Fonts

Invisibility using virtual fonts

Sebastian Rahtz

Abstract

The SLiTEX 'invisible' fonts are currently produced by a special set of METAFONT files; an alternative method of generating 'invisible' versions of any font is presented, using virtual fonts.

1 Introduction

As soon as Donald Knuth announced [Knuth 1990] that 'virtual fonts' would be an official part of the TEX version 3 distribution, I started to think of ways in which they could make life easier for me. Now that drivers exist which support the new font format (I have used Tom Rokicki's `dvips` for PostScript output, and Eberhard Mattes' driver family distributed with emTEX for screen previewing), I have been able to realize some of these ideas. In a forthcoming article for the journal of the UK TEX Users Group, *Baskerville*, I discuss the use of virtual fonts in a PostScript environment to implement the suggested extended font layout for TEX; here I look at a very simple use of virtual fonts to generate the invisible fonts needed by SLiTEX.

SLiTEX needs a set of fonts which produce no output, but use up the same amount of white space as the main fonts, in order to produce overlays [Lamport 1986, pp. 136–37]. Leslie Lamport achieved the desired effect by taking the set of fonts he was using for SLiTEX anyway, and altering the METAFONT source so that they produced no marks on the paper but had the same metrics as the parent font. Despite continued complaints that these fonts are not found in all LATEX distributions, the system works well, provided that one sticks to the default fonts. The big disadvantage, of course, is that if one uses a different typeface one has to go right back to METAFONT sources (which may be unavailable or non-existent) to generate invisibility. But who ever dared change the font setup in SLiTEX anyway? The exciting work of Mittelbach and Schöpf [Mittelbach & Schöpf 1990] changes the situation, however. It is now very easy to build a SLiTEX which uses a different set of fonts; but if we decide to do our slides in, say, Optima, how do we get invisibility? There are three approaches:

1. If we are using a PostScript printer, there is a simple, and elegant, solution. Where we want invisible text, we simply typeset it in the parent font, but bracket it with a pair of `\special` commands which instruct PostScript to set these letters in white, i.e., invisible.[1] This has the great advantage that only one set of font metrics is needed for TEX, but has the disadvantage that it is dependent on the printing device. Leslie Lamport (*pers. comm.*) has written an appropriate style file for SLiTEX which does the necessary work (this is *not* part of the LATEX distribution at present).
2. We could tinker slightly with TEX to do the same thing, i.e., read a single font metric file, but in a certain mode produce only white space rather than the characters. So far as I am aware, this has not been done. Whether it is possible by arcane TEX macros, I feel incompetent to judge! The helpful reviewer of this paper suggests, as a start:

   ```
   \catcode`a = \active
   \defa{\setbox0=\hbox{a}\hskip\wd0\relax}
   ```

   Chris Torek's 'mctex' distribution goes in this direction by allowing for a mapping file read by the drivers which lets you assign the names of conventional `tfm` files to the invisible names, and add an attribute of invisibility; this forces the driver to look up the width in the `tfm` file, and move right by the required amount.
3. We can take an intermediate route, and generate a set of font metrics which satisfy TEX itself about the size of characters in the font, but actually point to virtual fonts which produce just white space. This is the approach I have adopted. Its advantage is that it is portable, and applicable to any font, but has the disadvantage that TEX still has to load two sets of font metrics, which are identical.

2 Methodology

Even if we have no METAFONT source for a font, we certainly have a TEX `tfm` file. We therefore start with that as our basis, and extract from it the widths of every character. This data can be used to write a virtual font which has an entry for every character, but whose body simply contains a `dvi` instruction to move sideways by the width of the original character.

As the format `tfm` is not very easy to read, let us look at the work that needs to be done in the

[1] This scheme would fail, not if we printed on coloured paper, but if we overlaid text on some other colour, in which case PostScript's imaging model would produce white letters.

human-readable pl form. The format of the vpl file, the corresponding human-readable form of the vf file, is a superset of the pl format [Knuth 1990, p. 18ff.]. The first part of the file, which describes the basic characteristics of the font, can be left more or less unchanged, adding just a VTITLE line; for example:

```
(VTITLE created on Sunday, October 28, 1990
  11:28 pm)
(COMMENT an invisible form of the font)
(FAMILY TIMES-ROMAN)
(CODINGSCHEME ADOBESTANDARDENCODING)
(DESIGNSIZE R 10.0)
(COMMENT DESIGNSIZE IS IN POINTS)
(COMMENT OTHER SIZES ARE MULTIPLES OF DESIGNSIZE)
(FONTDIMEN
   (SLANT R 0.0)
   (SPACE R 0.25)
   (STRETCH R 0.3)
   (SHRINK R 0.1)
   (XHEIGHT R 0.448)
   (QUAD R 1.0)
   )
```

A typical entry for a character in the pl file looks like this:

```
(CHARACTER C a
   (CHARWD R 0.444)
   (CHARHT R 0.458)
   (CHARDP R 0.014)
   )
```

so all we need do is reproduce this, and duplicate the width in a MOVERIGHT command:

```
(CHARACTER C a
   (CHARWD R 0.444)
   (CHARHT R 0.458)
   (CHARDP R 0.014)
   (MAP
    (MOVERIGHT R 0.444))
)
```

There is no need to worry about heights and depths, as the dvi file will contain commands to reposition the current print position at the start of each line, and TEX will have ensured that the baseline separation is correct. Even characters like floating accents, such as the circumflex from Times-Roman, have a negative 'depth' to tell TEX that they occupy only the top area of the box:

```
(CHARACTER O 303
   (CHARWD R 0.333)
   (CHARHT R 0.675)
   (CHARDP R -0.494999)
   )
```

This presents no difficulties because it is TEX which has done all the work establishing where the accent is to go, and has written instructions into the dvi file to position us there. The simple MOVERIGHT should continue, therefore, to work.

3 Implementation

An ideal conversion program would read a tfm file and write a vf file. I have adopted a simpler approach, which is to write a utility in the text-processing language Icon which reads an ASCII pl file and writes a vpl file. My program consists of the following steps, illustrating how easily a pl can be parsed:

1. run tftopl on the base font
2. open the pl file
3. read and concatenate lines until matching sets of (and) are found; process result.
4. for each element that is a CHARACTER, copy out the CHARWD value as a MAP ... MOVERIGHT command at the end of the entry.
5. simply copy out other elements (e.g. FONTDIMEN).
6. call vptovf.

When the resulting vf and tfm files are installed, we are ready for action. If we started with a file called ptmr.tfm, we generate ptmrj.tfm and ptmrj.vf[2]. TEX is told the invisible font is called ptmrj, and reads the metric file. The dvi driver finds ptmrj.vf, obeys the MOVERIGHT instructions, and no further characters are typeset.

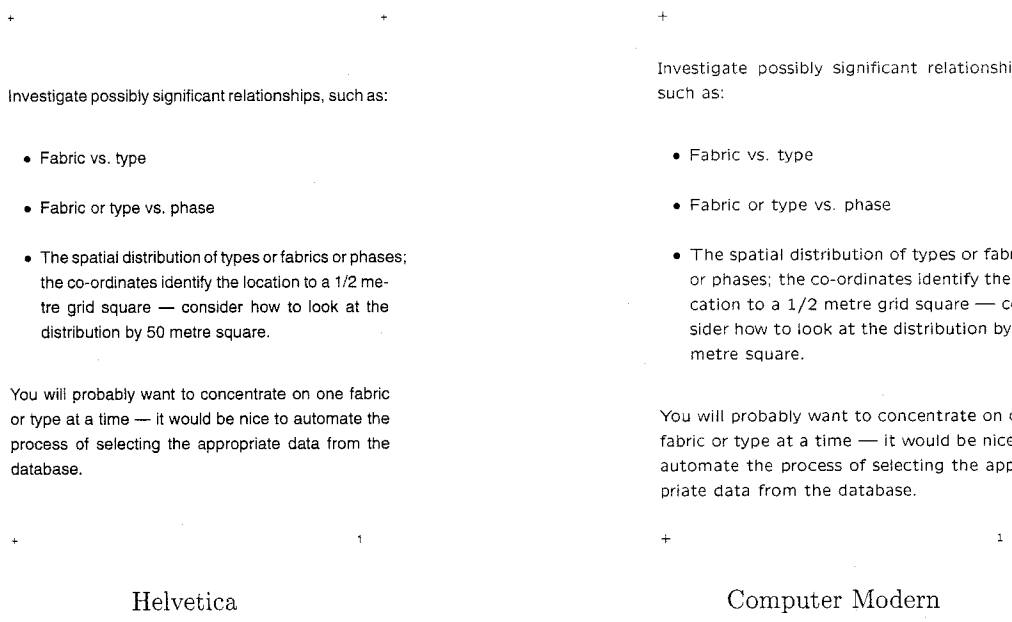
4 Results and acknowledgements

This idea has not been fully tested, and TEX may have tricks up its sleeve to waylay us. But it does work in simple examples (including things like accentuation), and I have some confidence that a portable tftoiv program can be written to make this a general technique. Those who use a PostScript printer will find the 'write-white' approach more attractive (especially as it saves on font space) and extensible, but less portable.

Russell Lang (rjl@au.edu.monash.cc.monu1) has translated my rough Icon program into C, and this latter version is available from him or me for interested parties who care to rewrite the whole thing in WEB for the TEX community, or make it read and write tfm and vf files.

Frank Mittelbach has written an experimental redefinition of the \invisible macro in SLiTEX which takes the ingenious route of simply prepending 'iv' to the name of the current font family, and then calling \selectfont to pull out the correct font, bypassing the hard-wired font names in the

[2] Using Berry's font naming scheme, and adding a 'j' attribute for invisibility



Helvetica 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