through 37h in the PDI set);
2. cursor positioning routines, e.g. active position set, forward, back, up, and down (08h, 09h, 0Ah, 0Bh, and 1Ch in the C1 set);
3. routines that put characters on the screen (for the primary, supplementary, and mosaic sets there is one routine that handles every character in that set);
4. routines that effect text display attributes, such as reverse/normal video and small/medium/normal text (48h through 4Ch in C1 set).

At the second highest level are the routines that determine how to interpret a given character. These routines do the work of determining what characters are part of ongoing sequences; what characters serve to terminate ongoing sequences and initiate others; etc. These routines call the processes at the next level down to do the actual work, once they have determined what needs to be done. The highest-level routine essentially performs the same function at a slightly higher level; it is described above.

It should be apparent that the decoder is highly modular in structure. The high degree of modularity is possible because the decoder was thoroughly designed from the top down, This modularity itself enables the clear separation of separate tasks. Because the decoder is written in Forth, each module could be thoroughly tested before being integrated with other modules. This simplifies the testing process while improving reliability.

A final advantage of this program structure is the relative portability of the decoder. A NAPLPS decoder is by definition machine-dependent. The structure of this decoder, however, clearly separates the machine-dependent aspects of decoding from the machine-independent aspects. Porting this decoder to different hardware requires rewriting only those routines that are strictly hardware-dependent; that is, what were referred to above as the lowest-level routines. (This assumes that a Forth system exists on the other hardware, of course.)

In general, developing this decoder was an intriguing and enjoyable project. (Most of the intrigue came from the mysteries and vagaries of NAPLPS.) Utilizing good program design techniques was essential; had we not started out in that fashion, we might still be debugging. Using Forth was almost as important in making the project manageable, possible, and fun.

4 REFERENCES

[1] Videotex/Teletext Presentation Level Protocol Syntax: North American PLPS. Published jointly by American National Standards Institute, Inc., New York, as ANS X3.110-1983; and by Canadian Standards Association, Rexdale (Toronto), Canada, as CSA T500-1983; December, 1983.

[2] Fleming, J. and W. Frezza. "NAPLPS: A New Standard for Text and Graphics." BYTE; 8:2,3,4; February, March, April, 1983.

[3] Gries, David. Compiler Construction for Digital Computers. John Wiley and Sons, New York, 1971.

[4] Aho, A. V. and J. D. Ullman. The Theory of Parsing, Translation and Compiling: Volume 1. Parsing. Prentice Hall, Englewood Cliffs, New Jersey, 1972.