Substituting outline fonts for bitmap fonts in archived PDF files †

STEVE G. PROBETS AND DAVID F. BRAILSFORD
School of Computer Science
University of Nottingham
Jubilee Campus
Nottingham, NG8 1BB
UK

22 May 2001

Abstract

As collections of archived digital documents continue to grow the maintenance of an archive, and the quality of reproduction from the archived format, become important long-term considerations. In particular, Adobe's PDF is now an important 'final form' standard for archiving and distributing electronic versions of technical documents. It is important that all embedded images in the PDF, and any fonts used for text rendering, should at the very minimum be easily readable on screen. Unfortunately, because PDF is based on PostScript technology, it allows the embedding of bitmap fonts in Adobe Type 3 format as well as higher-quality outline fonts in TrueType or Adobe Type 1 formats. Bitmap fonts do not generally perform well when they are scaled and rendered on low-resolution devices such as workstation screens.

The work described here investigates how a plug-in to Adobe Acrobat enables bitmap fonts to be substituted by corresponding outline fonts using a checksum matching technique against a canonical set of bitmap fonts, as originally distributed. The target documents for our initial investigations are those PDF files produced by (LA)TEXsystems when set up in a default (bitmap font) configuration. For all bitmap fonts where recognition exceeds a certain confidence threshold replacement fonts in Adobe Type 1 (outline) format can be substituted with consequent improvements in file size, screen display quality and rendering speed. The accuracy of font recognition is discussed together with the prospects of extending these methods to bitmap-font PDF files from sources other than (LA)TEX.

†This is the final draft of a paper which was accepted for publication in *Software — Practice and Experience* and which appeared in issue 33(9), 2003 pp. 885 – 899

1 INTRODUCTION

Over the past 10 years Adobe's PDF has become extremely popular as an archiving format, largely because of its PostScript-based architecture which allows complex material to be rendered at very high quality on page and on screen. A second important consideration is that PDF can give an accurate rendering of exactly what was published in hard-copy format with all layout, including page breaks, line breaks and so on, kept intact. PDF files are viewed in the Acrobat viewer software and can either have all the required fonts embedded in the file or can rely on finding the fonts in the system environment where the Acrobat viewer is running. The option of embedding all fonts, to maximise portability of the PDF, is becoming increasingly popular.

When a PDF file is the only archived form of a document it is important that it be of high quality. Unfortunately there are many examples of PDF documents available on the Web where the quality of fonts and diagrams is so bad that the screen display is almost indecipherable.

If a corpus of PDF documents is to be properly maintained then, in an ideal world, any upgrading of the PDF should be achieved by completely reprocessing the enhanced and amended source material. To this end, the publisher should also archive all the source files together with all the processing software needed to transform the source to the PDF. This is a very severe requirement and one that few organisations would be prepared to undertake. Note that, in extreme cases, it would be necessary to retain the precise release of Acrobat Distiller (to create the PDF) and all the system fonts that were used. More importantly, it would be vital to keep the exactly correct release of the text processing software, since important algorithms such as hyphenation and diagram placement can vary greatly from release to release.

In practice many publishers of scientific material now archive PDF files together with some XML metadata about each article. If the publisher's workflow starts with full-text SGML or XML then this may well be archived also, but what will almost always be missing are the source files for the 'typesetting middleware' that lies between the abstract XML document and the final PDF form. Indeed, many of the preservation schemes for maintaining digital resources, especially ones based on the Open Archival Information System (OAIS) model [1], are only just beginning to address the problems of maintaining the necessary hardware and software resources to enable accurate replication of archived material over a period of time.

For all these reasons maintenance and upgrade of an electronic document archive will often have to be on the final-form PDF only. If there are poor quality diagrams or screen shots within a PDF then, quite apart from the technical problems of inserting better quality material, the diagrams will generally be so specific to each article that the author's positive collaboration would be needed to secure new versions. On the other hand, if the typesetting system used to create the PDF files has made use of bitmap fonts then, in principle, better

quality fonts could be substituted provided the fonts in use can be unequivocally recognised.

A random sample of bitmap-font PDF documents, drawn from public sources on the Web, showed that a high proportion of them arose from use of the popular TEX or LATEX software by authors who were either unaware of the fact that better quality outline fonts were available, or who lacked the skill to configure their systems to use them. The large amount of material available, and the fact that suitable replacement fonts could be identified, caused us to try out the feasibility of PDF font replacement on this (LA)TEX-originated material.

2 TEXand LATEX

The TEX typesetting software [2], and the later 'structured' development of it called LATEX [3] occupy an important niche in the creation of electronic documents with a high technical or mathematical content. These two software systems have a loyal following within the academic community and in the STM (Scientific, Technical and Medical) segment of journal and book publishing. In what follows we trace the processing of TEX and LATEX to PostScript via a common program called `dvips` and thence to Adobe's PDF format. When discussing possible source texts for `dvips` we use the logo (LA)TEX to mean 'TEX or LATEX '.

Knuth's original TEX not only made technical typesetting software freely available but it also provided a free set of bitmap fonts at various resolutions. These fonts implemented a typeface design called Computer Modern (CM), loosely based on Victorian 'Schoolbook' designs. The Computer Modern fonts, in bitmap or outline forms, are an important part of the look and feel of most (LA)TEX-created material. The CM fonts were designed with a program called METAFONT [4] — another remarkable item of Knuth-created software — which uses an outline representation of characters internally but whose output will generally be a *bitmap* font for a particular resolution and a particular pointsize. The standard output format from METAFONT is called `gf` (generic font) but device drivers at the far end of the (LA)TEX chain (and `dvips` in particular) now generally use a compressed bitmap format called `pk`, which is created from the `gf` format via a utility called `gftopk`.

The problems with bitmap fonts went largely unnoticed in the days when high-quality output was solely to devices such as laser-printers. Provided these printers had the bitmap fonts on a local disk, or could download them over a fast printer link, all the user saw was the perfectly acceptable output that resulted from using hand-tuned bitmaps matched to the printer resolution. But difficulties began to emerge with a new generation of screen previewers (e.g GhostScript and Acrobat) that were capable of using the CM printer fonts *directly*, rather than relying on low-resolution screen-font clones. The first of these programs, GhostScript[5], is public-domain software for previewing PostScript output on a variety of hardware platforms, while Adobe's Acrobat viewers interpret Portable Document Format (PDF) files where PDF is itself based on Level 2 PostScript. What GhostScript and Acrobat have in common is that they can display typeset pages very realistically by directly

rendering, on screen, the selfsame bitmap or outline fonts (in either of the Adobe Type 1 or Type 3 formats) that will be used for output to a laser-printer. They are both capable of magnifying a page and thereby allowing the user to zoom in on fine detail. Under these circumstances bitmap fonts dissolve into 'staircases' and 'jaggies'; the advantages of the outline format in terms of rendering speed and elegant scalability become all too apparent.

3 Type 1 vs. Type 3 Adobe PostScript fonts

PostScript has long been the most popular choice of graphic output format for (LA)TEX documents and its fonts are collections of procedures which describe character shapes. The procedures are grouped together in a font dictionary data structure which can be accessed by the PostScript interpreter. The simpler of the two font formats is Type 3 which simply requires the font designer to provide a font dictionary entry called BuildChar which the interpreter can call every time it needs to build a character. The PostScript 'Blue Book' [6] gives further details of Type 3 fonts which can be created in outline or bitmap formats. Although the former use a vectorised description of characters they have no built-in 'hinting' mechanism to intelligently control the pixel dropout which occurs, due to rounding, when characters are rendered on low-resolution devices. On the other hand, Type 3 bitmap fonts make use of PostScript's image bitmap format, and the imagemask operator, to build characters based on bitmaps. Type 3 fonts can be based on a character cell of arbitrary size and the units into which the cell is subdivided can be chosen by the font designer.

By contrast, Type 1 font programs have an implicit BuildChar procedure; fonts of this sort must be outlines rather than bitmaps and they must be based on a 1 point character cell employing units of 1/1000 point. The subset of PostScript that is used by the Type 1 buildchar is carefully optimised, within the interpreter, for good performance. Type 1 fonts are also capable of carrying hinting information. In the early days of PostScript, Type 1 format was used solely by professional type designers; the details of the format were confidential and the character outlines were encrypted. But since 1990 the format specification has been publicly available [7].

When attempting to substitute Type 1 fonts for Type 3 fonts, within a PDF file, it is important not only that the characteristics of the fonts should be similar (e.g. stem weights, serifs, x-height) but also that the character widths of the Type 1 substitutes be as near identical as possible to those in the original Type 3. If this latter condition does not hold then the substituted text could show characters partly over-printing one another and previously justified text now appearing to be typeset 'ragged right'. Fortunately, in the case of fonts used in (LA)TEX systems, outline versions of the CM fonts are available from three sources. A set is available from Blue Sky Research (initially distributed via Y&Y Inc. [8]) These are commercially-produced fonts of very good quality and considerable care has been taken with the hinting — a very desirable state of affairs given the thin stems and

other delicate features that are a characteristic of the CM designs. About 10 years ago a consortium of publishers paid a fee to Blue Sky Research so that these fonts could be made freely available to all. Increasingly, therefore, they are included, as a matter of course, with modern TEX packages such as teTeX. A second set, which has recently been put in the public domain, was designed by Basil K. Malyshev (BaKoMa). They can be found on the Comprehensive TEX Archive Network (CTAN)[9]. BaKoMa fonts were created by automatic conversion of METAFONT outlines. The font hinting was also generated automatically (in contrast to the Blue Sky set where the hinting was done by hand) but the BaKoMa fonts are, nevertheless, very serviceable. The most comprehensive CM font set in the Type 1 format is now the Super-CM font set of V. Volvovich released in 2001.

The crux of this paper, given the popularity of Adobe's PDF as an archiving format, is how one 'rescues' (LA)TEX-produced PDF files, with embedded bitmap fonts, and replaces them with an equivalent PDF, using outline fonts, by backtracking to various stages of the processing cycle. The next sections review how outline fonts can be inserted into PostScript output either by `dvips` itself or by post-processing the PostScript that `dvips` produces. We then go on to describe our plug-in for Adobe Acrobat which processes (LA)TEX-produced PDF files to replace bitmap CM fonts with outlines. This is often the only way to proceed in circumstances where a PDF file is available, but the (LA)TEX source, the `dvi` code and the PostScript have not been archived.

4 Device independence and the `dvips` program

TEX was designed from the outset for accurate positioning of its character boxes, on any output device, to an accuracy of around a millionth of an inch. For this reason the files of intermediate code output by classic releases of (LA)TEXare essentially *device independent* [10] and this in turn explains the `dvi` file extension that they use. As ever, the rationale for using an intermediate format such as `dvi`, is that it eases the problem of adding a new output device, or driving new visualisation software. All that is needed is to write a driver to convert the `dvi` code to the desired new format.

The `dvi`-to-PostScript translator most commonly used is `dvips` developed by Tomas Rokicki [11]. The mappings between font names used inside (LA)TEX and the external file names that contain the fonts is controlled by the file `psfonts.map`. By using this mapping file, programs such as `dvips` can locate fonts, for the CM family, that are either PostScript outlines or `pk` bitmaps created from METAFONT. In the case that the target files are bitmaps in the `pk` format, `dvips` can convert them, on the fly, into PostScript Type 3 bitmap format. On the other hand if the files pointed to by `psfonts.map` have either a `.pfa` or `.pfb` extension then these are taken to indicate PostScript fonts that are already in Type 1 (outline) format.

The character positions within a (LA)TEX font are allocated according to a scheme which

started with Knuth's original 7-bit encoding and which has now evolved into a standard known as 'T2A' within the (LA)TEXcommunity (more detail on encodings and many other (LA)TEX-related font matters can be found in [12]). These encodings are designed to utilise to the maximum all the potentially available character positions in a single-byte positional encoding. However, these mappings do not correspond to any of the standard font encodings (e.g. Windows, Adobe Standard, ISO Latin1 etc.) found in system fonts on Windows or UNIX machines. For this reason we cannot generally rely on picking up correct (LA)TEX fonts from the operating system environment, when printing takes place — for safety's sake they are usually *embedded* into the PostScript output stream, by `dvips`, to ensure that the correct output appears.

When locating and embedding a font `dvips` performs some transformations which we need to understand if any post-processing is to take place to substitute bitmap fonts with outlines. If a font, at a particular pointsize, is not available via `psfonts.map` then `dvips` will attempt to activate METAFONT, followed by `gftopk`, to create a new bitmap font specifically tailored for the desired point size and resolution. If the call to METAFONT should fail for any reason then a combination of (apparently) bizarre resolution options for CM bitmap fonts, coupled with various tricks performed by `dvips`, is often sufficient to keep output looking reasonable. These matters are explained further in section 6 but one example suffices for the moment: if a document is being created in 10 point type at 300 dpi and a specific 12 point version of that same font is not available, then a scaling change from 10 point to 12 point can be achieved, very neatly, by changing to an equivalent 10 point font created at 360 dpi (because $360 = 300 \times 12/10$). If a requested pointsize change falls outside the range of what can be accommodated, via the other pointsizes and resolutions already available in the bitmap fonts, `dvips` will then actually generate a PostScript scalefont command to scale an existing bitmap font. Given that such a direct scaling of bitmap fonts leads to inelegant results such a strategy has to be a last resort and, again, is used only if METAFONT is unavailable.

In contrast to the complexities of scaling bitmap fonts, things are much simpler if outline fonts are in use. Here, the scaling to any required pointsize is achieved by embedding a copy of the Type 1 font into the output PostScript and, once again, generating the appropriate call of the PostScript scalefont operator. Since Type 1 outlines have hints, which control the fine details of rendering the font, an elegant scaling can be achieved over a large range of point sizes.

5 Reprocessing legacy material

Figure 1 shows the steps by which (LA)TEX source can be transformed into PDF. A fairly recent development, shown in the lower half of the figure, has been the release of the `pdftex` software which outputs PDF directly and uses Type 1 outline fonts. Although this route is

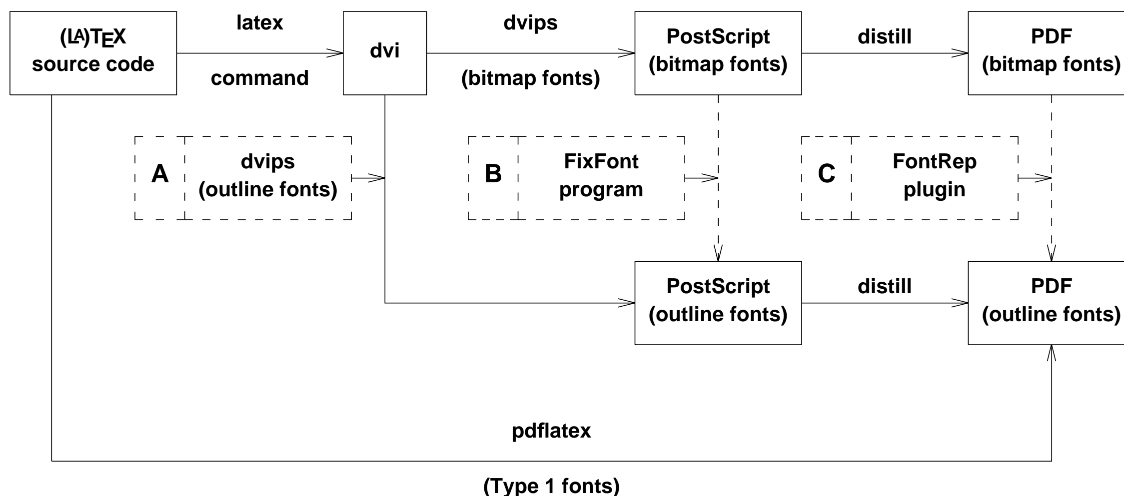

Figure 1: Processing (LA)TEX to PDF

gaining in popularity, and has the advantage that its default font set is of high quality, it requires that all inserted diagrams be prepared as PDF files with an origin of coordinates at (0,0); it therefore lacks the flexibility of the traditional `dvips` route (see upper part of Figure 1) with its ability to incorporate arbitrary encapsulated PostScript. An informal analysis of (LA)TEX-produced PDF files available on line, and those submitted to conferences, shows that the majority of (LA)TEX users still process their output via `dvips` and PostScript. This, in turn, means that a large amount of PostScript and PDF is created with embedded CM bitmap fonts, largely because many users are unaware that a simple `-Pcmz` command line flag to `dvips` would cause outline fonts to be embedded instead.

Figure 1 shows how the PostScript output from `dvips` is transformed via a program such as Adobe Distiller or `ps2pdf`, into a PDF file. The solid lines joining the boxes indicate the conventional routes to PDF, using either bitmap or outline fonts. The dashed lines show the points at which intervention can take place in order to re-process archived files with embedded bitmap fonts into better-performing and generally smaller files, using outline fonts. In the figure the points marked A, B and C show where outline fonts can be inserted, or substituted, into the processing stream.

For newly-created material there is no doubt that the best solution is to have outline fonts available at the earliest point in the processing cycle i.e. at point A, when `dvips`

is creating the PostScript. The needed fonts are obtained as either `.pfa` or `.pfb` files via `psfonts.map`. As we have seen, the necessary fonts will then be embedded in the PostScript, and will be retained via the Distiller program in the final PDF.

On the other hand, for archived (LA)TEX documents, any or all of the (LA)TEX source code, `dvi` file, PostScript or PDF may be available and the fonts in the latter two of these formats could well be CM bitmaps. If the original (LA)TEX source is available it seems sensible, at first sight, to re-process the material from that source and to introduce outline fonts at point A. Indeed, it *is* the best way to proceed provided one accepts that the re-processing may not be entirely straightforward. If the source code is several years old, and if it is important to recreate the exact page layout and line breaks of material currently archived as PDF (say), then one has to recreate as exactly as possible the entire original processing environment i.e. there is a need to archive the exact release of (LA)TEX and its styles, the exact version of `dvips` and the exact fonts that were used. If this is not done then old source code has to be processed with more recent releases of the processing software and this leads, all too easily, to frustrating problems where the processing software crashes because the source text is using 'legacy' features no longer supported; or a new hyphenation-and-justification routine causes line breaks and page breaks to change in the output; or mysterious gaps appear due to exotic characters being chosen from fonts that are no longer available. Putting these problems right can take an astonishing amount of effort.

For all these reasons, and in circumstances where the (LA)TEX source text is not available, the next best alternative — point B in Figure 1 — can be considered, provided that the `dvips`-produced PostScript for the document has been safely archived. The next section describes an existing program that analyses PostScript files of exactly this sort. Although our ultimate goal is to enable font substitution in PDF files the next two sections show that replacing bitmap fonts with outline fonts, in any PostScript-based file format, is far from straightforward; scaling factors need to be calculated for the replacement font which depend on the original resolution of the bitmap fonts and on the innately different character cell sizes of Type 1 and Type 3 character glyphs.

6 EMERGE's FixFont

The FixFont package, originally released by the Emerge corporation [13], consists of a UNIX shell script and a C program. It attempts to modify `dvips`-produced PostScript by substituting Type 1 outline fonts for any embedded CM Type 3 bitmap fonts. It can be used to greatly enhance the quality of (LA)TEX-generated legacy PostScript that contains Type 3 bitmap fonts. Figure 2 is a small example, using the same Computer Modern Roman fonts in bitmap and outline formats, which shows very clearly the superior quality of an outline font.

Type 3 Bitmap
Type 1 Outline

Figure 2: Comparison of bitmap and outline Computer Modern fonts

FixFont achieves a font substitution by first analysing the PostScript to locate and recognise the Type 3 CM fonts. These are then substituted with Type 1 fonts before rewriting a new PostScript file. Because of the need for careful analysis of the original PostScript, FixFont is designed to work only with PostScript produced by Tomas Rokicki's `dvips` program, in its configuration as constituted around 1995 (release 5.495).

In addition to the bitmap font-scaling strategies attempted by `dvips`, its generated PostScript has another major drawback when attempting font substitution: the release 5.495 `dvips` software does not name the Type 3 fonts it embeds with their usual TeX name but instead it generates its own name for the fonts such as `Fa`, `Fb` etc. When the font is embedded the bits making up each character within the font are included as a hexadecimal string in PostScript `image` format. Because of this font name problem, it is necessary to try to recognise what the font actually is, before substitution can occur. This is done by analysing the actual bitmaps of the characters within the font. Once it has been determined that `Fa` is, for example, Computer Modern Roman at 10 points (`CMR10`) and that `Fb` is Computer Modern Math Italic at 10 points (`cmmi10`) then sustitution can be performed safely.

However life isn't quite that simple. Most (LA)TeX installations contain different versions of each bitmap font for various resolutions. Standard resolutions in early (LA)TeX releases were 300, 329, 360, 432, 518, 622, 746, 896, 1075, 1290 and 1548 dpi. More recently these have been supplemented with a new set at 600, 657, 720, 864, 1037, 1244, 1493, 1792, 2150, 2580 and 3096 dpi. So not only does the Computer Modern typeface have a different bitmap font for many different point sizes (6, 8, 9, 10, 12, etc.), it also has a different `pk` font for each different resolution at every point size. This can lead to (LA)TeX installations having hundreds of these fonts, with names like `CMR10.300`, `CMR10.329`, `CMR10.360`, `CMR12.300` etc. In creating the PostScript, `dvips` might pick any one of these fonts, as described in section 5, to minimise the scaling required.

6.1 The FixFont software

FixFont was developed in 1996 or thereabouts and although released via Emerge, there is a distinct impression that it has its roots in work started at Adobe Systems Inc. somewhat earlier. FixFont compares the CM bitmap Type 3 fonts in a PostScript file with the characteristics of the standard METAFONT CM bitmap fonts contained in a database. It works in two stages. The first stage is to generate a database of checksums for all the characters in all possible Computer Modern fonts, once they have been converted to Adobe Type 3 format. This stage only needs to be performed once. Once the software has knowledge of these checksums, the second stage of the process can occur. In this stage the characteristics of bitmap fonts within legacy PostScript files can be compared with the database, in an attempt to recognise the fonts included in the file. If recognition is successful then font replacement can occur.

The second stage takes the PostScript file where font substitution is required, and analyses all the bitmap Type 3 fonts within it. The same algorithm that was used to generate the database is used to generate checksums for all the characters in all the embedded fonts. For example the checksums for fonts `Fa`, `Fb`, `Fc` etc. are calculated. These checksums can then be compared to the database generated in step one, and if all the checksums match, then it can be assumed that font `Fa` is actually `CMR10.300` etc. The details of the algorithm for generating the checksums do not matter too much, so long as the algorithm is fast and leads to a distinct checksum for every character in the database. This same algorithm is then used to analyse the bitmap fonts within the PostScript file. For interest, FixFont generates its checksum by cycling through the bitmap character data in 4-byte segments, and keeps a running total of the summed values of each 32 bit segment as the final checksum.

The first of the two FixFont programs is called `makedb`; it is a Unix shell script and it is responsible for creating the font database. It works by creating a LaTeX file for every font on the system. Thus, for example, a file named `CMR10.300.tex` is created which contains ASCII characters 0 to 127 for the `CMR10.300` font. Similar files are created for all other fonts in the (La)TeX installation. Each file is then processed by LaTeX and `dvips` to generate a 'font-sampler' PostScript file which contains a hexadecimal string representation of the bitmaps for the first 128 characters in the font. The resulting PostScript file is analysed by `makedb` and used to generate a checksum for all the character glyphs. Although the fonts in the intermediate PostScript will be called `Fa` (or similar), the `makedb` script will have created the original LaTeX file and named it `CMR10.300.tex`. Therefore the resulting PostScript file will be named `CMR10.300.ps`. The `makedb` script can now relate the glyph checksums to the original METAFONT file name in order to build up a list of checksums for `CMR10.300`. Similar information is collected for all fonts on the system and stored in a simple data file. Although FixFont refers to this as a 'database' it lacks the internal structure of a conventional database; the term *datafile* would be a more accurate description.

Once the database has been created, the second FixFont utility can be used to perform

the substitution. This second utility is a C program called `substitute` which modifies an existing PostScript file by replacing the embedded Computer Modern Type 3 fonts with their Type 1 equivalents. The PostScript file is analysed by `substitute` and, using the same algorithm as `makedb`, it creates a checksum for all the bitmap Type 3 characters within it. These are then compared against the database on a font-by-font basis, and, if there is a match, the Type 3 font is removed from the PostScript file and replaced by its corresponding Type 1 font. If the Type 1 fonts are already present on the system running `substitute`, then the bitmap fonts are replaced by embedded Type 1 fonts. If the Type 1 fonts are not present on the system, then the bitmapped fonts are replaced by *references* to the Type 1 fonts (which must then be available at the time the new PostScript file is printed, displayed or distilled). If an exact match is not attainable across all the characters in any particular font then the `substitute` program can be configured to perform fuzzy matching. Configuration options include performing substitutions only when a given percentage of glyphs appears to match a given font, or only when a given number of glyphs match the database. A pre-made database is distributed with FixFont.

6.2 FixFont's output

The PostScript file generated by `substitute`, containing Type 1 outline fonts, must, aside from an obvious improvement in font quality, render the page identically to the original bitmap-font PostScript file. To achieve this, a simple Type 3 for Type 1 font substitution is insufficient; it is also necessary to make additional alterations to the output PostScript. Suppose, for example, that the initial (LA)TEX file had used `CMR10` at 10pt. The `CMR10.300` font would have been embedded into the PostScript. However, this font has been designed to display at a point size of 10 on a 300 dpi device, without the need for scaling. PostScript, on the other hand, uses a coordinate system with 72 dots per inch, so to make the font display at the correct size a scale factor of 72/300 is applied to the glyphs by `dvips` in the original Type 3 PostScript file. If we now replace the Type 3 font with a Type 1 font we must scale the new font to make it appear at the same size. Firstly, it must be scaled by 300/72 (i.e. 4.16) to counteract the factor of 72/300 originally applied to the Type 3 font. Secondly, it needs to be scaled up by a factor of 10 to account for the fact that Type 1 fonts are designed to display glyphs at point size 1, whereas the Type 3 font `CMR10.300` is designed to display characters at point size 10. The first of these scaling factors is applied via a PostScript Font Matrix and the second via the scalefont operator. As an example, if the Type 3 font `CMR10.300` is replaced by the Type 1 font `cmr10` the following lines would need to be included in the new PostScript:

```
/DvipsFontMatrix [ 4.16 0 0 -4.16 0 0 ] def
/Fa { /cmr10 findfont 10.00 scalefont DvipsFontMatrix makefont setfont } def
```

In common with many other typesetting systems (LA)TEX works on a coordinate system with origin at the top left-hand corner and where y is positive as one goes *down* the page. PostScript, on the other hand, uses classic Cartesian coordinates with $(0, 0)$ at the bottom left-hand corner and where y is positive upwards. It is perfectly possible, in PostScript, to set up a transformation for it to use a typesetting coordinate system, but the only problem is that the characters themselves will then render upside down if, internally to the character cell, they are using the conventional Cartesian system. Things can be put to rights very easily by applying a Font Matrix to the bitmap font, which reverses the y direction; this explains why the fourth value in the above font matrix is -4.16 rather than 4.16.

Occasionally a point size will be requested that is not in the commonly-occurring range from about 6 pt to 20 pt. Under these circumstances many (LA)TEXsystems will try to activate METAFONTto create a new bitmap font 'on the fly'. If this fails for any reason then `dvips` has to decide which of the available CM point sizes, and at which resolution, would scale most elegantly to give the desired output. The PostScript code for the correct scaling must then be generated. For example, if the file requires CMR output at point size 37, one way to do this would be to specify `CMR10` at 37pt. In this case let us suppose, hypothetically, that `dvips` chooses to use the `CMR10.432` font and scales it by a factor of 2.57. If this were the case then when FixFont replaces `CMR10.432` by the Type 1 font `cmr10`, the latter needs to be scaled by $(300 \times 10)/72 \times (432/300) \times 2.57$.

Although FixFont manages the rewriting of PostScript very adroitly, its total dependence on a release of `dvips` which is now 7 years old illustrates the general difficulty of having to archive appropriate intermediate processing software if one wishes to regenerate a PDF from any of the intermediate processing stages shown in Figure 1. More recent releases of `dvips` have adopted a strategy of embedding a PostScript comment ahead of every embedded Type 3 font to help post-processors to recognise which font is in use. Unfortunately, rewriting FixFont to pick up this comment would not be sufficient to bring it up to date, because `dvips` has also totally changed the PostScript syntax it uses to embed Type 3 fonts and thus FixFont's parser can no longer analyse the embedded font data.

It is very obvious that upgrading FixFont would be a never-ending process which still could not overcome the fact that it will always be reliant on features that appear *only* in `dvips`-produced PostScript. In no sense is it a generalised PostScript analyser and the sheer variability of the PostScript likely to be produced by any other text processing software employing bitmap fonts would make such a generalised analyser extremely difficult to write. A more fruitful way forward seemed to lie in writing a plugin to perform font replacement within a final-form PDF file.

7 Font replacement in Acrobat

Previous sections have highlighted the difficulties of trying to regenerate archived material from source text or intermediate code files: in the specific domain of (LA)TEX-generated PDF material with embedded bitmap fonts, it will only occasionally be the case that the original (LA)TEX source, `dvi` and PostScript files are also available.

Our motivation in writing an Acrobat plug-in, which we call FontRep, to perform a similar job to FixFont, was prompted by the prospect of eventually developing a general method for replacing bitmap fonts within PDF files. But there are other clear advantages to be gained from creating such a plugin. PDF is based on Level 2 PostScript and the way PDF is used in output from Acrobat Distiller relies on a set of procedure definitions and dictionary entries that correspond closely to those used by Adobe Illustrator. Thus, the conversion of PostScript to PDF, by Distiller, effectively 'normalises' the PostScript. By analysing this more standardised output format it should be possible to perform the font replacement operation, initially, on a wider variety of (LA)TEX-generated documents and eventually to generalise the methods to cope with bitmap font replacement in PDF files generated from sources other than (LA)TEX.

7.1 The FontRep plug-in for Acrobat

The FontRep plug-in, for the full MS-Windows version of the Adobe Acrobat viewer (release 5.0) works in a similar manner to the FixFont application for PostScript described above. The database must be generated in the same way as before, but font substitution occurs on PDF files rather than PostScript files. As with FixFont, once the database has been created the first step in the substitution process is to locate every Type 3 bitmap font in the PDF document. As each font is encountered, the bitmaps for every character in the font are extracted. If necessary, filter decoding and bit manipulation are applied to each character to obtain a hexadecimal representation of the bitmaps. From here, generation of a checksum, and comparison against the database, takes place very much as with FixFont.

It is worth noting that when PDF is generated by Acrobat Distiller it changes `dvips`'s obscure font names such as `Fa`, `Fb` to the equally obscure `T1`, `T2` etc. The result, therefore, of comparing these embedded fonts to the canonical ones encapsulated within the database is the knowledge that the bitmap font `T1` equates to `cmr10`, say, or that `T2` is `cmbx12`. Acrobat API (Application Programming Interface) calls can then be used from within the plug-in to search the system fonts on the host machine and, if a matching Type 1 font is present, it can be embedded into the PDF and the corresponding bitmap font removed.

Unfortunately, as with FixFont, identifying and replacing fonts is only a part of the problem. As before, the Type 1 font's innate font dimensions and units cause PDF content streams to render at a very different size to that seen with the bitmap Type 3 font being replaced. This was overcome in FixFont's regenerated PostScript by including font

matrices to scale the Type 1 character widths. Sadly it is not so easy in PDF, which has only a subset of PostScript's capabilities and which, in particular, does not permit a font matrix to be associated with a PDF Type 1 font object. There are two initially obvious strategies for overcoming this problem: the first is to scale the glyphs within the font itself before embedding; the second is to rewrite the content streams within the PDF, scaling the characters as required. The former method has the advantage of leaving the PDF content streams untouched and so it was attempted first, but it was foiled by the internal design of Acrobat. Scaling the glyphs within a font, by using a font matrix prior to embedding, means that the width of each glyph increases. Glyph widths in Type 1 fonts are specified in units of 1/1000th pt, and so widths are in the region of 100 to 1000 units for commonly occurring characters. Scaling these by a factor of $300 \times 10/72$ (for `CMR10`) or $300 \times 12/72$ (for `CMR12`) means that the widths of the glyphs are transformed to a region between 4000 and 40000 units for `CMR10`, and even higher for fonts designed at larger point sizes. Now the PDF font format requires an array of character widths to be populated for all fonts, but the internal type used by Acrobat is an array of signed 16 bit integers, meaning that the maximum possible width for a glyph in Acrobat is 2^{15} - 1 (32767). Trying to input character widths greater than this causes overflow, leaving a residue that is an apparently negative quantity. This, in turn, causes the PDF to render incorrectly, with the most noticeable effect being that wide characters such as 'M' can shift a few centimetres to the left of where they ought to be.

The second possibility, of doing the scaling via a rewriting of PDF content streams, was attempted next. With the use of API calls a scaling matrix could be applied to text runs within the PDF content stream. Unfortunately, some minor bugs in certain API calls resulted in badly formatted content. The only solution to the problem was for the plug-in to rewrite the content streams itself, without relying on the API calls. To do this the plug-in had to decompose the PDF content stream into tokens, with each token consisting of a single content object, operator or argument. The tokens could then be analysed, added to or altered, for example a scaling factor could be applied, or existing matrices altered, before the revised PDF content stream was output.

This latter approach has the added benefit that more control is available over the output. It becomes easier to create an option whereby users are asked what action should be taken in the case of uncertain glyph matches. For example, suppose 53 out of 54 glyphs in font `T1` are recognised as being from the Type 3 bitmap font `CMR10.300`. In this case the Type 1 `cmr10` is embedded and the user is asked whether the one remaining glyph should be left as a Type 3, or mapped to a Type 1 glyph on the assumption that it is the standard character at that font position in TEX encoding.

The release of Acrobat 5.0 appears to have fixed some of the problems with the API, but the success of the tokenising approach has meant that this is still a very promising strategy.

7.2 Other font replacement strategies

The font replacement strategies described in the previous sub-section involve the creation of reconfigured Type 1 fonts, for the entire CM family, in Adobe `.pfa` format. The reconfiguration involves adjusting the font matrix within the font file to account for the various scalings and axis inversions that have already been described. Once this has been done the fonts are installed as system fonts and the Acrobat API for Windows has a method which imports a system font and allows it to replace the corresponding Type 3 bitmap font already present.

Although the strategy of rewriting the text streams has met with some success there are still some niggling problems with character positioning and the rather worrying prospect that the tokeniser has to be 100% accurate in recognising the intended point size of each component in the text stream before a correct text matrix can be applied.

From almost every viewpoint a solution where simple font substitution can be performed, with no rewriting of text streams, is greatly to be preferred. We have already discussed, in section 3, that a Type 3 font is not constrained to having bitmap character glyphs; it could equally be an unhinted set of outlines for drawing the character shapes. Thus, if the CM Type 1 fonts could be regenerated as Type 3 outline fonts, either directly from METAFONT or via reconfiguring a Type 1 CMR font in a tool such as Macromedia Fontographer, then it might be possible to substitute a Type 3 bitmap font with an equivalent Type 3 outline.

Initial testing of this idea is under way and early results seem promising. Although Type 3 fonts are not allowable as system fonts under Windows, it turns out that an API call of CosObjCopy allows an object to be copied over from a given PDF file to replace an object in another PDF. By creating a PDF consisting entirely of embedded PDF Type 3 CM outline fonts, we thereby create a resource from which the diffent fonts (complete with the appropriate font scaling matrices) can be abstracted, for use as bitmap Type 3 font replacements within the target PDF.

The appearance of the PDF once Type 3 outline fonts have been substituted for Type 3 bitmaps, is certainly improved. This is helped in large part by a new feature in the Acrobat 5 viewer software that allows line art to be smoothed. One has to remember, here, that the drawing routines for character shapes in a Type 3 font charproc entry are not interpreted in a specialised way, as they would be in a Type 1 font. They are simply treated as vector drawing (i.e. line art) routines. However, this smoothing of the character shapes cannot call on the hinting information of Type 1 fonts. Perhaps for this reason, and to avoid delicate hairline strokes disappearing altogether, the smoothing errs on the side of 'rounding up' rather than 'rounding down'. The overall effect of this is that text in outline Type 3 CMR appears 'bolder' than its Type 1 equivalent.

Prior to a first release of FontRep we are evaluating the replacement accuracy and the glyph quality offered by this alternative approach as opposed to the text stream rewriting described in section 7.1. The FontRep home page [14] gives details of our progress.

8 Problems encountered

With the aid of the checksum and the replacement font databases any (LA)TEX-generated PDF with bitmap CM fonts can, in principle have the bitmap fonts replaced by outlines. The CM glyphs, in common with the Schoolbook designs on which they are based, have rather thin stems and some users of (LA)TEX systems exploit the so-called modedefs capability of METAFONT to create replacement glyphs with heavier stem weights. If this is done then the checksum match against our database will fail for any glyph that has been altered in this way. Another constraint is that the success of the checksum method in recognising bitmap CM character glyphs relies on METAFONT having been invoked in a generic manner, for the resolution and point size required, independent of any particular output device (i.e. with 'null' printer settings). So far, on a relatively small test sample of 20 or so (LA)TEX-produced PDF files, using CM bitmap fonts in the 300 and 600 dpi ranges, we have only encountered two files where none of the fonts was recognised. A far more common occurrence is that a large proportion of the glyphs within a given font can be matched but not all of them. This could be due to something as subtle as rounding effects, when executing METAFONT and `dvips` on different machine architectures, or the more gross mismatch occasioned by characters being remapped to different positions within the embedded version of the bitmap font as opposed to the 'canonical' ordering expected within the bitmap font database. We have certainly seen cases where the quote mark characters and the 'fi' and 'ffi' ligatures have been permuted around.

At the moment the version of FontRep which rewrites text streams leaves any unrecognised glyph as an inserted bitmap character but an alternative strategy (as implemented in FixFont) would be to substitute the entire replacement font if more than a pre-set proportion of the Type 3 bitmap glyphs match the database checksums.

There is no doubt that, as FontRep develops further, more flexible font recognition and glyph-matching procedures will be needed. Unusual combinations of point size and resolution requests in the (LA)TEX source document will cause METAFONT, if available, to try and construct such a font 'on the fly' and it is clearly not feasible to expand our database indefinitely to encompass this infinity of possibilities.

9 Wider applicability

The foregoing sections have shown that it is not sufficient simply to identify the typeface implemented within a set of bitmap fonts and hope that some commercially available outline version of the same typeface will be a suitable substitute. It is also necessary to analyse the bitmap fonts carefully for their resolution, character encoding and glyph metrics to ensure that the substituted outline font will be truly compatible. Any evidence on these matters that is gleaned simply from the embedded fonts within a single PDF is likely to be

very sketchy — the source and the characteristics of any new bitmap font families must be carefully identified.

Nowadays the leading operating systems such as Windows, MacOS X and Sun Solaris have virtually no bitmap fonts; they distribute a wide range of outline fonts as part of the standard environment. Any PDF files prepared with these latter fonts embedded will generally render well on screen and cause few problems. However, the one remaining popular operating system that is a potential source of bitmap-font PDF files is Linux, largely because a common factor in almost all distributions of Linux is the use of the GhostScript environment as the common meeting point for printer-driver software.

In particular, GhostScript is capable of creating and previewing PostScript and PDF files, but its default font environment consists of a collection of public-domain fonts from a variety of sources. Although some of these fonts are Type 1 outlines, the great majority are Type 3 'bitmap clones' of popular typefaces such as Times and Helvetica. Recent releases of GhostScript do implement a Type 1 rasteriser but the responsibility of purchasing Type 1 fonts and installing them in the system, to replace the Type 3 clones, rests entirely with the user. The `ps2pdf` PDF output driver for GhostScript (for those who do not have Adobe Distiller) is just one instance of a strategy that enables a wide variety of non-PostScript printers to be driven from a PostScript-based canonical starting point, but in many cases the PostScript pages are simply converted into page bitmaps for the target printer with the problem that bitmap fonts used in GhostScript get carried over into the output.

Perhaps the largest specialist set of bitmap fonts commonly used with GhostScript are precisely the (LA)TEX bitmap fonts that are the focus of this paper. Since they are used in so much material now available on-line they formed an obvious subset for a first investigation into the PDF font replacement problem. But, as we have just seen, the other bitmap fonts in the GhostScript collection may well be employed by any other text processing software running under Linux, or even by (LA)TEX itself if users want to use fonts other than the CM family.

The methods described here are capable of being extended to other bitmap fonts distributed with GhostScript though increased care would have to be taken to ensure that the encoding vector of any substituted font matches that of the bitmap original.

10 Conclusions

Because it is such a useful print-on-demand format many publishers have chosen PDF for 'appearance based' electronic archiving. The need for maintenance and updating of such an archive is not confined simply to the quality of embedded fonts. The recent introduction of a structure tree option within PDF, from release 1.4 onwards, affords the possibility of intelligently re-flowing PDF for small-screen hand-held devices and for reading out the contents of a PDF to visually impaired computer users. It seems inevitable that many

archived PDF files will need to have a structure tree added as the years go by.

But even with the everyday issue of fonts in an unstructured PDF file, the fact that an Acrobat plug-in such as FontRep is needed at all, bears testimony to the great care that needs to be taken with the quality of the PDF that is stored away. Many users of (LA)TEX systems, including the publishers themselves, are only just realising the scale of the bitmap font problem as they move to making their archived (LA)TEX-produced PDFs available as e-books or other 'electronic' products. The best long-term solution is to use outline fonts from the very outset and, therefore, for (LA)TEX distributions to adopt the Blue Sky or BaKoMa outline fonts as their standard rather than the METAFONT bitmaps. There are encouraging signs that this is happening with recent LATEX systems such as OzTeX (for Macintosh) and MikTeX (for Windows). It is somewhat unfortunate, therefore, that the teTeX distribution for UNIX systems defaults to bitmap fonts and hides away the fact that outline fonts can be embedded, very simply, by giving a `-Pcmz` flag to `dvips`.

Given the very large amount of bitmap-font material already archived by academic and STM publishers, together with the amazing amount of bitmap-font PDF material still submitted as 'camera ready copy' to conferences, it is likely that the FontRep plug-in will be useful for several years to come.

11 Acknowledgements

Our thanks are due to Jeff Finger and Dick Sites, of Adobe Systems Inc., for their help with font recognition algorithms and tokenising of PDF. Thanks are also due to an anonymous referee for suggesting the approach of section 7.2 and to Peter Thomas for trying it out.

References

[1] "ISO Archiving Standards". `http://ssdoo.gsfc.nasa.gov/nost/isoas/`.

[2] D. E. Knuth. *The TEX book* — volume A of *Computers and Typesetting*. Addison-Wesley, Reading MA, 1986.

[3] Leslie Lamport. *LATEX: A Document Preparation System (2nd edn)*. Addison-Wesley, second edition, 1994.

[4] D. E. Knuth. *The METAFONT book* — volume C of *Computers and Typesetting*. Addison-Wesley, Reading MA, 1986.

[5] "The Ghostscript home page". `http://www.cs.wisc.edu/~ghost`.

[6] Adobe Systems Inc. *PostScript Language Tutorial and Cookbook*. Addison-Wesley, Reading, Massachusetts, 1985.

[7] Adobe Systems Inc. *Adobe Type 1 Font Format.* Addison-Wesley, Reading, Massachusetts, 1990.

[8] "The Blue Sky Type 1 CM fonts". Y&Y, 106 Indian Hill, Carlisle, MA01741 `http://www.yandy.com/cm.htm`.

[9] Basil Malyshev. "The BaKoMa fonts". Available from CTAN archives as `CTAN:fonts/cm/type1/bakoma`.

[10] D. E. Knuth. Device-independent file format. *TUGboat*, 3(2):14–19, 1982.

[11] Tomas Rokicki. *DVIPS: A TEX Driver*, January 1993.

[12] Michel Goossens, Sebastian Rahtz, and Frank Mittelbach. *The LATEX Graphics Companion.* Addison-Wesley, 1997.

[13] "TEX and PDF: Solving Font problems". Mirrored at `http://ecco.bsee.swin.edu.au/text/texpdf/texpdf-ffont.html#FIXFONT`.

[14] "The Fontrep home page". `http://www.eprg.org/research/`.