Margin kerning and font expansion with pdfTEX

Hàn Thé Thành

Introduction

pdfTEX has some micro-typographic extensions that are not so widely used, for the lack of documentation and quite complicated setup. In this paper I would like to describe their use in a step-by-step manner so the reader can give a try afterwards. Two extensions will be introduced: margin kerning and font expansion.

Margin kerning is the technique to move the characters slightly out to the margins of a text block in order to make the margins look straight. Without margin kerning, certain characters when ending up at the margins can cause the optical illusion that the margins look rather ragged. Margin kerning is similar to hanging punctuation, but it can also be applied to other characters as well. When used with appropriate settings, this extension can help to considerably improve appearance of a text block.

Font expansion is the technique to use a slightly wider or narrower variant of a font to make interword spacing more even. A font that can be expanded thus has some "stretchability" and "shrinkability". The potentiality to make a font wider or narrow then can be used by the line-breaking engine to choose better breakpoints.

I will describe the use of those two extensions by examples, as I find it a very good way to explain how to use something in practice. People interested in the concepts and further details will find more in my thesis.

Margin kerning

The simplest use of margin kerning can look like:

```
\input protcode.tex
\font\f=cmr10
\rpcode\f`\-=700
\adjustprotcode\f
\f
\pdfprotrudechars=1
Some text...
```

`\rpcode` stands for "right protruding code". The first parameter is a font identifier, and the second parameter is a character code. The third parameter specifies the amount how much the character will be moved to the right margin. The above example says that if the hyphen character from font `\f` ends up at the right margin, it should be moved out to the margin by 700 thousandths of its width (i.e. 70 %).

It is conveninent to specify the protruding factor for individual characters in thousandths of character width. This is also the way how `\rpcode` was implemented in versions up to 0.14h. However, this method cannot be used for characters with zero width ("faked" characters that can be used to protrude other elements than normal characters), so in version 0.14h and later, the protruding amount is specified in thousandths of an *em* of the font. A macro called `\adjustprotcode` (defined in file `\protrude.tex`) is used here to checks whether the used version is older than 0.14h and if so it will convert the settings for versions before 0.14h (i.e. in thousandths of character width) to the corresponding settings for later versions (i.e. in thousandths of an *em*).

By default, all characters have their `\rpcode` set to zero, so no characters will be protruded unless we explicitly set `\rpcode`.

The primitive `\pdfprotrudechars` is used to control margin kerning at global level.

- ≤ 0: no margin kerning.
- 1: "level 1 margin kerning", which does not have effect to line breaking. This setting is handy if you want to keep line-breaking to be compatible with TEX.

- ≥ 2: "level 2 margin kerning", which usually gives different result of line breaking. This setting causes the line-breaking engine to take the amount needed to protrude the characters at the margins into account. As the result, interword spacing is better.

  Now we go for another example:

```
\input protcode.tex
\font\f=cmr10
\lpcode\f‘\‘=700
\rpcode\f‘\’=700
\adjustprotcode\f
\f
\pdfprotrudechars=2
Some text...
```

This example introduced another primitive, `\lpcode`. It is the counterpart of `\rpcode`, used for protruding at the left margin. The above example thus sets the left quote to be protruded at left mergin and the right quote at the right margin. We set `\pdfprotrudechars` to 2, which may lead to different line breaking.

Fortunately one does not have to set `\lpcode` and `\rpcode` for each character. A common set of protruding factors works quite reasonably for most of body fonts, so one can use them as in the following example:

```
\input protcode.tex
\font\f=cmr10
\setprotcode\f
\f
\pdfprotrudechars=2
Some text...
```

The macro `\setprotcode` is also defined in the file `protcode.tex` and it will call `\adjustprotcode` after settings the common values for protruding, that is why we do not have `\adjustprotcode` in this example.

Assigment to `\lpcode` and `\rpcode` is always global. In a LaTeX document, setting up margin kerning can look like:

```
\documentclass{report}
...
\input protcode.tex
\begin{document}
\setprotcode\font
{\it \setprotcode\font}
{\bf \setprotcode\font}
{\bf\it \setprotcode\font}
Some text...
\end{document}
```

In case the settings in `protcode.tex` do not look for a particular font, you can always change it to your taste

Font expansion

Use of font expansion is more complicated due to the need to create expanded versions of a font. This task can be done on-the-fly, however its setup is system-dependent so I will describe rather how to create these fonts manually.

Creating expanded fonts Font expansion can be used with

1. fonts based on Computer Modern fonts (i.e. Computer Modern fonts and variants for Czech, Vietnamese, etc.);
2. Type 1 fonts;
3. Multiple Master fonts with width axis.

Expanded fonts have the name from the base font, followed by the expansion amount in thousandths. For example, `cmr10` expanded by 10 thousandths (1 %) will be named `cmr10+10`. Expansion value can be negative, which stands for condensing.

Putting the created fonts into their correct location is another matter. I suggest simply putting all expanded fonts into a single directory to play with.

Computer Modern fonts Computer Modern fonts can be expanded by altering the *unit width* in the source. For example, `cmr12+10.mf` is created by copying the source of the base font (`cmr12.mf`), with a line appended after the place where the unit width is defined as:

```
u#:=20/36pt#;       % unit width
u#:=u#+10/1000u#;
```

Then the changed source is used to generate the expanded TFM as well as the bitmap font needed for rendering the output. Note that while use expanded Computer Modern fonts, the corresponding entries in used map files must be commented out, otherwise pdfTEX will treat them as Type 1 fonts.

Type 1 fonts Type 1 fonts are expanded by altering the *FontMatrix* of Type 1 fonts. pdfTEX will do that for you, the only task is to create the expanded TFM files. Suppose that we have an AFM file `putr8a.afm` (Adobe Utopia Regular). Then creating 8y-encoded TFM expanded by 10 thousandths (1 %) can looks like:

```
afm2tfm putr8a.afm -e 1.010 -T texnansi.enc putr8y+10.tfm
```

You also need an entry in the map files that looks like:

```
putr8y Utopia-Regular "TeXnANSIEncoding ReEncodeFont" <texnansi.enc <putr8a.pfb
```

Only the entry for the base font (non-expanded) is needed. The expanded versions will be embedded according to the entry for the base font.

Multiple Master fonts Only Multiple Master with width asix can be expanded. The idea is to create a new instance with the width value increased by the expansion amount. The following example shows how to create an Multiple Master instance and a variant expanded by 20 thousandths:

```
mmafm --weight=400 --optical-size=12 --width=535 --output pmnr8a12.afm MinionMM.afm

mmafm --weight=400 --optical-size=12 --width=545.7 --output pmnr8a12+20.afm MinionMM.afm
```

The magic number 545.7 comes from the expression $535 \times (1 + \frac{20}{1000})$, which means that we increase the width value of the base instance by 20 thousandths.

The PFB font files are created in a similar way. Afterwards the AFM can be converted to TFM using afm2tfm:

```
afm2tfm pmnr8a12.afm -T texnansi.enc pmnr8y12.tfm

afm2tfm pmnr8a12+20.afm -T texnansi.enc pmnr8y12+20.tfm
```

Similar to Type 1 fonts, only the entry for the base font is needed in map files:

```
pmnr8y12 MinionMM_400_535_12_ <texnansi.enc <pmnr8a12.pfb
```

Using font expansion Suppose that given a font, we know how to create expanded versions of that font. Now let us try an example:

```
\font\f=cmr10
\pdffontexpand \f 20 10 5 1000
\efcode\f`\e=1000
\f
\pdfadjustspacing=1
Some text...
```

The primitive `\pdffontexpand` says that font `\f` can be expanded up to 20 thousandths, condensed to 10 thousandths by step 5 thousandths. This means that only variants whose expansion amount is a multiple of 5 are needed. In our example, the following variants are needed (they must be created before running the example): `cmr10+20`, `cmr10+15`, `cmr10+10`, `cmr10+5`, `cmr10-5`, `cmr10-10`.

The last parameter is so-called font expansion factor, and 1000 is the recommended value for most cases. More details in my thesis.

The primitive `\efcode` (character expansion factor) has similar syntax to `\lpcode` and `\rpcode`. The third parameter says how much the character 'e' is allowed to be expanded in thousands, in this case fully. This parameter can be used to restrict expansion of certain characters that are more sensitive to expansion. The default value of `\efcode` is zero, thus no expansion is allowed unless we explicitly set `\efcode`.

`\pdfadjustspacing` is also similar to `\pdfprotrudechars`, i.e. is used to control font expansion at global level: 0: no expansion; 1: expansion with backward compatibility (unchanged line breaking); and ≥ 2: expansion that may change line breaking.

A macro called `\resetefcode` available in file `efcode.tex` can be used to reset all expansion factors to 1000.

So we can try to put margin kerning and font expansion together:

```
\input protcode.tex
\input efcode.tex
\font\f=cmr10
\setprotcode\f
\reasetefcode\f
\pdffontexpand\f 20 20 5 1000
\pdfadjustspacing=2
\pdfprotrudechars=2
\f
Some text...
```

or in a LaTeX document:

```
\documentclass{report}
...
\input protcode.tex
\input efcode.tex
\def\setupfont#1{
    \setprotcode#1
    \resetefcode#1
    \pdffontexpand#1 20 20 5 1000
}
\begin{document}
\setupfont\font
{\it \setupfont\font}
{\bf \setupfont\font}
{\bf\it \setupfont\font}
Some text...
\end{document}
```


Conclusions

Margin kerning is a simple extension but quite effective. Its setup and use is very easy, while the gain is considerable. I would recommend it to regular use.

Font expansion must be used with care, as use with too tolerant expansion limits can be dangerous. According to practical experiments, the limit is $\pm 2\,\%$.

References

1. My thesis (PDF version): http://www.fi.muni.cz/~thanh/download/thesis.pdf
2. Various macros for margin kerning and font expansion with pdfTeX: ftp://ftp.cstug.cz/pub/tex/local/cstug/thanh/pdftex/ext/