Metafont's mode_def in action

Erik Frambach

Rijksuniversiteit Groningen, Vakgroep Econometrie,
Postbus 800, 9700AV Groningen
e.h.m.frambach@eco.rug.nl

Abstract

In order to obtain maximum output quality when using METAFONT for rendering bitmapped fonts you need to specify the characteristics of the intended output device. This is done by defining a mode_def in which several variables are assigned. The meaning and effect of these variables are discussed in a case study of two types of laser printers.

Keywords: METAFONT, mode_def, bitmapped fonts.

1 Introduction

Being familiar with Hewlett-Packard's Laserjet family of laser printers, I was much surprised when I saw output from a Xerox Docutech printer the first time. The latter is a 600 dpi PostScript compatible laser printer, and as such comparable to the HP Laserjet 4M.

Using resident fonts such as Times Roman the differences are subtle though obvious. But when using e.g. Computer Modern fonts generated for the Laserjet 4M the differences are dramatic. Lines are much too thin or even seem to disappear at small font sizes.

Obviously, fonts for the Docutech system need to be generated with different METAFONT parameters. And indeed, after generating new fonts specifically for the Docutech the output looked much better, even better than the Laserjet's output. METAFONT did its job very well, using a mode_def specific for the Docutech printer.

2 What is a mode_def?

A mode_def is a definition of a series of assignments to various device-specific variables. It tells METAFONT how to compensate for certain device-specific characteristics. They are usually stored in a file called local.mf that is typically used when making METAFONT base file (iniMF). The file local.mf may contain many mode_defs, though you may want or need to restrict it to the locally used printer types to avoid exceeding METAFONT's capacity.

Now that we know that mode_defs can make the difference between good and bad output we will examine the variables and compare the mode_defs for the Laserjet and the Docutech.

```
mode_def ljIV =
  proofing:=0;
  fontmaking:=1;
  tracingtitles:=0;
  pixels_per_inch:=600;
  blacker:=0;
  fillin:=.2;
  o_correction:=.6;
  enddef;

mode_def docutech =
  proofing:=0;
  fontmaking:=1;
  tracingtitles:=0;
  pixels_per_inch:=600;
  blacker:=1;
  fillin:=.1;
  o_correction:=0.9;
  enddef;
```

The variables in these definitions are explained in the METAFONT book, but Karl Berry's explanation in his mode_def collection[1] is quite sufficient for non-experts (page numbers refer to the METAFONT book):

aspect_ratio: the ratio of the vertical resolution to the horizontal resolution (page 94).

blacker: a correction added to the width of stems and similar features, to account for devices which would otherwise make them too light (page 93). (Write-white devices are best handled by a more sophisticated method than merely adding to blacker, as explained above.)

fillin: a correction factor for diagonals and other features which would otherwise be 'filled in' (page 94). An ideal device would have fillin=0 (page 94). Negative values for fillin typically have either gross effects or none at all, and should be avoided.

fontmaking: if nonzero at the end of the job, METAFONT writes a TFM file (page 315).

o_correction: a correction factor for the 'overshoot' of curves beyond the baseline or x-height. High resolution curves look better with overshoot, so such devices should have o_correction=1; but at low resolutions, the overshoot appears to simply be a distortion (page 93). Here

[1] Available at ftp.umb.edu: pub/tex/modes.mf or at CTAN: fonts/modes/modes-2.1.mf.

some additional comments about `o_correction`, courtesy of Pierre MacKay (edited by Karl):

At present, I find that `o_correction` works nicely at 80 pixels per em, and gets increasingly disturbing as you move down towards 50 pixels per em. Below that I do not think it ought to happen at all.

The problem, of course, is that full `o_correction` is supposed to occur only at the zenith and nadir of the curve of 'o', which is a small region at high resolution, but may be a long line of horizontal pixels at medium resolution. The full `o_correction` does not change a 300 dpi cmr10, but it changes a 21-pixel high cmr12 to be 23 pixels high. The extra height and depth is seen along a line of seven pixels at the bottom and five at the top. This is a pronounced overshoot indeed.

For high-resolution devices, such as phototypesetters, the values for `blacker`, `fillin`, and `o_correction` don't matter all that much, so long as the values are within their normal ranges: between 0 and 1, with the values approaching 0, 0, and 1 respectively.

`pixels_per_inch`: the horizontal resolution; the METAFONT primitive `hppp` (which is what determines the extension on the GF filename, as among other things) is computed from this (page 94). (An 'inch' is 72.27 pt in the TEX world.)

To be more precise, you can determine the resolution of a font given a `mode_def` and a magnification `m` by simply multiplying `pixels_per_inch` for that `mode_def` by `m`. (Your results may differ from METAFONT's if you don't use equivalent fixed-point arithmetic.) Then you can determine the number used in the name of the GF font output by rounding. For example, a font generated at `magstep(.5)` (which is $\sqrt{1.2}$, which METAFONTcomputes as 1.09544) for a printer with `pixels_per_inch=300` will have a resolution of 328.63312 dots per inch, and the GF filename will include the number `329`.

`proofing`: says whether to put additional specials in the GF file for use in making proofsheets via, e.g., the utility program GFtoDVI (page 323–4).

`tracingtitles`: if nonzero, strings that appear as METAFONT statements are typed on the terminal (page 187).

Neenie Billawala's article in the April 1987 issue of *TUGboat* describes how to test your printer for the best set of values for the magic numbers above. Here are some brief comments on the subject, courtesy of Rocky Bernstein, again edited by Karl:

For medium-to-low resolution devices, you can first set the `blacker` and `o_correction` to 0 and decide on a `fillin` value by looking at the diagonal of a lowercase 'z' in `cmtt10`, or various lines in LaTeX's line10 font. The diagonal should be the same thickness as the horizontal bars of the 'z'. Then I decide on the `blacker` value, generally by checking the smaller fonts for too much filling in. Finally, you can set the `o_correction` using the guidelines suggested above.

End quote. From the Docutech values we can conclude that it tends to print much lighter, it doesn't fill in diagonals as much as a Laserjet, and the overshoot is nearly ideal.

3 The proof of the pudding

To determine if the variables are set correctly you need printed copy. Unfortunately that is not possible in an article such as this. However, to proof that the different values do effect METAFONT's output we can use the MS-DOS program PK2BM[2] to convert a character from a PK font file to a bitmap. Output from BM2PK typically looks like this:

```
character : 64 (@)
   height : 29
    width : 34
     xoff : -4
     yoff : 28

  ............********............
  .........***........***.........
  ........**..............**.......
  ......**..................**.....
  .....**....................**....
  ....**......................**...
  ...**........******..........**..
  ..**.......****.....**.........**.
  ..**......***........**.......**..
  .**......***..........*****....**.
  .**.....****..........****.....**.
  **......***...........****.....**
  **.....****...........****.....**
  **.....****...........****.....**
  **.....****...........****.....**
  **.....****...........****.....**
  **.....****...........****.....**
  **......***...........****.....**
  .**.....****..........****.....*.
  .**......***..........*****....**.
  ..**......***........******....*..
  ..**.......****.....**..***...**..
  ...**.........******.....*****....
  ....**.............................
  .....**...........................
  ......**.........................
  ........**.....................****
  ..........***............******...
  ............************.........
```

It can also display bitmaps in hexadecimal format:

```
character : 64 (@)
   height : 29
    width : 34
     xoff : -4
     yoff : 28

  0007f80000
  0038070000
  00c000c000
  0300003000
  0600001800
  0c00000c00
  1803f00600
  301e0c0300
```

[2] Available at CTAN: `fonts/utilities/ps2pk/ps2pk14x/msdos/emx/pk2bm.exe` or `fonts/utilities/ps2pk/ps2pk14x/msdos/djgpp/pk2bm.exe`.

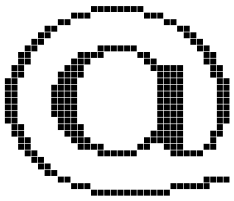
```
  3038060300
  607003e180
  60f001e180
  c0e001e0c0
  c1e001e0c0
  c1e001e0c0
  c1e001e0c0
  c1e001e0c0
  c1e001e0c0
  c0e001e0c0
  60f001e080
  607003e180
  303807e100
  301e0ce300
  1803f07c00
  0c00000000
  0600000000
  0300000000
  00c00003c0
  0038007e00
  0007ff8000
```

Even C-code can be generated:

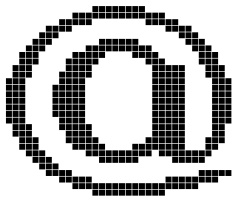
```
#define cmr5_@_width 34
#define cmr5_@_height 29
#define cmr5_@_xoff 0
#define cmr5_@_yoff 0
static char cmr5_@_bits[] = {
   0x00,  0xe0,  0x1f,  0x00,  0x00,
   0x00,  0x1c,  0xe0,  0x00,  0x00,
   0x00,  0x03,  0x00,  0x03,  0x00,
   0xc0,  0x00,  0x00,  0x0c,  0x00,
   0x60,  0x00,  0x00,  0x18,  0x00,
   0x30,  0x00,  0x00,  0x30,  0x00,
   0x18,  0xc0,  0x0f,  0x60,  0x00,
   0x0c,  0x78,  0x30,  0xc0,  0x00,
   0x0c,  0x1c,  0x60,  0xc0,  0x00,
   0x06,  0x0e,  0xc0,  0x87,  0x01,
   0x06,  0x0f,  0x80,  0x87,  0x01,
   0x03,  0x07,  0x80,  0x07,  0x03,
   0x83,  0x07,  0x80,  0x07,  0x03,
   0x83,  0x07,  0x80,  0x07,  0x03,
   0x83,  0x07,  0x80,  0x07,  0x03,
   0x83,  0x07,  0x80,  0x07,  0x03,
   0x83,  0x07,  0x80,  0x07,  0x03,
   0x03,  0x07,  0x80,  0x07,  0x03,
   0x06,  0x0f,  0x80,  0x07,  0x01,
   0x06,  0x0e,  0xc0,  0x87,  0x01,
   0x0c,  0x1c,  0xe0,  0x87,  0x00,
   0x0c,  0x78,  0x30,  0xc7,  0x00,
   0x18,  0xc0,  0x0f,  0x3e,  0x00,
   0x30,  0x00,  0x00,  0x00,  0x00,
   0x60,  0x00,  0x00,  0x00,  0x00,
   0xc0,  0x00,  0x00,  0x00,  0x00,
   0x00,  0x03,  0x00,  0xc0,  0x03,
   0x00,  0x1c,  0x00,  0x7e,  0x00,
   0x00,  0xe0,  0xff,  0x01,  0x00, };
```

Another nice program to examine or even hand-edit PK font files is the MS-DOS (or OS/2) program (PKEDITPM).[3]

A good candidate for displaying the difference between a font generated for Laserjet or Docutech is cmr5. The ampersand character will be shown because of its delicate thin lining.

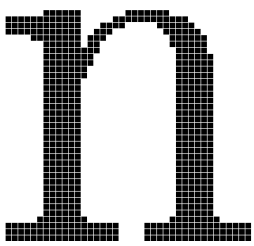


Laserjet cmr5

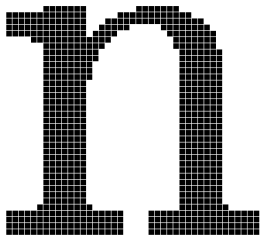


Docutech cmr5

The differences are obvious: The Laserjet font is much thinner. This was an extreme case where lines were as thin as one pixel. Naturally the differences become less and less visible with higher font sizes. At 10 pt there is still a noticeable difference as shown in the letter n:



Laserjet cmr10



Docutech cmr10

The stems of the Laserjet font are one pixel thinner, just like the serifs. At size higher than 15 pt the differences become insignificant.

4 Conclusion

It is obvious that `mode_defs` are essential for high quality output using fonts written in Metafont. If you have searched Karl Berry's `mode_def` collection and found that your device is not listed, the best thing you can do is try to find a device with similar resolution, and see if that suits. Otherwise you will have to fiddle yourself the variables mentioned. Ideally you would try normal, bold an italic versions, at sizes 5 pt, 10 pt and 15 pt.

If you make a new `font_def`, please send it to Karl Berry.[4] Please mention what fonts at what sizes you tested it on. This will help other people wondering where particular values came from.

[3] Available at CTAN: `systems/msdos/emtex/disk5/pkedit.zip` or `systems/msdos/emtex/betatest/pkeditpm.zip`.

[4] E-mail address: `karl@cs.umb.edu`.