R W Witty

my DIC file

SCIENCE AND ENGINEERING RESEARCH COUNCIL
RUTHERFORD APPLETON LABORATORY

INFORMATICS DIVISION

D I S T R I B U T E D I N T E R A C T I V E C O M P U T I N G N O T E 9 7 1

issued by
A J Kinroy

DISPLAY Performance Analysis

3 July 1984

DISTRIBUTION: SEG
A S Williams
J Haswell

(see next page)

Changes to display May/June 1984

A study of the performance of the program was undertaken to see if it could be determined what was taking the most time and whether the speed of output could be improved.

Step 1 - Procedure call counting

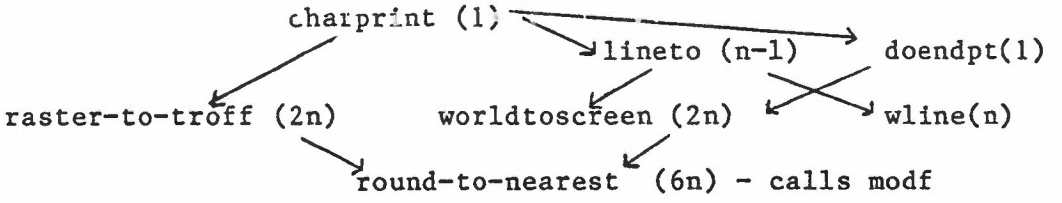
By implementing a general procedure which incremented a variable whose address was passed to the procedure a count of the number of times each procedure was called was obtained.

The results obtained from a sample of V7 troff output were:

Procedure	no of calls	calls per character
POINTSIZE	9	
SETLEADING	29	
CONTROL	112	
FONTPOSITION	468	
CHARPRINT	468	
TROFFCHAR	471	
SETESCAPE	575	
DOENDPT	1884	4.02
MOVETO	2352	5.02
LINETO	9354	19.98
WORLDTOSCREEN	20592	44.00
RASTER-TO-TROFF	24349	52.02
ROUNDTONEAREST	65535	140.03

Results from t.i.troff output were in similar proportions.

Theoretically if a character was considered as one continuous line segment the procedure call tree would look like this:



where n is the number of Hershey points per character (typically 20)

The times for the above example as given by time (1) were:

```
real 1:08.0  (68)
user   49.5
sys    12.3
```

As a first attempt at reducing the number of procedure calls the raster-to-troff and worldtoscreen procedures were replaced by macros.

The times after this change were:

```
real 1:04.0 (64)
user   45.5
sys    12.4
```

This is an improvement of about 6%.

Step 2 - Accounting for the time

From an article in PERQNEWS 4 the time to make a wline procedure call was given as 1 ms when buffering is used.

In the example there are a total of 9354+1884=11238 wline calls which would account for 11.238 of the 12.4 secs of system time.

To find out how much time was being taken in reading the 4 default font files for V7 troff some examples of minimal troff input (eg .sp) were used. The times obtained for this were:

```
real 11
user  3.2
sys   1.9
```

To reduce the time in reading files when different fonts were selected it was decided to bind in the font data at compile time.

The implementation details of this are described later.

The times obtained for the same minimal troff input after the fonts had been compiled in were:

```
real 10
user  0.1
sys   2.0
```

Thus the start-up time due to reading files has been exchanged for a slow start-up time until the program has loaded but there will not be any time dealays in switching to a different font with the new implementation.

The times for the original example with all the fonts compiled in were:

```
real 1.03  (63)
user   42.6
sys    11.9
```

In an attempt to determine where the user time of the program was being spent some timings with the ftime system call were taken. The results from this were inconclusive as trying to time procedures causes extra overheads in system calls thus ruining the accounting of user and system time by the time command.

In an attempt to give some visual indication of how long was being spent in each procedure a set of areas on the screen were reserved for blacking the area when the procedure was entered and clearing the area when the procedure was exited. For frequently executed short procedures this shows as a rapid flickering, for larger procedures the black area will be more persistent. When this was implemented the results it gave were qualitative rather than quantative. As expected the 'charprint' area flickered intermittently and the 'roundtonearest' area flickered continuously. An area for the main while loop in the charprint routine was also displayed. This appeared black quite a large proportion of the time indicating a lot of time was spent in this loop.

An attempt was made to account for the 42 secs of user time by totaling up operation and procedure call times given in CBP Assessment Note 10: Results of Benchmarking UNIX systems. I managed to account for 4.3s in if-else tests, while loops assignments and procedure call overheads, and as a result of my own timings 9.5s in modf function calls making a total of 13.8 out of 42.6s.

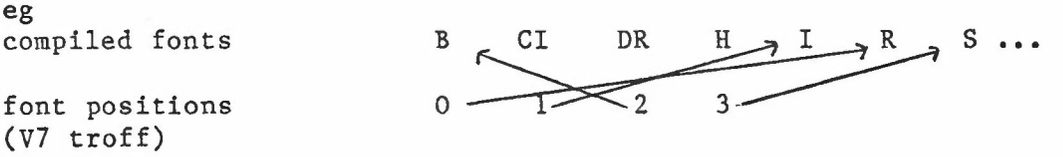
It is obviously unsatisfactory that so much of the user time is still unaccounted for but in the absence of execution profiling on PNX it has not been possible to proceed further.

Binding in compiled fonts

The original version of display read Hershey packed and index files into memory for accessing fonts. In the case of V7 troff output the four fonts selected at the start of the page were held in memory, for t.i.troff output the typesetter description file (which contains width and code information for currently mounted fonts) and the Hershey files for the currently selected font only were held in memory. This meant there was a visible delay in reading font-files into memory if there was a change of mounted fonts in V7 troff or a change in selected font in t.i.troff. To overcome this it was decided to compile in the Hershey font information.

As the fonts and their ordering in the statically defined tables now has to be fixed it was necessary to introduce another table to index the fonts. This table is set at run-time by interpreting the information in the troff output which determines the fonts that should be associated with each font position.

eg



T.i.troff has font positions numbered 1 to n for general use and 0 for special (overlaid) use. The indexing by this second table is exactly the same.

The array in which the pointer indexes are held is called f-actual. It is used to index tables of font abbreviations (fontabbv), font names (mntfnts) and the Hershey points and indices (combined-font).

In order to make initialised C files of the Hershey characters and indices the program ftupack was modified to make a program ftcom which will read the packed & index files for a particular font and generate an initialised C file containing the font data.

DIC44/jg