mf2pt1

Produce PostScript Type 1 fonts from Metafont source

Scott Pakin, `pakin@uiuc.edu`

This file documents `mf2pt1` version 1.00, dated 18 June 2001.

Copyright © 2001 Scott Pakin.

This program may be distributed and/or modified under the conditions of the LaTeX Project Public License, either version 1.2 of this license or (at your option) any later version.

The latest version of this license is in:

`http://www.latex-project.org/lppl.txt`

and version 1.2 or later is part of all distributions of LaTeX version 1999/12/01 or later.

1 Introduction

METAFONT is a high-level, mathematically oriented language for producing fonts. The METAFONT interpreter produces device-dependent bitmaps, which render well at the target resolution on the target device, but poorly at other resolutions or on other devices. Adobe's PostScript Type 1 font format is the de facto font standard for printers these days. It is a vector format, which implies that it scales better than bitmaps, and it delegates the device- and resolution-dependent tweaking from the font source to the target device's PostScript renderer. However, Type 1 fonts are extremely difficult to code by hand. Usually, one uses a WYSIWYG program to design a Type 1 font. METAFONT, with its font-specific programming language, is an elegant alternative. A font designer can write reusable subroutines for repeated features, such as serifs and accents. He can define a font in terms of arbitrary parameters, such as "boldness" or "italicness", making it trivial to produce entire families of fonts from a single source (hence the "meta" in the name "METAFONT"). Ideally, we would like to design a font using the METAFONT language, but produce PostScript Type 1 output instead of bitmaps.

`mf2pt1` helps bridge the gap between METAFONT and Type 1 fonts. `mf2pt1` facilitates producing PostScript Type 1 fonts from a METAFONT source file. It is *not*, as the name may imply, an automatic converter of arbitrary METAFONT fonts to Type 1 format. `mf2pt1` imposes a number of restrictions on the METAFONT input. If these restrictions are met, `mf2pt1` will produce valid Type 1 output. (Actually, it produces "disassembled" Type 1; the `t1asm` program from the `t1utils` suite can convert this to a true Type 1 font.)

1.1 Requirements

Before using `mf2pt1`, you will need to install the following programs:

Perl
: `mf2pt1` is written in Perl. You will need a Perl interpreter to run it.

MetaPost
: `mf2pt1` actually uses MetaPost, not METAFONT, to produce PostScript output. Specifically, you will need the `mpost` executable and the '`mfplain.mp`' base file.

Type 1 utilities (`t1utils`)
: Producing properly encoded Type 1 fonts is tricky. `mf2pt1` delegates the effort to `t1utils`, specifically, to the `t1asm` program within that suite.

Perl is available from the Comprehensive Perl Archive Network (`http://www.cpan.org`). MetaPost and the Type 1 utilities are available from the Comprehensive TEX Archive Network (`http://www.ctan.org`). (In addition, MetaPost's home page is `http://cm.bell-labs.com/who/hobby/MetaPost.html`, and the Type 1 utilities' home page is `http://www.lcdf.org/type/`.)

I find that it also helps to have a WYSIWYG Type 1 font-editing program handy. Among other uses, this sort of program lets you add Type 1 "hints", which help produce quality rendering at low resolutions, such as on a computer monitor. I use PfaEdit (`http://pfaedit.sourceforge.net/`), which, besides being free, is capable of producing hints, outputting TrueType and X Window fonts, and doing various other things that complement `mf2pt1`.

1.2 Installation

To install `mf2pt1`, move the '`mf2pt1`' executable someplace where your operating system can find it. If you're running Microsoft Windows, you should rename the program to '`mf2pt1.pl`', so that Windows knows it's a Perl script. (Alternatively, if you have `pl2bat`, use that to produce a '`mf2pt1.bat`' file, which you can run as simply '`mf2pt1`'.)

The next step is to produce a '`mf2pt1.mem`' file from the supplied '`mf2pt1.mp`'. The command to do this differs from system to system, but it's usually something like this:

```
inimpost "\input mf2pt1; dump"
```

or this:

```
mpost -ini "\input mf2pt1; dump"
```

Move the resulting '`mf2pt1.mem`' file someplace where MetaPost can find it.

The `mf2pt1` documentation (what you're reading now) is written in Texinfo and can therefore easily be converted to a variety of formats:

PostScript ('`mf2pt1.ps`')

```
texi2dvi mf2pt1.texi
dvips mf2pt1.dvi -o mf2pt1.ps
```

HTML ('`mf2pt1.html`')

```
makeinfo --html mf2pt1.texi
```

Info ('`mf2pt1.info`')

```
makeinfo mf2pt1.texi
```

N.B. The `install-info` command is a convenient way to install '`mf2pt1.info`' on your system.

On Unix, you may also want to generate an `mf2pt1` man page. The man page is embedded within the '`mf2pt1`' Perl script and can be extracted with `pod2man`:

```
pod2man --center="User Commands" --date="18 June 2001" \
  --release="v1.00" mf2pt1 > mf2pt1.1
```

You can then move '`mf2pt1.1`' into '`/usr/man/man1`' or any other man page directory. Note that the `mf2pt1` man page is fairly rudimentary. It is primarily a list of the command-line options (see Section 2.2 [Font information], page 3) and a pointer to the document that you're reading now.

2 Usage

`mf2pt1` is fairly straightforward to use. To process a METAFONT source file, merely specify the filename:

```
mf2pt1 myfont.mf
```

The above command reads '`myfont.mf`', uses `mpost` to convert each character to a separate Encapsulated PostScript file (named '`myfont.`*num*'), and then merges those files into a single "disassembled" Type 1 font, called '`myfont.pt1`'. To convert '`myfont.pt1`' to a proper, binary, Type 1 font, just enter the following:

```
t1asm myfont.pt1 myfont.pfb
```

I then use PfaEdit (see Section 1.1 [Requirements], page 1) to add hinting information to the result, although this is by no means required.

2.1 Restrictions

If `mf2pt1` sounds too good to be true, it is, somewhat. `mf2pt1` is not a general-purpose METAFONT-to-Type 1 converter. Rather, it can convert only certain METAFONT constructs. This is not a showstopper for new fonts designed with `mf2pt1` in mind, but it is unlikely that `mf2pt1` will work on an arbitrary METAFONT source file.

`mf2pt1`'s biggest restriction is that each glyph must be drawn entirely from nonintersecting, closed paths, using METAFONT's **fill** and **unfill** commands. (`mf2pt1` has limited support for **draw** and **undraw**, but their use is currently strongly discouraged.) Note that there is a METAFONT library called '`roex.mf`', which is available from CTAN (`http://www.ctan.org/`). '`roex.mf`' defines a command, **remove_overlap**, which merges intersecting paths into a single path, and a command, strongexpand_stroke, which expands a stroke into a closed path using a circular pen. You may want to use these to work around `mftpt1`'s path restrictions.

A secondary restriction is that `mf2pt1` redefines a number of Plain METAFONT commands, such as **beginchar**, **fill**, and **unfill**. METAFONT programs that redefine or bypass these (using METAFONT primitives) will not be convertible with `mf2pt1`.

A far less severe restriction is due to `mf2pt1`'s utilizing MetaPost's METAFONT interface instead of METAFONT itself. The implication is that commands not defined by MetaPost's '`mfplain.mp`' cannot be handled by `mf2pt1`, either. Very few fonts will have a problem with this restriction, but see the MetaPost manual for more information.

2.2 Specifying font information

METAFONT fonts normally specify a set of **fontdimen**s, which provide information about a particular font that cannot otherwise be inferred. These include things like the font's x-height, quad width, interword stretchability and shrinkability, and other features that TeX makes use of. PostScript fonts utilize a largely different set of font parameters, such as the underline position and thickness, font family name, and copyright notice. `mf2pt1` provides METAFONT commands to define the PostScript font parameters in the generated Type 1 font. These parameters should appear in the METAFONT source file as follows:

```
if known ps_output:
  ...
fi;
```

`ps_output` is defined by `mf2pt1` but not by Plain METAFONT. Checking if it is known is the recommended way to determine if the font is being built under `mf2pt1`.

The following table lists all of the font information commands provided by `mf2pt1`, the command-line equivalent (discouraged), and what the command means. Commands preceded by an asterisk are also defined by Plain METAFONT and therefore do not need to be enveloped within a test for `ps_output`.

	font_coding_scheme	`--encoding`	The mapping between character numbers and PostScript names. `mf2pt1` recognizes only `standard` (Adobe standard encoding), `isolatin1` (ISO Latin 1 encoding), and `ot1` (TeX 7-bit encoding). Anything else will generate a warning message and cause `mf2pt1` to use `standard` instead.
	font_comment	`--comment`	A textual comment that will appear within the generated font. This is often used for copyright notices.
	font_family	`--family`	The family that this font belongs to. For example, “Utopia Bold Italic” belongs to the `Utopia` family.
	font_fixed_pitch	`--fixedpitch`	Whether the font is monospaced (**true** or `--fixedpitch`) or proportionally spaced (**false** or `--nofixedpitch`).
*	**font_identifier**	`--fullname`	The full name of the font, e.g., `Utopia Bold Italic`.
	font_name	`--name`	The symbolic font name, used to load the font from a PostScript document. Spaces are forbidden. Generally, the font name is of the form *family-modifiers*. For example, the font name of Utopia Bold Italic would be `Utopia-BoldItalic`.
*	**font_size**	`--designsize`	The font design size. This is specified in “sharped” units within METAFONT code or as a point size on the command line.

*	**font_slant**	`--italicangle`	When specified with **font_slant**, the amount of slant per point. When specified with `--italicangle`, the angle in counterclockwise degrees from the vertical (i.e., zero for an upright font, negative for a right-slanting italic font).
	font_underline_position	`--underpos`	The vertical position at which an underline should lie.
	font_underline_thickness	`--underthick`	The thickness of an underline.
	font_unique_id	`--uniqueid`	The unique ID for this font. The ID should be between 0 and 16,777,215. If not specified, `mf2pt1` will choose an ID at random from the "open" range, 4,000,000-4,999,999. All IDs not in that range are allocated by contacting Adobe's UniqueID Coordinator. (I don't believe a fee is involved, but I don't know for sure.)
	font_version	`--fontversion`	The version number of the font. This should be of the form *MMM*`.`*mmm*, where *MMM* is the major version number and *mmm* is the minor version number.
	font_weight	`--weight`	The font weight. For example, the font weight of Utopia Bold Italic is `Bold`.

In addition, the command-line argument `--help` outputs a brief description of `mf2pt1` and the other command-line arguments.

The next table lists the METAFONT type and default value of each of the parameters listed in the previous table.

font_coding_scheme	**string**	`"standard"`
font_comment	**string**	`"Font converted to Type 1 by mf2pt1, written by Scott Pakin."`
font_family	**string**	(The value of **font_identifier**)
font_fixed_pitch	**boolean**	**false**
font_identifier	**string**	(The input filename, minus '`.mf`')
font_name	**string**	(The value of **font_family**, plus an underscore, plus the value of **font_weight**, with all spaces removed)

font_size	**numeric**	(Crudely estimated from the font bounding box.)
font_slant	**numeric**	0
font_underline_position	**numeric**	-**font_size**/10
font_underline_thickness	**numeric**	**font_size**/20
font_unique_id	**string**	(Randomly generated in the range 4000000-4999999)
font_version	**string**	`"001.000"`
font_weight	**string**	`"Medium"`

The following METAFONT code shows the usage of all of the parameters that `mf2pt1` accepts:

```
if known ps_output:
  font_coding_scheme       "ot1";
  font_comment             "Copyright (C) 2001 Scott Pakin.";
  font_family              "Kerplotz";
  font_fixed_pitch         false;
  font_identifier          "Kerplotz Light Oblique";
  font_name                "Kerplotz-LightOblique";
  font_size                10pt#;        % Important to include this.
  font_slant               1/6;
  font_underline_position  -1pt#;
  font_underline_thickness 1/2pt#;
  font_unique_id           "4112233";    % Better to omit this.
  font_version             "002.005";
  font_weight              "Light";
fi;
```

In the above, the **font_fixed_pitch** call could have been omitted, as it defaults to **false**. Also, unless you've requested a unique ID from Adobe, it's generally better not to assign **font_unique_id**; let `mf2pt1` choose a random value itself.

The same parameters can also be specified on the command line as follows:

```
mf2pt1 --encoding=ot1 --comment="Copyright (C) 2001 Scott Pakin."
  --family=Kerplotz --nofixedpitch --fullname="Kerplotz Light Oblique"
  --name=Kerplotz-LightOblique --designsize=10 --italicangle=-9.5
  --underpos=-100 --underthick=50 --uniqueid=4112233 --version=002.005
  --weight=Light kerplotz.mf
```

Note that a METAFONT font slant of 1/6 is equal to a PostScript italic angle of approximately -9.5. The conversion formula is $s = -\tan\theta$, in which s is the slant and θ is the italic angle. In addition, the underline position and thickness must be multiplied by 1000/**font_size** to convert from the METAFONT units that are used within the '`.mf`' file to the PostScript units that are used on the command line.

3 Future Work

There are two features I am currently thinking about adding to `mf2pt1`: Type 1 hints and support for overlapping paths. Hinting is a way for a font designer to specify how a font should be rendered at low resolutions, for example, at typical monitor resolutions. In METAFONT, this is done by controlling the way that points are mapped to pixel locations, using commands such as **define_corrected_pixels**, **define_blacker_pixels**, and **lowres_fix**. Type 1 fonts are hinted in a completely different way. Type 1 hints distinguish key character features, such as stems and dots, from decorations that can be discarded at low resolutions. The PostScript interpreter uses that information to determine how to map points to pixels. I currently use PfaEdit (see Section 1.1 [Requirements], page 1) to add hints to my Type 1 fonts, but it would be a lot more convenient for `mf2pt1` to provide METAFONT commands for **hstem**, **vstem**, **dotsection**, and the other Type 1 hints. That way, hints will no longer need to be manually re-added every time `mf2pt1` regenerates a Type 1 font.

The second feature I'd like to add to `mf2pt1` is support for overlapping paths. The `roex` library for METAFONT provides a **remove_overlap** command that splits paths into nonintersecting subpaths and an **expand_stroke** command that converts a stroked, possibly open, path into a filled, closed path. I'm currently considering how to integrate **remove_overlap** and **expand_stroke** into **mf2pt1**, as this would enable more METAFONT fonts to be converted to Type 1 without modification.

In addition to Type 1 hints and support for overlapping paths, I also plan to make `mf2pt1` support more font encodings. The following are the encodings that `mf2pt1` will most likely accept:

`TeXMathItalicEncoding`
: Upper- and lowercase Greek and Latin letters, old-style digits, and a few symbols and accents.

`TeXMathSymbolEncoding`
: A variety of symbols, as well as calligraphic Latin majuscules.

`TeXMathExtensionEncoding`
: Variable-sized symbols, such as braces, integrals, and radicals.

`AdobeExpert`
: Small caps, currency symbols, old-style digits, and various superior and inferior letters and digits.