Interactive Videodisc Control and Computer-Based Training on the Apple Macintosh™

Nick Francesco
J.A.M., Inc.
300 Main Street
East Rochester, NY 14445

In the education field, the use of video had traditionally been limited to short movies. The instructor (or audio-visual assistant) would turn off the lights, start the film, and for the next 45 minutes students sat through a linear presentation, attempting to take notes in the dark.

With the advent of videotape, the instructor had a slightly wider repertoire. The videotape could be used in a non-linear fashion; that is to say, the instructor could fast-forward or rewind to different areas of the tape. If the machine was sophisticated enough, the instructor could hold on a single image. However, the searches to different areas of the tape are slow; tapes can stretch, which means that the tape counter will not be accurate after five or six uses; and holding on an image ("freezing") will eventually wear through the tape at that point, resulting in a loss of the image or even tape breakage.

Videodisc offers an excellent alternative to videotape. The image on a videodisc can be frozen indefinitely without degradation of the image; search times are under 3 seconds end-to-end of the disc (depending on the player), and videodiscs are nearly indestructible, even in a classroom environment!

There are three methods of controlling the presentation on a videodisc. The first, and least expensive method, is to use the videodisc player's remote control unit. This allows the instructor to play a sequence, jump to any area of the disc, freeze the action, etc. The second method, which necessitates a higher grade player, involves a small program on the videodisc which is automatically downloaded into an onboard microprocessor when the videodisc is first started. This program, in conjunction with the remote controller, offers a slightly easier method of control.

The third method, and the one that most concerns us here, is to use a microcomputer to control the action of the videodisc player. This has the great advantage of allowing an instructional designer to create a complete course for the student, with remediation and branching, that can either be run by the instructor in a classroom setting, or by the individual student at his or her own pace.

Control of the videodisc by a microcomputer offers the highest degree of interactivity. Testing, branching through use of menus to different sections of the program, and remediation can be cleaner and faster through the microcomputer than through the remote controller.

J.A.M., Inc. is a company specializing in interactive videodisc. We have a number of products that run from the Apple //™ and IBM-PC™ series computers. I was asked to transport our software to the Apple Macintosh™. At that time, the only available languages were Apple Pascal (an interpreted Pascal, very limited), Microsoft BASIC, and MacFORTH from Creative Solutions, Inc.

The Macintosh has a reputation as being very difficult to program. The very features that make it so easy to use (windows, pull-down menus, mouse control, graphics, etc.) also make it difficult to program. Porting standard software to the Mac makes that software look extremely dull in comparison to software created to take advantage of the features of the Mac.

Creative Solutions has implemented a version of Forth that accesses all of the Mac's features in a painless fashion. In fact, I was able to implement a demonstration version of our software (using all the Mac's features and controlling a videodisc player) in two weeks!

The first step was to implement the videodisc control words. The player chosen was the Pioneer LD-V6000. Interfacing was accomplished through the Mac's modem port. MacFORTH includes the words **S.EMIT**, **S.EXPECT**, **S.?TERMINAL** (which perform the obvious functions to the serial port) and **S.?READY** to determine if the port is busy. Thus:

```
HEX

: XMIT  ( c -- \ send char to videodisc player )
   BEGIN  S.?READY UNTIL  ( wait for it... )  S.EMIT  ( go! ) ;

: VD> ( n -- \ get n chars from vd player to pad ) ( "vd from" )
   BEGIN S.?TERMINAL UNTIL ( wait for something coming in... )
   PAD SWAP ( set up addr, count )
   S.EXPECT ( move buffer to PAD ) ;

: HALT BF XMIT ;

: GET.STATUS ( -- c \ get status byte from player )
   04 XMIT 1 VD> PAD C@ 7F AND ;

: ?BUSY ( -- f \ get status from player - 0 if stopped )
   GET.STATUS 65 = NOT ; ( fudge for true = busy )

: DELAY BEGIN ?BUSY NOT UNTIL ; ( just wait til player ready )
```

These few words form the basis for the rest of the words needed to control the LD-V6000. Finding out which frame the player is currently showing, initiating a search, or playing a sequence are very simple:

```
: ?FRAME ( -- n \ get frame number from player )
   03 XMIT ( request frame # )
   2 VD> PAD W@ ;
```

```
: VPLAY FD XMIT ;

: #FIND ( n -- \ search to frame n )
   DUP STARTFRAME ! ( save copy in case need to repeat )
   DUP ?FRAME = IF ( are we already there? )
      DROP ELSE ( do nothing )
         SEND.NUMBER F7 XMIT
         BEGIN GET.STATUS 50 = NOT UNTIL ( honest )
   THEN ;
```

The word **SEND.NUMBER** is necessary to turn the frame number on the stack into the sequence of bytes that the LD-V6000 expects.

This program, being a prototype, did not have a complete authoring language implemented. Therefore, I used some specific conventions: any word that returned a value started with a question mark (**?FRAME**, **?BUSY**); any word that the instructional designer might have to use that expected a number on the stack started with a number sign (**#FIND**, **#PLAYTIL**); any word that I needed which was already a MacFORTH word I simply added a "V" to the beginning of (**VPLAY**, **VPAUSE**). This meant that the designer could control the player directly from the keyboard, testing options and making decisions about sequences of motion and still frames with a minimum of fuss, using Forth's immediate mode (Of course, I didn't tell our designers that they were writing programs!):

```
42322 #FIND  43743 #PLAYTIL
50435 #FIND 50647 #PLAYTIL MAIN.MENU
```

Once the designer had decided on the exact sequences he or she wanted, and their order, I would code the final controls into Forth words. Meanwhile, the designer would be creating a standard text file of questions and answers for student testing. In order to make it as simple as possible for the designer, I told them I would worry about formatting to fit the windows. I created some special windows for the questions and answers to give an additional visual interest to the program:

```
 DECIMAL

NEW.WINDOW TEST.W ( create the window for the test questions )
50 20 240 240 TEST.W W.BOUNDS ( where the window will appear )
3 TEST.W W.TYPE ( no title, no close, drop shadow )
NOT.VISIBLE TEST.W W.ATTRIBUTES ( don't show it til I want it )

NEW.WINDOW CORRECT.W ( create the window for the answer )
100 260 217 500 CORRECT.W W.BOUNDS
1 CORRECT.W W.TYPE ( no title, no close, double frame )
NOT.VISIBLE CORRECT.W W.ATTRIBUTES

TEST.W ADD.WINDOW   CORRECT.W ADD.WINDOW
```

The problem with creating programs on the Mac is that you never want to stop adding Mac features. I had already effectively ported the existing capabilities of our other versions, but now I added a game. We were teaching about Local Area Networks, and I added a game in which the student set up an office. He or she would place microcomputers, file servers, printers and dumb terminals in the office, connect them with cables, pick the type of network, then run it. The Mac would test each component of the network, and let the user know if the network had run or not. This section alone would have taken weeks, except that MacFORTH has the capability of loading a MacPaint graphic image into a MacFORTH window. This meant that the introduction, the office, and some additional graphics could be created off-line by me or an artist (note the distinction), and then added at a later date. This gave me the time to work on the game itself.

Each element of the game was fine-tuned to take advantage of the Mac. For example, running cable from one element of the network to another was done through a "rubber band" effect:

```
: RUBBER BAND ( x y -- \ simulate the rband effect )
   BEGIN
      2DUP @MOUSEXY ( starting and ending points )
      3 3 PENSIZE PATXOR PENMODE ( type, size of pen )
      4 PICK 4 PICK 4 PICK 4 PICK
      ( dup start/end pts - not elegant, but it works )
      VECTOR VECTOR ( draw & erase )
      DO.EVENTS MOUSE.DOWN =
   UNTIL ;  ( leave final draw for do.cable )

: DO.CABLE
   @MOUSEXY 2DUP
   WHERE.IS.IT? ( # of rect we're currently in )
   ?DUP ( rect# )  IF  ( we're in a network component... )
      1- ROT ROT RUBBER.BAND ( do the rubber band )
      @MOUSEXY 2DUP WHERE.IS.IT? ( # of rect we ended in )
      ?DUP IF ( we ended in a component... )
         1- 6 ROLL OVER OVER ( get second set for first link )
         SAVE.LINK ( to other end of cable )
         SWAP ( use first set, save in other link )
         SAVE.LINK ( back the other way )
      VECTOR ( ended okay ) ELSE 2DROP 2DROP DROP THEN
   ELSE 2DROP ( not a good start ) THEN ( started okay ) ;
```

The videodisc program was completed in record time. Forth's interactive nature allows me to test different ways of doing things to get just the "look" I want on the Mac. It's interpretive nature allows our instructional designers to map out different sections of the program and test each as they go along. It's speed gives me the ability to control every aspect of the player without resorting to assembly language.

I could not have completed the project in the amount of time I was given using a different language. The number of features available on the Mac, coupled with bringing up a complete instructional package on a new computer would have been overwhelming in a language that did not offer the power and friendliness of Forth.