TEXniques
Toolbox

Maarten Gelderman

abstract

This toolbox contains some varia. First I discuss some reactions on remarks I made in an earlier toolbox. Next, Hans Hagens texexec is used in the following section to create eps and pdf files from metapost source. Files created this way are often more usefull than the eps files metapost itself creates. How to prepare a single source file for usage with both traditional TEX and pdfTEX is discussed next. I also show how easy it is to set up a font different from Computer Modern for typesetting simple mathematics, pay some attention to a failed attempt to install a TrueType font and present a small PostScript header file that can be used to produce watermarks.

keywords

make, texexec, pdf, eps, math fonts

The previous toolbox

Two persons gave comments and corrections related to the previous toolbox. Jules van Weerden informed me that my description of make was flawed. In the previous toolbox I indicated that if multiple makefiles exist, make will first try to use the one capitalized as Makefile. Of course, it is never prudent to depend on capilization for such decisions, however, in this case it may even turn out to be lethal: the behaviour described by me is not standard make behavior, so any one using another make than that of the GNU variety may get unexpected results.

Wybo Dekker gave an improvement of my unreadable (emacs) regular expression. I presented the next regexp: `\\\(['\|\^\|'\|"]\)\([^\{]\)` → `\\\1{\2}`, Wybo mailed me the shorter—but fortunately equally unreadable—alternative: `\(\\['^'"]\)\([^\{]\)` → `\1{\2}`. But of course, real men, like Wybo, do not use emacs, but Perl. For those interested in carrying out the same substitution using this program, the regexp to use is: `s/(\\['^'"])([^{])/$1\{$2}/g`.

Making makefiles superfluous

However, since Hans Hagens texexec is available makefiles are superfluous to the average TEX-user. This Perl-script, which I will not discuss here, just read Hans' own description of it, takes care of all stages of compilation. It can even be used for easy previewing of metapost-files. By issuing the following command `texexec -output=pdftex -figures=c -tex=cont-en -result=plaatjes plaatjes.[0-9]*`, a PDF-file will be created which contains the pictures generated from the metapost-file plaatjes.mp.[1] The ability to generate pdf-files from metapost generated pictures is not only handy for previewing. It also is a reasonable foolproof way to solve font conflicts. Metapost does not include the fonts it uses in the eps-files it generates. It depends on a postprocessor (most often dvips or pdfTEX) for the final inclusion of these fonts. Although this approach may have some advantage in terms of efficiency, conflicts involving fontnames all too often occur. The pdf-file generated in the way described above is really self-contained and consequently solves such ambiguities. If desired, you can use GhostScript (pdf2ps) to convert the pdf file to (encapsulated) postscript.

Of course, I find typing the whole texexec command too much work, so I made yet another Makefile, to simplify life:

```
plaatjes.1: plaatjes.mp
        export TEX=latex;mpost factor
        texexec --output=pdftex --figures=c\
        --tex=cont-en --result=plaatjes    \
        plaatjes.[0-9]*
```

Using the same document with ordinary TEX and pdfTEX

Most readers of this journal will know it by now. TEX can be used to make (nearly) perfect PDF-documents. When you use the pdfTEX executable instead of the ordinary TEX, PDF-files instead of DVI-files are easily generated by executing the command `\pdfoutput=1` somewhere at the start of your document. LATEX users can obtain more advanced results by using packages like hyperref (the `\pdfoutput=1` is superfluous when this package is used). If you for instance include `\usepackage [pdftex, bookmarks=true, bookmarksopen=true] {hyperref}`

1. The `[0-9]`-convention used in the command requires you to use a reasonably modern shell like bash. Windows-users will probably have to enter the names of the metapost-generated files by hand.

in your preamble, bookmarks will be generated automatically and will be visible when the document is opened in Acrobat Reader.

However, this approach introduces one problem: every now and then you will want to use your ordinary TEX-executable to process this document. If your document includes the commands mentioned above, TEX will issue an error-message: either these commands themselves are undefined, or they make use of undefined commands. The solution to this problem capitalizes on this same feature: if \pdfoutput is defined, we know we are using pdfTEX and simply assume we want to generate a PDF-file. If it is undefined, apparently a traditional TEX is being used and we do not want to issue these PDF related commands. The implementation is rather simple. First we define a new boolean variable:

```
\newif\ifpdf
```

Next we check whether \pdfoutput is defined. If this is not the case we do not set \pdfoutput to 1 and we assign a false value to our boolean. In other cases we assign a value to \pdfoutput and make our boolean true.[2]

```
\ifx\pdfoutput\undefined
  \pdffalse
\else
  \pdfoutput=1
  \pdftrue
\fi

\ifpdf
  \usepackage[pdftex]{graphicx}
  \graphicspath{{pdf/}{png/}}
\else
  \usepackage[dvips]{graphicx}
  \graphicspath{{eps/}}
\fi
```

In the code presented above the \ifpdf boolean is also used to solve another problem: in ordinary TEX I traditionally use .eps-files. PdfTEX is only able to process .eps-files generated by metapost. Fortunately pdfTEX *is* able to deal with two other file formats: PNG (portable network graphics, for bitmap files) and PDF. Assuming that each picture used in our document is present in .eps-format in the eps-directories and in either .png or .pdf format in the png and pdf directory respectively, the above code assures that depending on the executable used, the right graphics-file will be selected.

Poor-man's-math

By now, most TEX users are aware of the fact that we do not need to stick to Computer Modern when preparing our documents. As long as one does not want to typeset mathematics, Type I fonts are easily integrated. If one wants to do advanced mathematical typesetting, using Computern Modern remains the only option for most users. The reason traditionally given is simple: a whole bestiary of mathematical symbols is available in Computer Modern. This same bestiary—and more often only a subset of it—can only be found in a number of extensions to commercial fonts. Such extensions are available for Times, Helvetica, Courier and Lucida. The user who does not want to buy these fonts, is forced to stick to Computer Modern, or to combine his/her Type I font with Computer Modern or Euler (a free mathematics only font).[3]

So far for theory, now to practice. I do not know how your documents look, but the math in a lot of my documents is hardly more complex than $y = a + b_1x_1 + b_2x_2$. Why would I need complex math symbols for such documents? Provided that I can live with a limited set of symbols and would be willing to sacrifice some of the niceties of math spacing that TEX normally tries to provide, shouldn't it be possible to find some way to set such simple math out of a arbitrary Type-I font? After some experiments with commands discussed in *The LaTeX companion* I learned that, although TEX complains bitterly about such indecent treatment, the next set of commands produces reasonably acceptable results.

```
\documentclass{article}
\renewcommand{\rmdefault}{padj}
\DeclareMathVersion{normal}
\DeclareMathVersion{bold}
\DeclareSymbolFont{operators}{OT1}{pad}{m}{n}
\SetSymbolFont{operators}{bold}{OT1}{pad}{b}{n}
\DeclareSymbolFont{letters}{OT1}{padj}{m}{it}
\SetSymbolFont{letters}{bold}{OT1}{padj}{b}{it}

\begin{document}\thispagestyle{empty}
$$\sum_{a=1}^{\infty}\frac{a_1+b^2}{3+c^{i_3}}$$
\end{document}
```

I do not claim to know exactly what I am doing, so I can only give a small elaboration on the meaning of these commands. The main thing to notice is the padj and pad stuff, which stand for Adobe Garamond with and without oldstyle numerals respectively. If you replace both pad and padj by e.g. phv your formulas will be set in Helvetiva, if you replace both by pcr, Courier will be used instead etc.

2. Some LaTeX-classes/packages define \pdfoutput, consequently it is essential to set the boolean to false or true before any classes/packages are loaded.

3. For Times, an alternative is available: the package mathptm does its uttermost best to imitate a Times with additional math symbols using a combination of Times, the PostScript Symbol font and Computer Modern.

$$\sum_{a=1}^{\infty} \frac{a_1 + b^2}{3 + c^{i_3}} \qquad \sum_{a=1}^{\infty} \frac{a_1 + b^2}{3 + c^{i_3}} \qquad \sum_{a=1}^{\infty} \frac{a_1 + b^2}{3 + c^{i_3}}$$

Figure 1. Examples of poor-man's-math as described in this article, using Adobe Garamond, Helvetica and Courier respectively.

A disillusion

Being able to use Type I fonts of course is nice. But of some fonts I happen to own just a TrueType version. Conceptually using these fonts does not seem to be too difficult. A first possible solution is conversion from TrueType to Type 1. However, this will result in lower quality font. A second approach seems more worthwhile: more recent PostScript interpreters can deal with so-called Type 42 fonts. Type 42 fonts can be used to 'embed' TrueType fonts in a PostScript font. This did seem to be the way to go. However, although I got a lot of help from the people on TEX-NL, I was not able to do the conversion. The AFM-files generated by the conversion program do not contain information on the x-height and the cap-height of the font and TEX can't do without. I hope I can report some progress in a next edition of MAPS.

Watermarks

Some people apparently are not able to live without them anymore: watermarks. On every sheet of paper that leaves their hands they print their name or some other superfluous remark in the background. I hate those watermarks, but nevertheless one sometimes does need them. Recently I had to present the results of a research project to the scientific council of a research institute. I was warned beforehand that it would be prudent to take some measures that would ensure that my instrument could not be copied without my name on it. 'You never know where those copies end up, and people might just not remember how they obtained them.' One does not ignore such advise. Fortunately I did remember that Siep Kroonenberg posted some information on this topic on TEX-NL, the information she provided, combined with the *PostScript Language Tutorial and Cookbook* sufficed to make the following code:

```
%!
/bop-hook {gsave
  /Helvetica-Bold findfont 35 scalefont setfont
  .7 setgray
  144 72 moveto 90 rotate
  gsave
  (If you want to use this instrument) show
  grestore
  0 -40 rmoveto
  (please contact Maarten Gelderman) show
  grestore} def
```

This little piece of PostScript code (which I saved in a file called `watermark.pro`) defines the begin-of-page (bop) hook. The graphic state of the PostScript interpreter is saved at the beginning, and restored at the end. The first thing the header file does is looking for a bold Helvetica, scaling it to 35 points and selecting it as the current font (PostScript uses Reverse Polish Notation, first the arguments are pushed on a stack and next the commands are given). Next I move to a position near the bottom of the page and select a rotation angle of 90°. The text between the parentheses is printed, I move 40 points down and print the next sentence.

Of course I need to include this header in the PostScript file I send to the printer. Although it is probably possible to do this using TEX specials, I prefer to use `dvips` to do such tasks. The following `dvips`-option suffices: `dvips -h watermark.pro`.