Incorporating PostScript fonts in TEX*

Erik-Jan Vens

Rijks Universiteit Groningen
ICCE
P.O.Box 335
9700 AH Groningen
the Netherlands
E.J.Vens@icce.rug.nl

Abstract

pfb2mf provides the TEX community with an interface to the PostScript Type One fonts. There is an overwhelming amount of these fonts for sale and there are a lot of fonts in the Public Domain, so it extends the range of typefaces the TEX user can choose from.

1 The typical look of a TEX-ed document

1.1 Some of my background

As a novice TEX[1] user, one will often be very enthousiastic about the program. At least, from those days when I was a rookie TEX user, I remember a sort of passion for what was possible with this incredible program. It was in 1985 and our university's computing centre had a Canon BLP-8 hooked up to their VAX 8650. I was a student then and used to work there almost every evening, sometimes nights. There was this typical small group of people working in the only room which was open to general users. And typically we would gather around the coffee-machine and discuss all sorts of things.

One of us knew about this program called TEX. He showed me some results and I was so impressed that the next day I visited the library to pick up 'The TEXbook'. (Needless to say, of course it was 'TEX and METAFONT: New Directions in Typesetting', so it was of little help.) Anyways, I spent many hours trying to create masterpieces of the publishing art. I was very enthousiastic about the program.

And I became even more enthousiastic when I met fellow TEXers, or saw publications made with TEX. But, how did I see these were made TEX? It was precisely because they were all typeset with the Computer Modern font family.

1.2 How and why did Knuth create the Computer Modern font family

When Knuth decided to write TEX, he wanted to be able to finish his series 'The Art of Computer Programming' with digital typography. As he writes in 'The Errors of TEX': 'The genesis of TEX probably took place on 1 February 1977, when I first chanced to see the output of a high-resolution typesetting machine. I was told that this fine typography [. . .] was produced by entirely digital methods; yet I could see no difference between the digital type and 'real' type. Therefore I realized that a central aspect of printing had been reduced to bit manipulation. As a computer scientist, I could not resist the challenge to improve print quality by manipulating those bits better. [. . .] By 13 February I had changed my plan to spend a forthcoming sabbatical year in South America; instead of travelling to an exotic place and working on Volume 4 of *The Art of Computer Programming*, I had decided to stay at Stanford and work on digital typography.'

He also needed a typeface. He chose Monotype Modern No. 8A, of which he says in 'Computers & Typesetting, Vol. E': 'In letterpress printing, modern fonts were technically troublesome because their delicate hairlines and serifs were particularly susceptible to damage; the broken letters produced a degraded text image. In digital typography with sufficient resolution, the delicate forms of modern can be rendered precisely. Computer typesetting and photolithographic printing are capable of reproducing modern typefaces with a clarity and sharpness unobtainable at the time of the original development of the style'.

Undoubtly, this is true. Anyone who has ever seen a text set in Computer Modern come out of a phototypesetter, knows this striking feeling of beauty. Only, most applications of TEX that I have seen, do not end up on a 2400 dpi phototypesetter. The typical output resolution is 300 dpi for a laser printer, 240 dpi for a dotmatrix

*Presented at EuroTEX '92, September 14–18, Prague, Czechoslovakia.

[1] Just as some people do mention that they denote 'he and/or she' by just using 'he', my usage of 'TEX' means 'TEX and/or LATEX'.

printer and 400 dpi for the NeXT laser printer. So choosing fonts that render well at low resolutions is certainly worthwhile.

1.3 Why there are no serious attempts at creating other faces

Creating a typeface from scratch is an extremely difficult task that takes lots of time, patience and may be energy. The only other complete family of typefaces designed with METAFONT is Neenie Billawala's Pandora. This is not just a recreation of an existing font, it is a series of tools, drivers and parameters that make up a family of faces, as she has described in TUGboat, Vol. 10, (1989), No. 4, pp. 481–489.

I am still working on a font together with a friend, where I do most of the programming and he does most of the designing. But this is our long, cold winter's sunday afternoon project, so its not nearly finished, even though we started out one-and-a-half-year ago.

And there are more people who have done some work in this area. Yannis Haralambous did some. He and other people have created new fonts, notably for Indian languages, Greek, Hebrew and Russian. But the TEX community is not a very large one, and so the field of font creation remains a largely undiscovered one.

2 The wealth of the world of PostScript fonts

But may be we do not need to wait for new fonts written in METAFONT. One of the many attractions of the PostScript world is its large range of available typefaces. Even in the Public Domain one can find numerous nice looking fonts. And if you consider buying them, there is such a large choice, that it becomes virtually impossible to get to know them all before taking your pick. So one tends to start at the traditional, standard PostScript look of a document.

2.1 The standard Times Roman plus Courier document

In much the same way that a TEXed document can be recognized by the use of Computer Modern, can a standard PostScript document be recognized by the combination of Times Roman and Courier, with Courier being slightly too thin next to the Times. Most people who come from the world of computer science know this combination from books as Kernighan & Ritchie's 'The C Programming Language' and 'Kernighan & Pike's 'The UNIX Programming Environment'.

Other traditional PostScript fonts include AvantGarde, Bookman, Helvetica, New Century Schoolbook, Palatino and some loose fonts as Zapf Dingbats and Zapf Chancery.

2.2 But there are more fonts

The past few years –as fonts grew cheaper and cheaper– many users have got used to be able to buy fonts. If one looks at the Top 40 bestsellers list of Adobe, one will notice that both in Europa and in the USA, Adobe Garamond is in the Top 10. On the other hand, Neville Brody's Arcadia, Industria and Insignia are the number one in the USA, where in Europe they rank only at 27. In Europe e.g. the GillSans ranks very high. Even in the 'Jobs' section of a widely read Dutch newspaper one can find many, many uses of GillSans. Even Flora and Praxis (both by Dutch designer Gerard Unger) can be found often.

3 Combining the worlds of TEX and PostScript fonts

Wouldn't it be nice if the user of TEX could incorporate the use of PostScript fonts in a TEX document? TEX still is one of the best, if not **the best** where the production of neatly spaced text and beautifully typeset mathematics is concerned. We –as TEX users– shiver in awe when we are confronted with the Swiss cheese output produced by WorstPervert, let alone the `eqn`-based formula editor it now provides. We don't even want to think about WordStar, and we have seen instances where the output of Ventura doesn't look extremely bad. We have heard that Word4Windows is nice. But still, TEX outranks them when pure text and mathematics is concerned.

3.1 Choosing a scheme

There are several possibilities open to the non Computer Modern user.

1. Completely switch to PostScript output. There is a Public Domain PostScript emulator, written by L. Peter Deutsch, in the GNU distribution. One will also have to use a `dvi` to `ps` translator, e.g. Tom Rokicki's `dvips`.
2. Use Piet Tutelaer's `ps2pk` package, based on the X-Windows, Release 5 distribution, which can translate a PostScript Type One font to TEX's `pk` format. This is extremely useful for those users who want fast, production quality fonts.
3. Use my `pfb2mf` package, which can translate a PostScript Type One font to METAFONT format. To create a bitmap, one will still have to run METAFONT. This last part of the job can be frustratingly slow, but it does provide the TEX user with options to change the output.

Beware. A caveat should be here. Producing beautifully typeset documents is just combining spaces and faces. But remember that this is an art, rather than a job which can be automated.

4 What does a METAFONT file look like

A METAFONT file consists of a combination of commands to set up the communication with TEX, commands to set up plottable functions and commands to set up a plot.

4.1 The macro-biotic aspects of METAFONT

Just as with TEX, macros play an important role. And just as with TEX, macros are essentially built up from other macros and primitives. Only because with TEX one has to make a distinction between pure text and macros, the macros have to be escaped with a back-slash. METAFONT macros on the other hand do not need the escape. And METAFONT macros can even handle binary operators. An example on p. 178 of 'The METAFONT book' states:

primarydef$w$ dotprod$z$ =

$(\mathrm{xpart}w * \mathrm{xpart}z + \mathrm{ypart}w * \mathrm{ypart}z)$ **enddef**.

I will not discuss this in in-depth, but basically this defines an infix operator '`dotprod`' which takes a left and a right operand, it then returns a value.

There a couple of important macros and primitives:

- '`beginchar`' sets up the beginning of the definition of a glyph. It takes four parameters:
  1. The character number, this can be given as a decimal, octal or hexadecimal number, or as the representation of the character, e.g. `"a"`.
  2. The width of the surrounding box. The character's shape can stick out of its box, but this box provides the metrics with which TEX will do its calculations.
  3. The height of the surrounding box. This is the height the characters extends above the baseline.
  4. The depth of the surrounding box.
- '`endchar`' declares the end of the definition.
- One can assign values to constants and variables. $u := 3.5$, or $width\# := 3.5pt\#$, where the # denotes a sharped, i.e. reallife, or not-rounded-to-the-raster value.
- One can declare points in space, or just their x- and y-values: $x_3 = x_2 - x_1$, or $z_1 = (\frac{1}{2}width, .8h - 3.5pt)$.
- The points can be connected with several types of lines. E.g. a straight line: $z_1 -- z_2 -- z_3$, or a smooth curve: $z_1 .. z_2 .. z_3$.
- One can draw these lines with a pen, or one can fill the area that is defined by a shape. Or combining these two. One can also erase a line, or an area. `draw` $z_1 -- z_2$ draws a straight line from z_1 to z_2 with the current pen. `fill` $z_1 .. z_2 -- z_3 .. cycle$ fills an area defined by a smooth line from z_1 to z_2, from there with a straight line to z_3, and from there with a smooth line back to the beginning.

5 What does a Type One font look like

We will have to do some work before we can take a look at Type One fonts with a plain text editor.

5.1 From `pfb` via `pfa` to `ps`

Type One fonts typically come in two flavors: binary or ascii representation. The binary representation is just an encoded form of the ascii representation. So, one needs a program to translate from the first to the last form. In the `pfb2mf` package that would be `pfb2pfa`. The ascii version, however, is a readable hexadecimal version of an encrypted series of PostScript commands. Once this encryption was secret, but it is now given to the world. So, one can translate the ascii version to human-readable PostScript. In the `pfb2mf` package this would be done with `pfa2ps`.

5.2 What does it mean

All Type One fonts are outline fonts which are filled. The PostScript language uses a reverse polish notation for its commands, so '`dx dy rlineto`' means 'draw a straight line from the current point to the point which lies at a distance (dx, dy). There are very few commands:

- '`endchar`' finishes a charstring outline definition.
- '`seac`' makes an accented character from two other characters in the font program.
- '`closepath`' closes a subpath. (Surprise!)
- All sorts of drawing and moving commands. These are all relative to the current point. Some commands are abbreviations for others, one can say, e.g. `dx hmoveto`, which means `dx 0 rmoveto`. All lines are straight lines. The curves are Bézier cubics, so `dx1 dy1 dx2 dy2 dx3 dy3 rrcurveto` means 'construct a curve from $currentpoint$ to $currentpoint + (dx1 + dx2 + dx3, dy1 + dy2 + dy3)$, using $currentpoint + (dx1, dy1)$ and $currentpoint + (dx1 + dx2, dy1 + dy2)$ as control points.
- Then there are the stem commands. This is the most difficult to understand part of a Type One font. I am still not sure whether I really do. Anyways, stem commands define zones within which the renderer can choose a position so that the output will look good. It is strongly related to METAFONT's commands for 'Discreteness and Discretion'.
- We can define subroutines and call them from within the Type One font program. So, any translation program must learn these subroutines and apply them when called.
- '`setcurrentpoint`' sets the current point, for how else can we start a curve where each point is defined in terms of its predecessor?

6 The uninteresting bits and pieces

The definition of how this is all done is not very interesting, so I will give just one example and leave it to the reader to pick up the sources if this reader is interested in more details.

6.1 An example of the font

There is a freely distributable version of Times New Roman out on the net in TrueType format. I have translated this to Type One format. This is an example of what the fonts look like:

Times New Roman

6.2 An example of the letter A in sourceform

```
/A {7 722 hsbw 450 221 rmoveto -256
    hlineto -45 -104 rlineto -12 -26 -5
    -20 0 -12 rrcurveto 0 -10 5 -9 10 -8
    rrcurveto 9 -7 21 -5 32 -2 rrcurveto
    -18 vlineto -209 hlineto 18 vlineto
    28 4 18 7 8 8 rrcurveto 16 15 19 32
    20 48 rrcurveto 233 545 rlineto 17
    hlineto 231 -551 rlineto 18 -44 17
    -29 15 -13 rrcurveto 15 -13 21 -8 27
    -1 rrcurveto -18 vlineto -261 hlineto
    18 vlineto 27 1 17 4 10 8 rrcurveto 9
    7 5 9 0 11 rrcurveto 0 14 -7 23 -13
    31 rrcurveto -40 95 rlineto -14 36
    rmoveto -112 268 rlineto -115 -268
    rlineto 227 hlineto endchar } ND
```

```
beginchar(65,722*FX#,677*FY#,0*FY#); "A";
fill ((457,221) -- (201,221) -- (156,117)
     .. controls (144,91) and (139,71) ..
     (139,59) .. controls (139,49) and
     (144,40) .. (154,32) .. controls
     (163,25) and (184,20) .. (216,18) --
     (216,0) -- (7,0) -- (7,18) ..
     controls (35,22) and (53,29) ..
     (61,37) .. controls (77,52) and
     (96,84) .. (116,132) -- (349,677) --
     (366,677) -- (597,126) .. controls
     (615,82) and (632,53) .. (647,40) ..
     controls (662,27) and (683,19) ..
     (710,18) -- (710,0) -- (449,0) --
     (449,18) .. controls (476,19) and
     (493,23) .. (503,31) .. controls
     (512,38) and (517,47) .. (517,58) ..
     controls (517,72) and (510,95) ..
     (497,126) -- (457,221) -- cycle)
     xscaled FX# yscaled FY#;
unfill ((443,257) -- (331,525) -- (216,257)
     -- (443,257) -- cycle) xscaled FX#
     yscaled FY#;
endchar;
```

7 Does it meet realworld constraints?

No, as of yet, it doesn't. However, the process of hinting is time-consuming and therefor very, very slow. So, if it is fast production quality you are after, you are well advised to pick up Piet Tutelaer's `ps2pk`.

I want to extend the possibilities of the program to provide the user with ways of implementing meta-ness. So if you'd like to fiddle with the font's parameters, `pfb2mf` is what you want. It will provide a great deal of tuneable parameters, and you can lift out part of a font an use that as a basis for other things.

8 And where can I get all this?

Both `pfb2mf` and `ps2pk` are ftp-able. E.g. from my home machine `obelix.icce.rug.nl`. It can be found in `pub/erikjan`.

9 A warning

Be careful! If you plan on using more fonts, you will not only have to worry about your text, and the layout you choose, you will also have to worry about what font you are going to use.

Creating Shaded Rectangles with PostScript

David Salomon

11835 Sapota Dr.,
Lakeside, CA 92040, USA
dxs@ms.secs.csun.edu

June 1992

Abstract

One of the most common graphics used in documents is text with a shaded background. This is hard to do with TeX but easy with PostScript. Simple PostScript code is presented here to create shaded rectangles, and a macro is developed to combine such a rectangle with text.

1 Introduction

It is well known that TeX lacks facilities for graphics. Certain drawings, such as logos, may be developed in METAFONT, but for any non-trivial graphics this is usually too time consuming. A ruled box, like this, is the closest TeX can get to anything resembling a drawing. Slanted lines can be handled by typesetting a dot and moving it in small steps. Hendrickson (1985), Cameron (1985) and Salomon (1989) use this idea to develop simple macros for slanted lines. The same principle can be used to typeset curves, as done by `Bezier.sty` in LaTeX, and by the PiCTeX macro package (Wichura, 1986). For more complex graphics, the `\special` command can be used to include graphics in the final document. Rahtz (1989) is an excellent survey of the different methods used to combine TeX and graphics.

In my work I have often felt the need for enclosing text in ruled boxes, either rectangular or with rounded corners, with backgrounds of shaded gray, and sometimes with a thick stroke around them. Glendown (1989) is an (unsatisfactory) attempt to draw similar boxes using the arcs provided in the LaTeX circle fonts. As a heavy Macintosh user, it seemed to me that my best choice was to use Postscript to achieve such effects. Postscript printers are widely available, and Postscript programs are supported, via `\special`, by more and more TeX implementations. The program used here has the additional advantage of being small and fast.

2 The `\shade` Macro

The result of my efforts is the macro `\shade` below. It creates a rectangular box with either sharp or rounded corners around text. The text can be placed, by the user, in either an `\hbox` or a `\vbox`, or it can be written explicitly as the macro argument (see third example below). The rectangle is filled with the desired shade of gray, and is optionally surrounded by a stroke of any desired thickness. It is also possible to make the shaded area larger than the text by any amount. The macro has five parameters:

- A decimal number specifying the proportion of white in the shaded background.
  This is a .97 background.
- A dimension specifying the

  extra size of shaded area (12pt on each side).
- A count specifying the width of the stroke in points. A zero or empty argument creates no stroke.
- A count—the radius of the rounded corners, in points. A zero or empty argument produces square corners.
- The text to be boxed. It is either straight text (if it fits on one line) or is enclosed, by the user, in an `\hbox` or `\vbox`.

The arguments are separated by commas, and the last one is delimited by '`\\`'.

Here is the listing of `\shade`.

```
\newdimen\tmp \newdimen\tmpw
\newdimen\tmpk \newdimen\tmph
\newcount\heit \newcount\widf
\newcount\strk \newcount\radius
\def\shade#1,#2,#3,#4,#5\\{%
\setbox1=\hbox{#5}%
\def\inner{#3}\ifx\inner\empty \strk=0
 \else\strk=#3\fi
\def\inner{#4}\ifx\inner\empty \radius=0
\else \radius=#4\fi
\tmp=#2 \tmp=2\tmp \advance\tmp\wd1
 \widf=\tmp \divide\widf65536
\tmp=#2 \tmp=2\tmp \advance\tmp\ht1
```

```
 \heit=\tmp \divide\heit65536
\setshadebox{#1}{\the\heit}{\the\widf}
 {\the\strk}{\the\radius}%
\tmp=#2
\tmpw=#2 \tmpw=2\tmpw
 \advance\tmpw by\the\strk pt
 \advance\tmpw\wd1
\tmpk=\the\strk pt \divide\tmpk by2
\tmph=#2 \tmph=2\tmph
 \advance\tmph by\the\strk pt
 \advance\tmph\ht1
\vbox to\tmph{\vss
\hbox to\tmpw{\kern\tmpk\lower\tmp\box0%
 \kern-\tmpk\hfil\box1\hfil}\vss}}
% The ps commands are placed, as the
% argument of a \special, at the
% bottom of \box0 whose width is set
% to zero.
\def\setshadebox#1#2#3#4#5{%
\setbox0=\vbox to\tmp{\hsize=0pt\vfil
 \special{postscript
/fpops{4 {pop} repeat} def
#3 0 moveto 0 0 0 #2 #5 arcto
newpath % start the rounded-corner
        % rectangle
moveto % the coordinates are in stack
0 #2 #3 #2 #5 arcto fpops
#3 #2 #3 0 #5 arcto fpops
#3 0 0 0 #5 arcto fpops
0 0 0 #2 #5 arcto fpops closepath
#4 0 ne
 {gsave #4 setlinewidth stroke
  grestore} if
% stroke width of zero not recommended
#1 setgray fill}}}
```

The `\special` with the Postscript commands is placed at the bottom of `\box0`, which is set to width zero. The text is placed in `\box1`. Both boxes are then placed, side by side, in an `\hbox` set to the right width (the width of the text, plus the extra width of the shaded area and the stroke).

The macro is normally used in vertical mode, but also works well in horizontal mode, not introducing any spurious spaces (the percent signs at the end of certain lines should not be removed). It has also been tested in both math modes. Note that Postscript code should normally be surrounded by a `gsave`, `grestore` pair. However, I use TEXtures, which automatically supplies these commands.

3 Examples

The following examples show samples of input and their respective output:

```
\shade0.90,16pt,5,10,\vbox{\hsize=100pt
\noindent A test in vertical mode.
Filled with 90\% white \&
a 5pt stroke}\\
```

A test in vertical mode. Filled with 90% white & a 5pt stroke

```
\shade0.97,5pt,1,3,%
straight text, not pre-boxed\\
```

straight text, not pre-boxed

```
\shade0.90,5pt,,3,%
\hbox{No stroke at all}\\
```

No stroke at all

```
\noindent A\lower30pt\hbox{%
\shade0.98,10pt,1,5,
\vbox{\hsize=100pt
 A test in horizontal mode. There are
no spaces between this text and the
'A' or the 'D' (98\% white)}\\}D
```

A very heavy stroke

```
\shade0.99,8pt,40,,%
\hbox{A very heavy stroke}\\
```

A A test in horizontal mode. There are no spaces between this text and the 'A' or the 'D' (98% white) D

```
\shade0.5,5pt,1,5,\hbox{\shade.99,5pt,1,3,%
 A Nested Expansion\\}\\
```

A Nested Expansion

4 Using the Macro in Practice

In practice, one usually uses shaded boxes of the same type throughout a document. This suggests using macros such as

```
\def\shadedbox#1{\shade0.95,1em,1,,#1\\}
\def\shadedRbox#1{\shade0.95,1em,1,5,#1\\}
```

instead of using of `\shade` directly.

The following examples below combine `\shade` with rules and leaders.

```
\vbox{\offinterlineskip
\shade0.97,4pt,1,,Section 3:\\
\hrule height4pt}
```

Section 3:

```
\hbox{\vrule width4pt%
 \shade0.95,4pt,,,Note!\\}
```

Note!

```
\line{\leaders\hbox{%
 \shade0.9,0pt,1,,\vrule height4pt width0pt%
 \kern6pt\\\kern4pt}\hfil}
```

```
\line{\leaders\hbox{%
 \shade0.9,0pt,1,2,\vrule height4pt width0pt%
 \kern6pt\\\kern4pt}\hfil}
```

References

[1] Cameron, A.G.W. “Wiggly Lines.” *TUGboat* **6**(3), pages 155–156, 1985.

[2] Glendown, G. “Rounded Boxes for `plain` TeX.” *TUGboat* **10**(3), pages 385–386, 1989.

[3] Hendrickson, A. “Some Diagonal Line Hacks.” TUGboat **6**(2), pages 83–86, July 1985.

[4] Rahtz, S. *A Survey of TeX and Graphics*. University of Southampton, Tech. Rep. CSTR 89-7, 1989.

[5] Salomon, D. “DDA Methods in TeX.” *TUGboat* **10**(2), pages 207–216, 1989.

[6] Wichura, M.J. *The PiCTeX Manual*. Number 6 in the *TeXniques* series. Providence: TeX Users Group, 1986.