Interactive NAPLPS (Telidon 709) Graphics on a modern computer
Technical note

John Durno (jdurno@uvic.ca)
University of Victoria Libraries
May 2017
Version 1.0

Introduction

This technical note extends an earlier note covering a similar subject, "Displaying NAPLPS (Telidon 709) graphics on a modern computer." For background on what NAPLPS and Telidon were and why it might be important to be able to render NAPLPS images on a modern computer, please see the introduction to that earlier note.

The previous note described how to obtain and configure software to display a pre-ordered sequence of NAPLPS graphics. While many works from the NAPLPS era were designed to be viewed in just that manner, other works were interactive, allowing the user to alter the viewing order by selecting from a menu of options. NAPLPS was, after all, originally used as an imaging encoding format for Videotex systems, and interactivity was a key feature of Videotex. My earlier note did not specify (or even suggest) how to add (or restore) interactive capability to a NAPLPS display.

This technical note addresses that omission by describing a method for building interactive NAPLPS presentations using readily available shareware, open source software, and a small amount of custom programming.

Background and Overview

I developed this method for displaying interactive NAPLPS graphics during the course of a project to restore a collection of Canadian videotex artworks developed in the early/mid 1980s. The original presentation had used proprietary software called Videophile to manage the sequencing and display of the images, while decoding of the NAPLPS graphics was done in hardware with a Quickpel expansion card for the IBM PC.[1] Neither Videophile nor the Quickpel card could be sourced in the present day.

As described in the previous NAPLPS tech note, dedicated hardware is not necessary for decoding NAPLPS graphics. In the early days of Telidon and NAPLPS special-purpose decoding devices were required because consumer grade computers of that era were too underpowered to do the job. However, during the mid-to-late 1980s consumer-grade computing hardware became powerful enough to decode and display NAPLPS graphics natively. A number of companies developed software for that purpose, and at least one software decoder from the NAPLPS era is still readily available, thanks to its persistence in the Simtel shareware archive: Personality Plus III (PP3), developed by MicroStar. We can use PP3 to substitute for the capabilities of the Quickpel card.

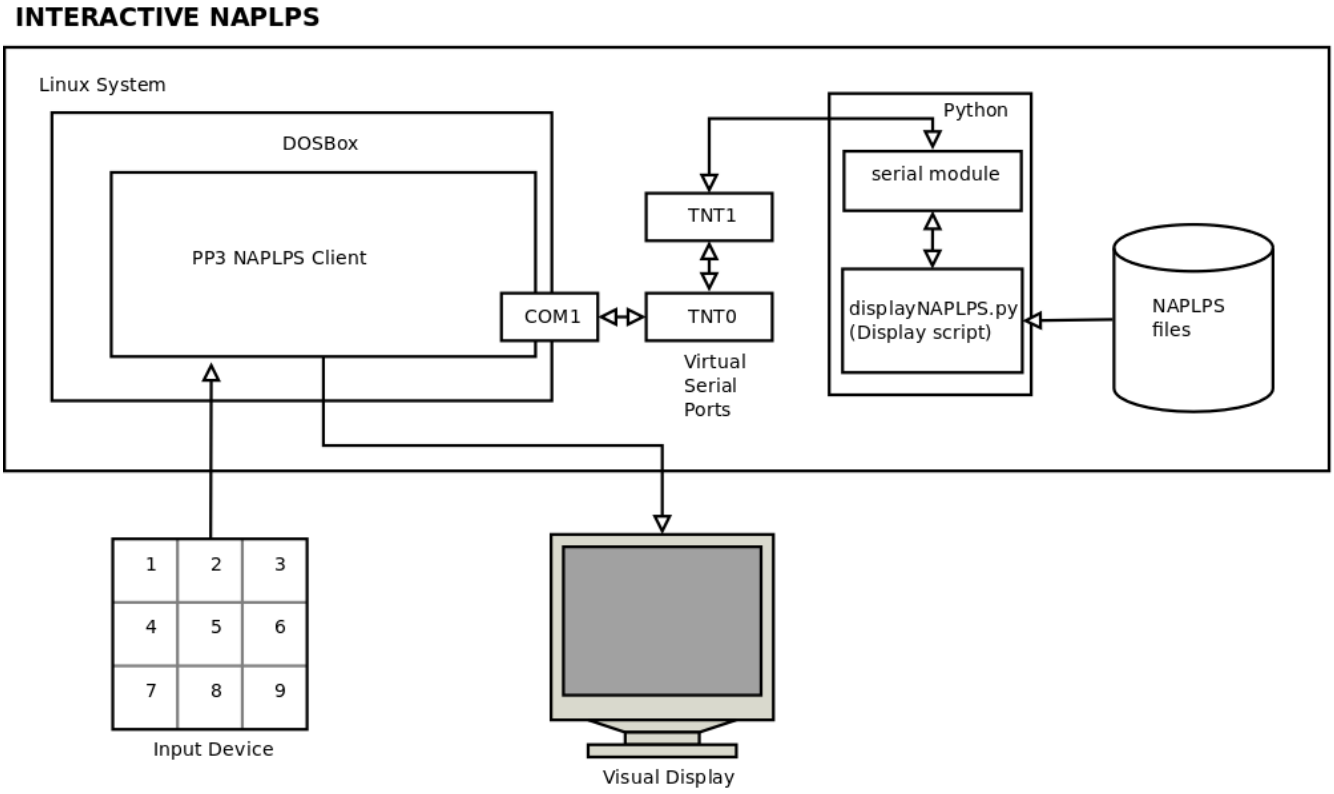
There is no software that can replicate the functionality of Videophile out of the box. However, the logic for managing an interactive display is fairly trivial, and can be quite readily coded in any common scripting language. Python was our tool of choice here.

One must of course address the question of how the PP3 client communicates with the python display script. Fortunately PP3 was developed as a client for bulletin board systems, so has built-in functionality to communicate over an RS-232 serial connection. Python has a well-developed serial

1 For more information on Videophile and Quickpel, see D. McCune, Build a NAPLPS system using Quickpel, Createx and Videophile together. Microsystems vol. 5 issue 7 July 1984

communication module, enabling us to code a lightweight NAPLPS server with a minimum of overhead.

Back in its day users of PP3 would typically have established serial connections via telephone modem. One could still do it that way of course, but unless the server and client are separated geographically it makes more sense to use a null modem connection. This can either involve a null modem cable if connecting two physical devices, or (what is described here) using a null modem emulator to enable both client and server to run on the same physical hardware. The following diagram illustrates the various components of the system we will be assembling and how they intercommunicate:



Technical Details

This technical note describes setting up interactive NAPLPS on an Ubuntu Linux computer. Somewhat greater technical knowledge is both required and assumed than was the case for the previous NAPLPS technical note. In particular, knowledge of the Unix command line environment is assumed, as is the ability to compile software. Some knowledge of Python scripting is also helpful. This document attempts to provide enough information for you to be able to look up details for yourself; it does not exhaustively describe everything you need to know. For example, it will tell you to add the Ubuntu Universe repositories; it won't tell you what those are or how to add them, as that information is readily available elsewhere.

In theory, it should be possible to set up interactive NAPLPS displays on alternative operating systems (Windows, Mac OS) using the same approach as described here, although different null modem emulators will be required (eg. com0com on Windows), and the serial port settings in DOSBox will need to be accordingly modified. The other software components (DOSBox, Python) are cross-platform compatible. All software is freely available either as open source or (in the case of PP3) shareware.

Install and configure tty0tty

tty0tty is a null modem emulator written for Linux systems. It is available from:

https://github.com/freemed/tty0tty .

It is a kernel module, so the installation process is not quite the same as for standard software programs.

Download the source for the module from github and follow the installation instructions in the file README.md in the top level directory. (Do NOT refer to the file INSTALL, which is curiously incomplete).

Note that the module will not stay active across reboots unless you follow the instruction to edit /etc/modules.conf (or /etc/modules in Debian). Even if you do modify the appropriate file, the permissions will still need to be reset after every reboot (sudo chmod 666 /dev/tnt*) unless additional steps are taken (beyond the scope of this note). Also, note that the whole installation process will need to be repeated after every kernel update.

As noted in the documentation, tty0tty creates 8 interconnected (emulated) serial ports. The serial ports are paired in ascending order, for example /dev/tnt0 is paired with /dev/tnt1, /dev/tnt2 is paired with /dev/tnt3, and so on. We will be using /dev/tnt0 and /dev/tnt1, the other port pairs will be ignored in this document.

To test whether your installation was successful, at the command line enter:

```
ls -l /dev/tnt*
```

Check to see there are no errors, and the permissions match the following:

```
crw-rw-rw- 1 root root 240, 0 Apr 13 11:16 /dev/tnt0
crw-rw-rw- 1 root root 240, 1 Apr 13 11:16 /dev/tnt1
crw-rw-rw- 1 root root 240, 2 Apr 13 11:16 /dev/tnt2
crw-rw-rw- 1 root root 240, 3 Apr 13 11:16 /dev/tnt3
crw-rw-rw- 1 root root 240, 4 Apr 13 11:16 /dev/tnt4
crw-rw-rw- 1 root root 240, 5 Apr 13 11:16 /dev/tnt5
crw-rw-rw- 1 root root 240, 6 Apr 13 11:16 /dev/tnt6
crw-rw-rw- 1 root root 240, 7 Apr 13 11:16 /dev/tnt7
```

Install DOSBox

DOSBox is a DOS emulator optimized for running old computer games, available from:

http://www.dosbox.com/

It is required because the NAPLPS client PP3 was written for the DOS operating system. DOSBox also provides important supporting functionality as detailed below (see “Configure DOSBox”).

Installers and documentation are readily available on the site. To install DOSBox on Ubuntu, first add the Universe repositories if these are not already on your system. Then open up your terminal and type:

```
sudo apt-get update
sudo apt-get install dosbox
```

Install PP3

PP3 is availble in various mirrors of the old Simtel archive that are scattered about the web. For example, it is currently (as of April 2017) available from:

http://cd.textfiles.com/simtel/simtel20/MSDOS/NAPLPS/.index.html

Other mirrors exist; if the mirror above is no longer available google PP32317A.ZIP (the file name) to locate a copy.

There is no PP3 installer, it can simply be unzipped into any directory you have read/write access to. Make a note where you installed it, we will need this later. For the purposes of this tech note we will unzip PP3 into the directory */home/demo/naplps/pp3217a*

Configure DOSBox

Run DOSBox from the terminal by typing:

```
dosbox
```

The first time it runs, DOSBox will create a default configuration file in your home directory, in a hidden folder called .dosbox. The config file is named *dosbox-<version>.conf.* So the config file for version 0.74 (the most recent at the time of this writing) is called *dosbox-0.74.conf*.

The following parameters in the configuration file will need to be modified in a text editor.

First, the display size will likely be too small unless it is modified. There are several ways to do this. Here is one example:

```
fullresolution=1680x1050          #the full resolution of your monitor
windowresolution=1024x768         #a reasonable size for the DOSBox window on your system
output=openglnb                   # the default (surface) doesn't let you specify windowresolution
```

If this doesn't result in what you want, see the comments in the DOSbox conf file for more options.

Second, the processor speed will need to be set low to emulate the rendering speed of early-mid 80s computing hardware. 400 cycles is an approximation. In the 1980s NAPLPS graphics were displayed on a variety of platforms; it is reasonable to assume rendering speeds varied even back then depending on the hardware and software being used. Adjust as you see fit.

```
cycles=400              #needs to be low for accurate rendering speed
cycleup=1               #small increments for adjustment
cycledown=1
```

Third, one of the DOSBox serial ports will need to be mapped to one of the tty0tty serial ports (/dev/tnt0). See the section on tty0tty above for more information.

```
serial1=directserial realport:tnt0 rxdelay:3000      #maps COM1 to emulated port TNT0
```

Finally, add the following lines under [autoexec] at the bottom of the file:

```
MOUNT C /home/demo/naplps/pp3217a          #or wherever you put the PP3 application
c:
```

These lines mount the directory where you installed PP3 as DOSBox's virtual C: drive, and make that the active drive.

Configure PP3

Start DOSBox and confirm that your prompt is C: (not Z:) and that you are where you expect to be, in the directory where you unzipped the PP3 files. Type dir /w at the DOSBox command prompt and you should see a list of files that looks like this:

```
C:\>dir /w
Directory of C:\.
[.]             [..]            CGA640.SCR      EGA640.SCR      EGA640E.SCR
EGA640F.SCR     ET4000ZF.SCR    FILE_ID.DIZ     HER720.SCR      LCD640.SCR
MCGA320.SCR     MVDIAPP.COM     NAPLPS.000      NAPLPS.001      NAPLPS.002
NAPLPS.003      NAPLPS.004      NAPLPS.005      NAPLPS.006      NAPLPS.007
NAPLPS.008      NAPLPS.009      NAPLPS.010      NAPLPS.011      PP3.BAT
PP3S.DAT        PP3SCOMM.EXE    PP3SE.MSG       PP3SET.BAT      PP3SF.MSG
README.1ST      README.PP3      REGISTER.PP3    T3100.SCR       TARGA16.SCR
TRS1000.SCR     VDA256.SCR      VGA320.SCR      VGA640.SCR
   37 File(s)            463,810 Bytes.
    2 Dir(s)       262,111,744 Bytes free.
```

Before you can launch PP3, you will need to specify its graphics driver. To set the graphics driver, type

```
pp3set e vga640
```

This setting will persist across restarts, so you shouldn't need to do it more than once.

Then to start pp3, type

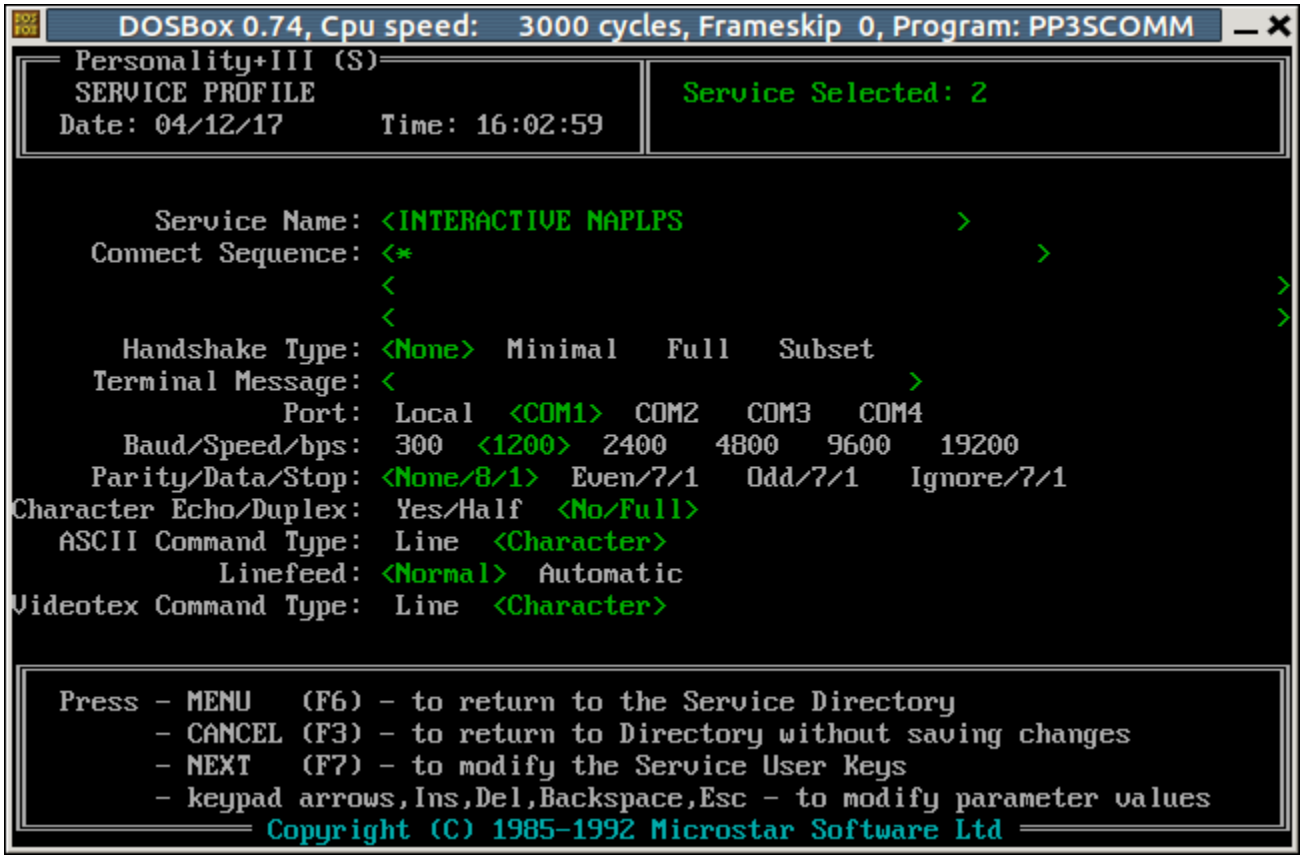
```
pp3
```

After the initial splash screen loads, you will see a service directory with several connection profiles for old Bulletin Board systems. We will need to replace one of those profiles with our own that points to our local system. Here we will select option 2 (Harris Technology Associates). Type 2 to select the profile, followed by F7 to edit it.

Modify the entry as follows:

```
Service Name: INTERACTIVE NAPLPS
Connect Sequence: *
Handshake Type: None
Terminal Message:
Port: COM1
Baud/Speed/bps: 1200
Parity/Data/Stop: None/8/1
Character Echo/Duplex: No/Full
ASCII Command Type: Character
Linefeed: Normal
Videotex Command Type: Character
```

Here is what it should look like when you are done:



Type F6 to save your changes, and F9 to exit PP3. Close the DOSBox window.

Now, return to the DOSbox configuration file you edited in the previous section. At the end of the file, under 'c:' add the following line

```
pp3 2
```

This line will start pp3 automatically after DOSBox starts, and initiate service 2, the one you just defined.

NAPLPS display script

As noted above, the display logic can be programmed in your language of choice, but it helps to choose one with a good serial communication library. Python is what we are using here, so you will need to have the python interpreter installed on your computer. Most linux and Mac systems have python installed by default.

A sample display script (called "displayNAPLPS.py") should have been distributed along with this technical note, along with a few NAPLPS demo files you can use to test your setup.

At the Unix command prompt, start displayNAPLPS.py by entering:

```
python displayNAPLPS.py
```

It will attempt to establish a connection to the serial port /dev/tnt1, which as noted above is paired with the serial port /dev/tnt0. If it is successful, it will begin listening on that port. If it is unsuccessful, it is likely that either tty0tty is not working, or permissions have not been set correctly.

If displayNAPLPS.py receives the character * at this point it will begin the NAPLPS presentation. Because you specified * as the Connect Sequence in the INTERACTIVE NAPLPS directory entry in PP3, your NAPLPS exhibit should start automatically when DOSBox starts up.

NAPLPS presentations can advance in one of three ways after a frame has rendered:
1. The system automatically advances to the next frame [auto=Y]
2. The system waits for user input before advancing to the next frame [auto=N]
3. The system waits for a specified interval for user input. If no input is received, it automatically advances to the next frame. [auto=M]

For simplicity's sake, user input in the demo always takes the form of single digit integers entered from a keypad.

Timings: Setting appropriate delays between frames can sometimes be critical, but setting delays is complicated by the fact that the display server only knows when it has finished transmitting a file; it has no way of knowing when a NAPLPS file has finished rendering on the client. This cannot be estimated exclusively by file size, as files of the same size may render faster or slower depending on the relative complexity of their content. Wait times must therefore be pre-established by recording the rendering times of each NAPLPS frame (which may vary from system to system) in advance.

File size: Very large (>20K) NAPLPS files can sometimes fail to render correctly, likely due to serial buffer overflows. To address this issue displayNAPLPS.py provides the option of sending large files in smaller chunks with variable delays in between. The same considerations discussed in "A note on timings" above also apply here.

Serial Connection Settings: The script settings must match the settings (baud rate, parity/data/stop bits) you set up in the PP3 service directory. See “Configure PP3” above.

File attributes

Behaviours of each frame (NAPLPS file) in the presentation are established by associating attributes with files in a python dictionary near the top of displayNAPLPS.py. Attributes include:

fn: name of file to load
auto: auto-advance to the next slide? Values Y, N, M [Yes/No/Maybe]
goto: command to invoke for auto-advance, see cmd parameter below. Does nothing if auto is N
renderwait: interval to pause after file is sent, in seconds
inputwait: how long to wait for input when auto=M
chunk: number of bytes to send at one time, 0=send whole file with no breaks
chunkwait: interval before sending the next chunk in seconds, 0=no wait. Does nothing if chunk is 0
cmd: nested dict of one or more commands indicating file to display next

So, for example, the attributes of the first frame in the Demo are:

```
j['idemo010']['fn'] = 'idemo.010'
j['idemo010']['auto'] = 'N'
j['idemo010']['goto'] = '1'
j['idemo010']['renderwait'] = 4
j['idemo010']['inputwait'] = 0
j['idemo010']['chunk'] = 0
j['idemo010']['chunkwait'] = 0
j['idemo010']['cmd']['1'] = 'idemo021'
j['idemo010']['cmd']['2'] = 'idemo022'
```

From this, we can see that the dictionary entry is labelled 'idemo010' and the associated filename is 'idemo.010'. It does not auto-advance (auto='N') and it pauses for 4 seconds after the file is sent to the client before it will accept user input. As auto is set to ‘N’ it will wait forever for input, therefore an ‘inputwait’ interval will be ignored (it is here set to zero). The file is not sent in chunks (ie the whole file is sent as a single continuous stream of bits with no pauses). After the file renders the script will accept either of two specified commands ('1' and '2' respectively) that specify which frame to load next. For example, if the user enters '2' the next file that displays will be the file associated with dictionary entry idemo022.

Here is an example of a frame in the Demo that auto advances to the next one:

```
j['idemo021']['fn'] = 'idemo.021'
j['idemo021']['auto'] = 'Y'
j['idemo021']['goto'] = '1'
j['idemo021']['renderwait'] = 6
j['idemo010']['inputwait'] = 0
j['idemo021']['chunk'] = 0
j['idemo021']['chunkwait'] = 0
j['idemo021']['cmd']['1'] = 'idemo030'
```

In this example the dictionary entry is labelled ‘idemo021’ and the associated filename is ‘idemo.021’. [This is the frame you would have advanced to had you entered ‘1’ for the previous example.] It is set to auto-advance (auto=’Y’). The system will pause for 6 seconds (renderwait=6) after it finishes transmitting the file ‘idemo.021’ before it automatically begins transmitting the file associated with dictionary entry ‘idemo030’. Note that the goto value (‘1’) matches the only available cmd value. The go-to value must match one of the cmd values otherwise auto-advance will fail.

Here is an example of a frame in the demo that auto-advances if input is not received within a given interval:

```
j['idemo024']['fn'] = 'idemo.024'
j['idemo024']['auto'] = 'M'
j['idemo024']['goto'] = '2'
j['idemo024']['renderwait'] = 1
j['idemo024']['inputwait'] = 10
j['idemo024']['chunksize'] = 0
j['idemo024']['chunkwait'] = 0
j['idemo024']['cmd']['1'] = 'idemo025'
j['idemo024']['cmd']['2'] = 'boom'
```

Auto advance is set to ‘M’ (“maybe”, meaning it will auto-advance if the user does not input a command within the interval specified by renderwait and inputwait added together). If the presentation does auto-advance, it will send the default specified by ‘goto’, command value 2. The display will only pause one second after rendering before it starts accepting user input, and it will wait 10 seconds before it stops listening for input and sends the default file. Command value 2 indicates the dictionary entry ‘boom’.

Here is an example of a file sent in chunks:

```
j['idemo023']['fn'] = 'idemo.023'
j['idemo023']['auto'] = 'Y'
j['idemo023']['goto'] = '1'
j['idemo023']['renderwait'] = 5
j['idemo023']['inputwait'] = 0
j['idemo023']['chunksize'] = 78
j['idemo023']['chunkwait'] = 6
j['idemo023']['cmd']['1'] = 'idemo024'
```

Normally one would not send a file this small (111 bytes) in multiple chunks. As noted above however, larger files (>20K, which is big for NAPLPS) can sometimes error out, likely due to serial buffer overflows. In such cases it is helpful to be able to insert delays in order to allow the client sufficient time to process the incoming serial data. In this example the first 78 bytes are sent (specified by ‘chunksize’), followed by a 6 second wait (specified by ‘chunkwait’), followed by the rest of the file (since the second chunk is smaller than 78 bytes).