Virtual Fonts in a Production Environment

Michael Doob
Department of Mathematics
University of Manitoba
Winnipeg, MB R3T 2N2
Canada
Internet: michael_doob@umanitoba.ca

Craig Platt
Department of Mathematics
University of Manitoba
Winnipeg, MB R3T 2N2
Canada
Internet: c_platt@umanitoba.ca

Abstract

The virtual font facility allows new fonts to be created from existing ones. It is possible to change the properties of a particular character, to rearrange characters within a font, to combine characters from several different fonts, and, perhaps most importantly, to execute sequences of instructions when printing a single character.

This paper will give several applications of virtual fonts that have made the printing of the journals of the Canadian Mathematical Society more efficient and more attractive. Most of these applications arise from the necessity of using a given set of PostScript fonts. There will be some discussion of the reasons why the use of virtual fonts became the best alternative.

There is no assumption of prior knowledge concerning virtual fonts. All necessary concepts will be explained as they arise.

Introduction

Virtual fonts were introduced by Knuth (1990, page 13) as a mechanism for making seamless applications of TEX to different types of printing hardware. There have been several applications of this mechanism since then, e.g., Hosek (1991), but the widespread use anticipated in the original article has not as yet taken place. This is unfortunate since virtual fonts very much enhance the flexibility with which TEX may be applied.

There are several purposes of this paper. We want to examine some problems that arose when using TEX to produce several journals for the Canadian Mathematical Society, and to show why virtual fonts turned out to be the best mechanism for their solution. We also want to gather material about the construction of virtual fonts that heretofore has been scattered in different publications. It is hoped that this will make it easier for others to use virtual fonts, and that the original enthusiasm of Knuth will be justified.

Using virtual fonts: the alphabet soup. Let's think for a moment about what happens when we use `{\it A}` within a TEX file. In the `dvi` file there is a command to change font and then a byte containing the ASCII code for the letter "A", i.e., the number 65. The software used for printing or previewing `dvi` files is generically called a *device driver*; when the device driver comes to this part of the `dvi` file, it will look up the appropriate (normally a `pk`) file, and use the data there to construct the image of the original letter. When a virtual font is used, the number 65 refers to a set of instructions. It may be simply to print the letter "A" as before, but it may also allow letter substitutions from the same font or from different fonts, or allow for a combination of different letters. In other words, several different physical fonts can be combined into one virtual font. Even more, the rules can add lines and move character positions, and can send `\special` commands to the printing device. And so to use this virtual font mechanism we need two

things: (1) a device driver that understands how to use virtual fonts, and (2) a method for creating these fonts.

The program *dvips* understands virtual fonts and is what is used in the (PostScript) environment at the Canadian Mathematical Society. So does the current version of *xdvi*, which can be used for previewing output on the screen of an X-terminal. In addition, *dvidrv* in the emTEX package and *Textures* (version 1.6) are able to interpret virtual fonts properly.

There is an alphabet soup of file names that are used with TEX (see Schrod (1993) for a complete list). Some of these are used in connection with virtual fonts. The ordinary use of TEX involves a `tex` file which contains the source code, the `dvi` file that receives the output of TEX, and another file (in our case a `ps` one) that may be produced in order to view or print the output. These files are in the left column of Figure 1. As TEX runs, it reads the `tfm` files to get information about, among other things, the bounding box (but not the actual shape) and the side bearings of the individual letters, and the kerning and ligature data. The device driver uses the `dvi` file for positioning characters on the page, and (usually) the `pk` files to get the shapes of these characters. To use the virtual font mechanism, it is necessary to have `vf` files; these contain the information to be decoded by the device driver, which can then produce the output in the usual manner. The `vf` file, like the `tfm` and `pk` file, is machine (and not human) readable.

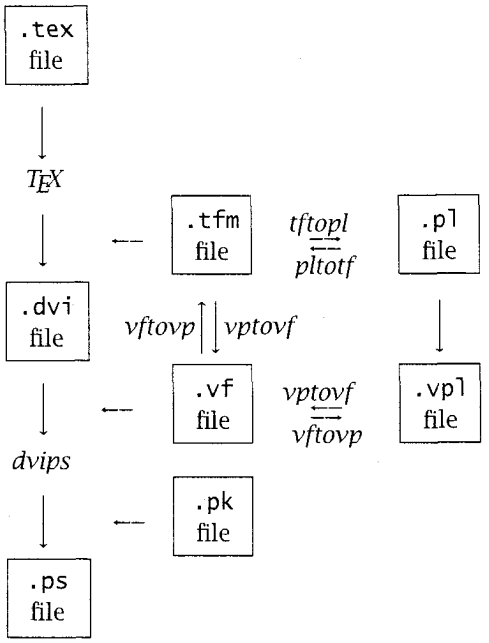

Figure 1: Some alphabet soup

It's possible to adjust the parameters in a `tfm` file via two auxilliary programs. The program *tftopl* takes a `tfm` file as input and produces a `pl` file as output. This is an ASCII file containing a description of the original `tfm` file; the parameters may be changed using a simple editor. Similarly the program *pltotf* will take the `pl` file as input and give the corresponding `tfm` file as output.

There is an extension of this idea to handle `vf` files. The program *vftovp* takes a `vf` and a `tfm` file as input and produces a human readable `vpl` (virtual properties list) file. This file may be edited. Conversely, *vptovf* takes the `vpl` file and produces the `tfm` and `vf` files. And so, as far as virtual fonts are concerned, the name of the game is to edit and adjust the `vpl` file until the desired result is achieved.

Working Examples

An all caps font. We use 12pt roman all caps for titles. At first blush, this should be trivial. After all,

```
\uppercase{the quick brown fox jumps
   over the lazy dogs}
```

will give

THE QUICK BROWN FOX JUMPS OVER THE LAZY DOGS.

But consider the following example:

```
The $ l^1 $ norm of $\xi$ is
   $\sum_{i=1}^\infty \xi_i$}
```

The use of `\uppercase` changes the text from

$$\text{The } l^1 \text{ norm of } \xi \text{ is } \sum_{i=1}^{\infty} \xi_i$$

to

$$\text{THE } L^1 \text{ NORM OF } \xi \text{ IS } \sum_{I=1}^{\infty} \xi_I.$$

This gives us a syntactically correct sentence that will cause great pain to functional analysts. Obviously we don't want to change the case of the mathematical symbols. The solution that then comes to mind is to use `\ifmmode` to check if the text is in math mode. So, for example, we might use something like the following:

```
\def\ucw#1 {\def\next{\ucw}%
   \ifx *#1 \def\next{\relax}
   \else \ifmmode #1
         \else \uppercase{#1}
         \fi
   \fi
   \next
   }
```

We have (rather arbitrarily) set up * as a terminator; we grab a word at a time and check for math mode

(the astute observer may have already noted how the space between words is replaced). If we use this with our last example we get THE l^1 NORM OF ξ IS $\sum_{I=1}^{\infty} \xi_I$ This has fixed the problem, at least as far as the mathematics is concerned. But note that the subscript of the original ξ_i is still being changed to upper case. A moment's thought will reveal the problem. If the entire mathematical expression `$...$` is grabbed at once, the test for math mode will come too late.

So it would seem that we need to grab a character (token) at a time. We could do this with something like the following:

```
\def\ucc#1{\def\next{\ucc}%
   \ifx *#1 \def\next{\relax}
   \else \ifmmode #1{}
         \else \uppercase{#1}%
         \fi
   \fi
   \next
   }
```

This macro will give us

THEl1 NORMOFξ IS$\sum i = 1 \infty \xi i$

All the mathematics is lower case now, but we have obviously caused problems in the way the line is parsed. It seems that our approach is not getting us too far.

So let's rethink the problem. The root cause is the fact that the mechanism for case conversion is the TEX primitive `\uccode`, and this is not defined on a font by font basis (in fact it works even in mathematical text: if you look at the TEX output from `\lowercase{${\cal B}I^\prime M{\cal C}$}`, you'll say "I'm floored!").

It is possible to assign new values to `\uccode`, so we could toggle the values when shifting in and out of math mode with a construction similar to `\everymath`. But the problem at hand is really a font level one; it cries out for a font level solution. One solution would be to use METAFONT and design an all caps font from scratch. This is an arduous job. In contrast, the solution using virtual fonts is almost trivial.

Let's see how to construct an all caps virtual font. According to Figure 1, we need a `vpl` file to edit; where does it come from? We can start with the `tfm` file and use *tftopl* to create a `pl` file. Since the virtual font description is a superset of the `tfm` font description, we can use this file as a starting point. So we can use the command `tftopl cmr12.tfm cmr12ac.vpl` to get started on a 12 point `cmr` all caps font.

The new file can be edited; the structure is strictly defined and not too hard to follow. The first few lines will contain some preliminary information about the font. This is followed by a short list starting with `(FONTDIMEN` (these are the same dimensions described in *The TEXbook* by Knuth (1990, page 433)). Then there is a long list under `(LIGTABLE` and finally a list of the 128 different character entries, each of which starts with `(CHARACTER`.

Within each list several types of objects are described: `(LABEL`, `(LIG`, and `(KRN`, for example, start the description of a label, a ligature, and a kern. Similarly `(CHARACTER`, `(CHARWID`, `(CHARHT`, `(CHARDP`, and `(CHARIC` start the description of a character, and its width, height, depth and italic correction. The object is usually followed by a parameter: `O 40` is the octal number 40, `D 32` is the decimal number 32, `C a` is the (ASCII code of the) character "a", and `R.9791565` is a real number as a multiple of the design size (which is after `(DESIGNSIZE` as one of the first entries of the `vpl` file). In our case the design size is 12 points, so the real number has the value of 11.75 points. Finally, there will be matching `)`s to finish the description.

So now we can interpret the text of the `vpl` file:

```
(LABEL C f)
(LIG C i O 14)
(LIG C f O 13)
(LIG C l O 15)
(KRN O 47 R 0.069734)
(KRN O 77 R 0.069734)
(KRN O 41 R 0.069734)
(KRN O 51 R 0.069734)
(KRN O 135 R 0.069734)
(STOP)
```

means that the we are describing the character "f", that there is a ligature with the character "i" and the pair is replaced by the character with ASCII code octal 14; there are two more similar ligatures; next we see that when "f" is followed by the character whose ASCII code is octal 47 (the "'" character), there is a kern of .069734 design units (a positive kern means that the letters are actually being spread apart), etc.

Similarly,

```
(CHARACTER C f
   (CHARWD R 0.299187)
   (CHARHT R 0.694444)
   (CHARIC R 0.069734)
   (COMMENT
      (LIG C i O 14)
      (LIG C f O 13)
      (LIG C l O 15)
```

```
    (KRN O 47 R 0.069734)
    (KRN O 77 R 0.069734)
    (KRN O 41 R 0.069734)
    (KRN O 51 R 0.069734)
    (KRN O 135 R 0.069734)
    )
)
```

means that the character "f" has width, height, and italic correction as given. Since (CHARDP doesn't appear, its value will be zero. Notice that it is also possible to have comments. In this case, the ligature and kerning information is repeated as a convenience.

Now let's add some new instructions to the vpl file to make our all caps font. First we add

```
(MAPFONT D 0
   (FONTNAME cmr12)
   (FONTCHECKSUM  O 13052650413)
   (FONTAT R 1.0)
   (FONTDSIZE R 12.0)
   )
```

before the (LIGTABLE. We are defining a font that can be used later: it means that font 0 refers to the font cmr12 which has the given checksum (note that this value is part of the output from *tftopl*; we need only copy it into place). The design size of the font is 12 points with a scaling factor of 1.

To replace the "f" entry by the "F" entry we replace the description of the character given above with

```
(CHARACTER C f
   (MAP
      (SELECTFONT D 0)
      (SETCHAR C F)
      )
   )
```

and that's it. Of course since we have no given values for CHARWID, CHARHT, CHARDP, and CHARIC, they all have the default value of 0. Unless we want all the characters to print one atop the other, this is undesirable. The correct values for "F"are given in the (CHARACTER C F listings, so we can just copy them into place. Now we have

```
(CHARACTER C f
   (MAP
      (SELECTFONT D 0)
      (SETCHAR C F)
      )
   (CHARWD R 0.638999)
   (CHARHT R 0.683333)
   )
```

If we do this for the other letters, we have then made the desired replacements. This takes only a few minutes with a smart editor.

There are a few other things to do: the ligature and kerning information still corresponds to the original font. In our case there are only three ligatures that need to be deleted: ff, fi, and fl. So we take those lines out of the (LIGTABLE listing. We also have the kerning for the old letters; we replace it with the corresponding upper case entries; as it happens, there are no kerns for "F", so all of the lines of the original entry

```
(LABEL C f)
(LIG C i O 14)
(LIG C f O 13)
(LIG C l O 15)
(KRN O 47 R 0.069734)
(KRN O 77 R 0.069734)
(KRN O 41 R 0.069734)
(KRN O 51 R 0.069734)
(KRN O 135 R 0.069734)
(STOP)
```

are deleted. Sometimes large all caps are typeset without kerning (yuk!). If desired this could be part of the virtual font parameters. Track kerning (the addition of a small uniform amount of space between letters) could also be done by changing CHARWD appropriately. Now we're done with the editing of the vpl file.

We now run

```
vptovf cmr12ac.vpl cmr12ac.vf cmr12ac.tfm
```

and the vf and tfm files are ready to go (of course these files must be in directories where TEX and the device drivers will look for them).

The TEX fragment

```
\font\ac=cmr12ac
\ac
The $ l^1 $ norm of $\xi$ is
   $\sum_{i=1}^\infty \xi_i$.
```

will now work properly.

The construction of the font is really quite easy once the proper pieces are assembled. There is a bonus for our production work. We must process files from authors that are in plain TEX, LATEX, and $\mathcal{A}_{\mathcal{M}}\mathcal{S}$-TEX, among other variants. The virtual font gives a single solution that works with all macro packages. This is an important benefit.

A small caps font. The problem with designing a small caps font within TEX has been addressed by Hendrickson (1990). Of course if you have cmcsc10.mf you can generate a Computer Modern small caps font using METAFONT. But for other

sizes or families the virtual font mechanism is again almost trivial. To construct, for example, a small caps font using the PostScript Times-Roman family, the procedure is hardly different from the first example. Suppose that `rptmr.tfm` is used by TEX to typeset Times-Roman, and the small caps should be 25% smaller than the upper case caps. It is only necessary to define two fonts in the `vpl` file:

```
(MAPFONT D 0
   (FONTNAME rptmr)
   (FONTCHECKSUM  0 24360352060)
   (FONTAT R 1.0)
   (FONTDSIZE R 10.0)
   )
(MAPFONT D 1
   (FONTNAME rptmr)
   (FONTCHECKSUM  0 24360352060)
   (FONTAT R 0.75)
   (FONTDSIZE R 10.0)
   )
```

The editing of the `vpl` file proceeds almost exactly as before starting with

```
\(CHARACTER C F
   (MAP
      (SELECTFONT D 0)
      (SETCHAR C F)
      )
   )
```

and

```
\(CHARACTER C f
   (MAP
      (SELECTFONT D 1)
      (SETCHAR C F)
      )
   )
```

There is a question as to which size accents to use: they can come from the larger or smaller font. You have to pick one (we use the smaller size).

One font, two uses. When our journals are ready to print, we send a PostScript file to The University of Toronto Press for high resolution printing, binding and mailing. Since this is over 1500 kilometres from our office, some care must be used to make sure that all the files are correct. Rerunning pages on a high resolution printer is expensive. In addition, we cannot reload fonts to replace ones that are resident on the printer in Toronto.

A consequence of this arrangement is that we must use Times-Italic for both italic text and mathematical symbols. This creates a number of problems with intersymbol spacing. For example, the letter "f" as text would normally extend out of its `tfm` bounding box both on the left and on the right. Normally the lower left tail will hang under the preceding letter. Similarly, the "j" and "p" also have tails that hang out of the bounding box. As a consequence, in expressions like `$f^p$` and `$\bigl(f$` the symbols will almost bump into each other. The situation can be greatly improved by adjusting both the position within and the width of the bounding box. We have already seen that we can use CHARWD to change the width of the bounding box. Similarly there are commands MOVEUP, MOVEDOWN, MOVERIGHT and MOVELEFT to adjust the position within the bounding box. Using our example from `cmr12` (with a design size of 12 points), we could move the letter "f" 1.2 points to the right using

```
(CHARACTER C f
   (CHARWD R 0.299187)
   (CHARHT R 0.694444)
   (CHARIC R 0.069734)
   (MAP
      (SELECTFONT D 0)
      (MOVERIGHT R 0.1)
      (SETCHAR C f)
      )
   )
```

In effect, the virtual font allows us to make microadjustments to the fonts in the printer in Toronto. In practice this has been extremely useful.

Character rearrangement. Several special alphabets are common in mathematical expressions. It is normal to use some type of script or calligraphic font, something like Fraktur or BlackLetter, and "blackboard bold" characters. Coding is simplified if, like the `\cal` control word in plain TEX, control words `\Bbd` or `\Frak` can be defined to use letters that appear in their natural ASCII position.

In our case we are given these special characters as part of a special (proprietary) symbol font from the University of Toronto Press. There are upper case "blackboard bold" letters and both upper and lower case Fraktur characters. These letters are scattered around and do not appear in their natural order, much less in their ASCII position.

It's easy to see how to solve this problem. Just define two new virtual fonts, one for each typeface. The construction is essentially the same as for the all caps font.

There is an extra advantage to this approach The "blackboard bold" characters are usually only defined for uppercase letters; sometimes the letter "k" and the number "1" are also included. Fraktur is only used for upper and lower case letters. If one tries to use an undefined character, say `{\Bbd 2}`,

there will be no tfm entry, but TEX will process the file anyway. No character will appear in the text. On the other hand, an entry in the vpl file like

```
(CHARACTER C 2
   (CHARWD R 0.7)
   (CHARHT R 0.8)
   (MAP
      (SETRULE R 0.8 R 0.7)
      )
   )
```

will cause a big slug to be printed; it will be evident that something is wrong.

It's possible to remap a PostScript font so that it will match each character in the cmr family. This separates TEX problems from external font problems and can simplify some macro implementations.

Lines above, through and below. TEX provides two ways of putting lines over characters. The \bar control word will put a line of a particular size over a character, while the \overline control word will put a (generally longer) line as big as the bounding box. Now it happens that in the PostScript Times-Italic font there is an accent that can be used with the \bar command. Unfortunately it is very narrow and while it is acceptable for use over the dotless i \imath, it looks terrible over, say, the letter "M". Also, \overline is too big because of the large bounding box for upper case Times-Italic letters. The solution is simple: just replace the given bar by a bigger one! This is a special case of adding horizontal and vertical rules to a character.

Let's go back to our example from the cmr12 font. Suppose we want to put a line above the letter "I" in our all caps font. We only need adjust the vpl file:

```
(CHARACTER C I
   (MAP
      (SELECTFONT D 0)
      (PUSH)
      (SETCHAR C L)
      (POP)
      (MOVEUP  R 0.683333)
      (MOVEUP R 0.1)
      (SETRULE R 0.03 R 0.6118)
      )
   (CHARWD R 0.6118)
   (CHARHT R 0.683333)
   )
```

One might visualize this as a pen moving to different positions. Several steps have been followed: the character "L" was set as before, the position was popped back to original starting point, the position was moved up the height of the letter and then moved up a little more, and finally a rule was set with height R 0.03 (0.36 points) and width R 0.6118 (the width of the letter). With appropriate adjustments to the dimensions, all of the letters which look badly with \overline and \bar can be replaced by a better looking substitute. It's even possible to have a font in which every italic letter has its own overline form.

It is also trivial to make a "strike through" font where each letter has a horizontal line through it. These are sometimes used in contract revisions to indicate deleted material. A little care with positive kerns will be needed if the strikethrough lines are to meet for consecutive letters.

The same principle allows the construction of an underlined font; it's even easy to include a gap for the descenders.

Some special characters. The PostScript Times-Italic font has a dotless i but no dotless j. Even this problem is easy to solve using a virtual font. Using the cmr12 example once more, consider the following addition to the vpl file:

```
(CHARACTER C j
   (MAP
      (SELECTFONT D 0)
      (PUSH)
      (SETCHAR C j)
      (POP)
      (SPECIAL " 1 setgray
         1.5 7.5 1.5 0 360 arc fill )
      )
   (CHARWD R 0.503005)
   (CHARHT R 0.683333)
   )
```

The (SPECIAL command works exactly like \special in the TEX file. Whatever follows is passed on to the device driver for processing. In this case (for *dvips*) it is a PostScript command that paints a little filled white circle right over the dot of the letter.

There is, however, a problem with this method. If an accent is put over the dotless j (and why else would the dotless j be used?), the accent is printed first and the letter next; if the accent is unfortunate enough to hit the dot over the j, then it will be erased along with the dot. One solution is to print the dotless j first using an \rlap, and then essentially print the accent over a phantom of the same character. A better solution has been provided by Sebastian Rahtz (who also discovered the original problem). It uses PostScript to clip the j at the height of a dotless i:

```
(CHARACTER O 32 (comment dotlessj)
   (CHARWD R 278.00)
   (CHARHT R 458.00)
   (CHARDP R 217.00)
   (CHARIC R 0.00)
   (MAP (SELECTFONT D 0)
      (SPECIAL ps: gsave newpath 0 0
      moveto (\31) true charpath
      flattenpath pathbbox
      /IHeight exch def pop pop pop
      grestore gsave
      newpath 0 0 moveto (\152)
      true charpath flattenpath
      pathbbox pop exch /JDepth
      exch def
      /JRight exch def /JLeft exch def
      grestore gsave newpath)
      (PUSH)
      (MOVEDOWN R 217.00)
      (SPECIAL ps: JLeft JDepth rmoveto
      JLeft neg JRight add 0 rlineto
      0 JDepth neg IHeight add rlineto
      JLeft neg JRight add neg 0
      rlineto
      0 JDepth neg IHeight add neg
      rlineto closepath clip)
      (POP)
      (SPECIAL ps: (\152) show
      grestore)
      )
   )
```

Conclusions

A number of applications of virtual fonts have been presented. The complete list of commands that may be used in a vpl file is contained in the WEAVE output of VPtoVF.web. A copy of this output is available on the internet from Walsh (1993). In fact, almost every facility was used here; they turned out to be just what was needed in an actual production environment. No doubt this reflects positively on the choice of tools by Donald Knuth and David Fuchs.

Perhaps the most important benefit has been a single solution that works over all macro packages. Virtual fonts have proven themselves valuable; with wider awareness of their uses, more applications will undoubtedly become available.

Bibliography

Hendrickson, Amy. "Getting TEXnical: Insights into TEX macro writing techniques." *TUGboat*, **11**(3), pages 359–370, 1990.

Hosek, Don. "Siamese TEX: Joining dvi Files at the Hip and Other Novel Applications of VF files." *TUGboat*, **12**(4), pages 549–553, 1991.

Knuth, Donald E. *The TEXbook* (nineteenth printing). Reading, Mass.: Addison-Wesley, 1990.

Knuth , Donald. "Virtual fonts: More Fun for Grand Wizards." *TUGboat*, **11**(1), pages 13–23, 1991.

Schrod, Joachim. "The Components of TEX." available via anonymous ftp on the CTAN servers in the documentation directory.

Walsh, Norm. "The VFtoVP Processor." (output of WEAVE applied to VFtpVP.web) /pub/norm/docs/web/vftovp.tex on the server ibis.cs.umass.edu, 1993.