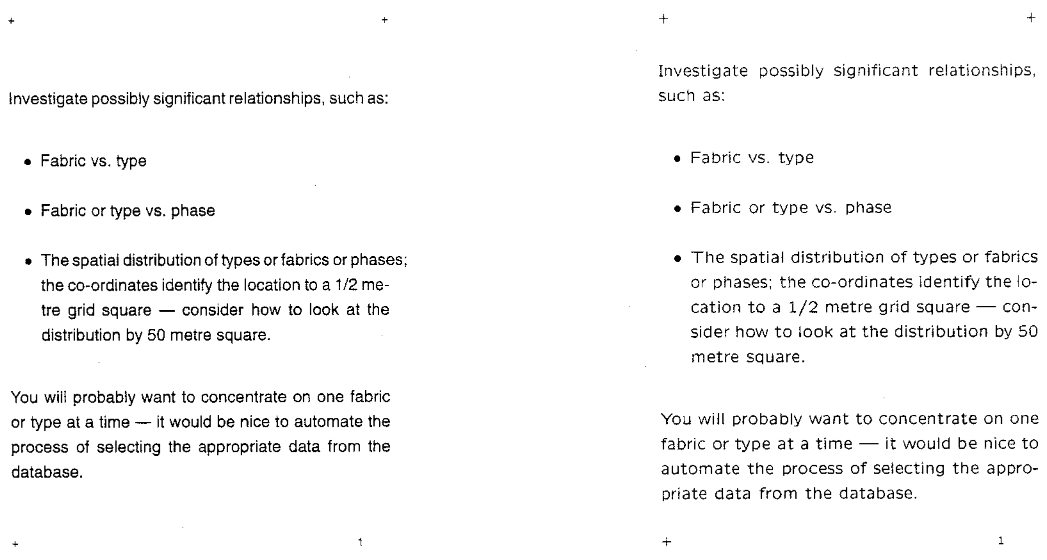

Investigate possibly significant relationships, such as:

- Fabric vs. type
- Fabric or type vs. phase
- The spatial distribution of types or fabrics or phases; the co-ordinates identify the location to a 1/2 metre grid square — consider how to look at the distribution by 50 metre square.

You will probably want to concentrate on one fabric or type at a time — it would be nice to automate the process of selecting the appropriate data from the database.

1

Helvetica

Investigate possibly significant relationships, such as:

- Fabric vs. type
- Fabric or type vs. phase
- The spatial distribution of types or fabrics or phases; the co-ordinates identify the location to a 1/2 metre grid square — consider how to look at the distribution by 50 metre square.

You will probably want to concentrate on one fabric or type at a time — it would be nice to automate the process of selecting the appropriate data from the database.

1

Computer Modern

Figure 1: The effect of various fonts in slides

original SLiTEX. Future releases of the font selection macros are likely to feature this system; combined with the virtual font suggestions outlined above, one can envisage a long-overdue renaissance for SLiTEX.

To demonstrate the effect of the virtual font approach in these pages would be difficult, but it may be of interest to readers to see the visual effect (albeit reduced) of SLiTEX slides set in Helvetica rather than the familiar hugely expanded Computer Modern (Fig. 1).

This article has benefited considerably from comments by the *TUGboat* reviewer.

References

[Knuth 1990] KNUTH, D. 1990. 'Virtual fonts: more fun for grand wizards', *TUGboat* 11, no. 1, pp. 13–23.

[Lamport 1986] LAMPORT, L. 1986. *LaTeX User's Guide & Reference Manual*, Addison-Wesley Publishing Inc., Reading, Massachusetts.

[Mittelbach & Schöpf 1990] MITTELBACH, F. AND R. SCHÖPF 1990. 'The new font family selection—User interface to standard LaTeX', *TUGboat* 11, no. 1, pp. 91–97.

◇ Sebastian Rahtz
ArchaeoInformatica
5 Granary Court
St Andrewgate
York Y01 2JR
U.K.

Packing METAFONTs into POSTSCRIPT

Toby Thain

Aimed at implementors of DVI-to-POSTSCRIPT translators, this article suggests adapting Rokicki's packed font format [1] to compactly define bitmap fonts in POSTSCRIPT, an approach which has been successfully implemented and tested by the author.

The problem of integrating METAFONT and POSTSCRIPT has been tackled in two completely different ways: by modifying METAFONT to output curvilinear paths and outlines [4, 5], and by using METAFONT's standard bitmap output directly. Since POSTSCRIPT allows flexibility in representation, the choice is largely philosophical. While outlines are less device-dependent and more amenable to linear transformations, this author feels that TEX users need an effective means of using METAFONT-generated bitmaps with the gamut of POSTSCRIPT devices.

Another consideration is that METAFONT's digitisation is likely to be better than that produced from a machine-translated outline font; current POSTSCRIPT printers are notably lax in this regard. (Adobe Type Manager is a significant improvement, but printers do not yet incorporate this renderer, resulting in the irony that some non-POSTSCRIPT printers using ATM render text better than many POSTSCRIPT printers.) In short, where low-resolution devices are concerned, the author believes that METAFONTs such as Computer Modern

Name	dpi	GF	PK	PS	VM
cmr10	300	13 040	5352	8370	14 948
"	600	24 124	10 808	13 919	20 404
"	1270	49 468	27 576	30 983	37 134
"	2540	108 828	59 492	62 424	68 360
cmsy10	2540	121 548	73 992	76 951	82 840
pkdict				2091	6846

Roman are better digitised by METAFONT than by PostScript from an outline. (This is also apparent from examples presented in [5].)

It is clear from the table that GF-format glyphs are excessively large at high resolutions, and even low-resolution fonts are cumbersome if no compression is used. Another constraint is the limited 'virtual memory' available to PostScript devices; all fonts in a document typically coexist in this space, and therefore it is important to manage it efficiently (the column headed 'VM' indicates how much of this memory is used by each packed font.) A compressed representation is also desirable where fonts are stored in a PostScript device's local file system.

This wastefulness suggests using a compressed description which could be interpreted by the printer itself. Having the advantages of high compression ratios and well-documented algorithms, PK format seemed a natural choice. Furthermore, existing TeX fonts are largely in PK format.

An implementation of this idea is shown in Figure 1, encapsulated in a dictionary named pkdict. Most of the work is done by the bc procedure; what remains is to show how specific fonts are arranged to make use of that code. (These examples, compromises between clarity and brevity, can be improved in either direction.)

An example

A bitmap font dictionary might be defined as follows (automatically generated from logo10.300pk):

```
/logo10 pkdict begin 10 dict dup begin
/FontType 3 def
/FontMatrix [26214400 109226469 div
   0 0 2 index 0 0] def
/Encoding ev def
/BuildChar /bc load def
/glyphs 128 array
77 [27 25 -3 24 33 0 t 12
   <f3d87d69fd4bfd29f13d031623b32624a326339336347346
   437346f535356f63336673137677768586f9396fd830> p
69 [20 25 -3 24 26 0 t 11 <dfeac5fc62e73c5dc0> p
84 [23 25 0 24 23 0 t 10 <d3ebba3a> p
65 [22 25 -3 24 28 0 f 12
   <5c8d3558543c33f3d131e63d3025e9d330> p
70 [20 25 -3 24 26 0 t 11 <dfeac5fc62e93c50> p
79 [26 25 -1 24 28 0 f 12
   <7cbd5768654d1443d333f3d531eb3d73f13d5324d3436a65
   d79d16> p
78 [22 25 -3 24 28 0 t 12
   <3d37d28d19d0ac613c623b624a6349644865476646674568
   4469436a426b326c316cad09d18d27e2d330> p
def
/FontBBox [0 0 30 25] def
end end definefont
```

```
/pkdict 11 dict dup begin /a [0 128 192 224 240 248 252 254] def
/p { ] 2 index 3 1 roll put} bind def /f /false load def /t /true load def
/r {string currentfile exch readstring pop} bind def /ev [0 1 255 {( ) dup 0 4 -1 roll put cvn} for] def
/gn {d p get hi {-4 bitshift /hi false def} {15 and /p p 1 add def /hi true def} ifelse} bind def
/gb {d p get i and 0 eq /i i dup 1 eq {/p p 1 add def pop 128} {-1 bitshift} ifelse def} bind def
/pb {{j or} if j 1 eq {l q 3 -1 roll put /q q 1 add def 0 128} {j -1 bitshift} ifelse /j exch def} bind def
/pn { gn dup 0 eq { {1 add gn dup 0 ne {exit} {pop} ifelse} loop
    exch {4 bitshift gn or} repeat 15 sub 13 df sub 4 bitshift df add add
  } { dup df gt { dup 14 lt {df sub 1 sub 4 bitshift gn add df add 1 add}
    {14 eq {pn} {1} ifelse /r exch def pn} ifelse } if } ifelse } bind def
/bc { save 3 1 roll exch /glyphs get exch get pkdict begin 16 dict begin aload pop
  /d exch def /df exch def /c exch def /w 6 index def
  3 index neg 3 index 6 index sub 1 add w 6 index sub 5 index 1 add setcachedevice
  false [1 0 0 -1 8 -2 roll 1 add] /p 0 def
  w 7 add -3 bitshift dup string /l exch def dup string /br exch def
  dup string exch 1 sub 0 1 3 -1 roll {1 index exch 255 put} for /wr exch def
  df 14 eq {/i 128 def {/j 128 def /q 0 def 0 w {gb pb} repeat j 128 ne {l q 3 -1 roll put} {pop} ifelse l}}
  { /r -1 def /hi true def /n 0 def
    { r 0 lt { /r 0 def /k 0 def /v 0 def /q 0 def /b w def /n n { dup 0 eq {pop pn /c c not def} if
      dup b lt dup {1 index} {b} ifelse dup
      k 0 ne { dup 8 k sub 2 copy gt {exch} if pop c {dup a exch get k neg bitshift v or /v exch def} if
        dup k add dup 8 eq {l q v put /q q 1 add def pop 0} if /k exch def sub } if
      dup 0 ne { dup -3 bitshift dup 0 ne { l q c {wr} {br} ifelse 0 4 index getinterval putinterval
        q add /q exch def 7 and } {pop} ifelse c {dup a exch get} {0} ifelse /v exch def k add /k exch def
      } {pop} ifelse exch {dup b exch sub /b exch def sub} {k 0 ne {l q v put} if sub exit} ifelse
    } loop def } if /r r 1 sub def l } } ifelse imagemask end end restore } bind def
end def
```

Figure 1. It looks complicated, but it works!

Such Type 3 fonts require a BuildChar procedure to draw glyphs. The Encoding array trivially maps the 256 possible character codes to 256 different names (required by the font cache). The FontMatrix defines the resolution; the font's pixels and measurements such as bounding boxes and widths are transformed by this matrix to 'user space'. glyphs is an array mapping character codes to 'packets' describing the character:

[*w h h_off v_off dx dy f dyn_f data*].

f is true if the first run of pixels is black, false otherwise, and *data* consists of the nybble-packed glyph description. The other entries correspond to PK character packet fields.

After the character descriptions comes the font's bounding box, which is computed by the PK-to-POSTSCRIPT translator.

Finally, the BuildChar procedure (which takes its definition from bc in pkdict above) is executed by the interpreter when a character from this font is needed. In short, bc looks up the character's description given the font dictionary and a character code, passes the width and bounding box to set\-cache\-device (thereby requesting that the character be cached), and draws the glyph one row at a time using imagemask. For further information on the structure of user-defined POSTSCRIPT fonts, see [2].

POSTSCRIPT's font cache ensures that each different glyph drawn is only decompressed once; subsequent requests for the same glyph bypass BuildChar and are satisfied by the cache. Only glyphs which are drawn are decompressed.

It is worth noting that some device-independence comes 'for free' with POSTSCRIPT; for example, bitmap fonts defined in this manner can be rendered at any resolution, under any linear transformation; hence, if bitmaps are not available for a specific resolution or magnification, those at another resolution can be substituted (albeit with less pleasing results).

Character positioning is an issue which can be dealt with on two levels. METAFONT records two different dimensions for each character: a rounded 'displacement' in pixels, and a 'TFM width' for typesetting purposes. Two considerations conflict when positioning characters: it is desirable that characters are spaced according to their TFM width, so that margins line up; and yet integral displacements will yield more even spacing (an issue addressed in more detail by [6]). The approach taken by this POSTSCRIPT representation is that characters are spaced by displacement, therefore the DVI-to-POSTSCRIPT translator will occasionally need to compensate for accumulated rounding errors according to [6].

If fonts are to be stored on a printer's local disk (freeing the host of the responsibility of managing them), a file usr/pkdict should be created containing the definition of pkdict above. Each font file should begin with

```
/pkdict where {pop} {(usr/pkdict) run} ifelse
```

which executes this file if necessary, rather than defining the dictionary in each font file. Font file names are conventionally prefaced by 'fonts/' (e.g., fonts/logo10). Local font files are executed by findfont to define the font dictionary, and therefore consist of the same text that would be downloaded 'on the fly' (see [3]).

Finally, the author would welcome discussion of the techniques and issues presented in this article, and any aspects of the integration of TEX and POSTSCRIPT. The author has also ported TEX 3·0, METAFONT 2·0 and associated tools to the Inmos T800 Transputer.

References

1. T. Rokicki, "Packed (PK) font file format," *TUGboat* **6**(3), pp. 115–120 (1985).
2. *POSTSCRIPT Language Reference Manual*, Adobe Systems Inc., Addison-Wesley (1985).
3. *POSTSCRIPT Language Program Design*, Adobe Systems Inc., Addison-Wesley (1988).
4. J.D. Hobby, "A METAFONT-like System with POSTSCRIPT Output," *TUGboat* **10**(4), pp. 505–512 (1989).
5. S. Yanai & D.M. Berry, "Environment for Translating METAFONT to POSTSCRIPT," *TUGboat* **11**(4), pp. 525–541 (1990).
6. D.E. Knuth, DVItype.web (1982–90).

◇ Toby Thain
RMB 712
Beaufort
Victoria 3373
Australia
Ph. +61 53 497296
Fax +61 53 497339