Font-specific issues in pdfTEX

Hàn Thế Thành
River Valley Technologies
`http://river-valley.com`

Abstract

In this paper I try to give a summary of some font-related topics in pdfTEX. Some of them are already described in the pdfTEX manual, such as font expansion and margin kerning, some have been mentioned only in various places like relevant mailing lists, `README` or example files coming with patches, and email exchanged between people interested in a particular topic. This article attempts to put everything into one place, hoping to make it easier to follow.

1 Introduction

A large part of the pdfTEX extensions is related to font handling. Having an overview of all those font-related issues is not always easy, since the pdfTEX manual is a somewhat dry thing to read as a whole. Apart from that, there are also things that are not described in the manual yet. In this paper I will try to give an overview of font extensions in pdfTEX. Instead of listing all relevant primitives with their description, I will write on particular topics that I consider interesting to mention here.

2 Font expansion and margin kerning

Since these features have been mentioned many times, I simply skip their description here and only give the references to relevant sources: [1], [2].

LATEX users who want to try out these features should start with the LATEX `microtype` package. ConTEXt users should consult the ConTEXt manual first.

Primitives relevant to font expansion:

- `\pdfadjustspacing`,
- `\pdffontexpand`,
- `\efcode`;

Primitives relevant to margin kerning:

- `\pdfprotrudechars`,
- `\lpcode`,
- `\rpcode`.

All those primitives are described very well in the pdfTEX manual.

3 Additional micro-typographic features

Apart from the above features, pdfTEX has some additional support for finer control on interword spacing and kerning. The `microtype` package provides an easy access to those features. Furthermore, the `microtype` manual [1] has a very good introduction to these additional features, which I copy here for convenience (slightly edited):

> ... On the contrary, pdfTEX was extended with even more features: version 1.30 introduced the possibility to *disable all ligatures*, version 1.40 a robust *letterspacing* command, the *adjustment of interword spacing* and the possibility to specify *additional character kerning*.
>
> Robust and hyphenatable *letterspacing* (tracking) has always been extremely difficult to achieve in TEX. Although the `soul` package undertook great efforts in making this possible, it could still fail in certain circumstances; even to adjust the tracking of a font throughout the document remained impossible. Employing pdfTEX's new extension, this no longer poses a problem. The `microtype` package provides the possibility to change the tracking of customisable sets of fonts, e. g. small capitals. It also introduces two new commands `\textls` and `\lsstyle` for ad-hoc *letterspacing*, which can be used like the normal text commands.
>
> *Adjustment of interword spacing* is based on the idea that in order to achieve a uniform greyness of the text, the space between words should also depend on the surrounding characters. For example, if a words ends with an 'r', the following space should be a tiny bit smaller than that following, say, an 'm'. You can think of this concept as an extension to TEX's 'space factors'. However, while space factors will influence all three parameters of interword space (or glue) by the same amount — the kerning, the maximum amount that the space may be stretched and the maximum amount that it may be shrunk — pdfTEX provides the possibility to modify these parameters independently from one another. Furthermore, the values may be set differently for each font. And, probably most importantly, the parameters may not only be increased but also

decreased. This feature may enhance the appearance of paragraphs even more. Emphasis in the last sentence is on the word 'may': this extension is still highly experimental — in particular, only ending characters will currently have an influence on the interword space. Also, the settings that are shipped with `microtype` are but a first approximation, and I would welcome corrections and improvements very much. I suggest reading the reasoning behind the settings in section 15.8.

Setting *additional kerning* for characters of a font is especially useful for languages whose typographical tradition requires certain characters to be separated by a space. For example, it is customary in French typography to add a small space before question mark, exclamation mark and semi-colon, and a bigger space before the colon and the guillemets. Until now, this could only be achieved by making these characters active (for example by the `babel` package), which may not always be a robust solution. In contrast to the standard kerning that is built into the fonts (which will of course apply as usual), this additional kerning is based on single characters, not on character pairs.

The possibility, finally, to *disable all ligatures* of a font may be useful for typewriter fonts.

The `microtype` package provides an interface to all these micro-typographic extensions. All micro-typographic aspects may be customised to your taste and needs in a straight-forward manner.

3.1 Letterspacing

We all probably know what letterspacing is and related problems when using it with TEX. The robust and reliable way to letterspace a font in TEX is to create a virtual font which inserts a fixed kern around each character. The famous `fontinst` package can be used to do this, however, it must be done for each font we want to letterspace. Furthermore, `fontinst` is not a tool for everybody.

There have been several attempts in pdfTEX to solve this problem: one idea was to insert an explicit kern before and after each character, when the character is typeset by TEX, roughly like typing i. e. `\kern.1em X\kern.1em` for each character '`X`'. This approach had several problems; the most serious one is that it disabled hyphenation. Another attempt used implicit kerns instead of explicit ones; while this method allowed hyphenation, it caused other problems. In the end, a method that generates a virtual font on-the-fly was implemented. It works very much like the way one uses `fontinst` to letterspace a font, but it is done automatically in pdfTEX, at run time. A minimal example looks like this:

```
\font\f=cmr10
\letterspacefont\fx=\f 100
\fx <letterspaced text>
```

The above commands create a letterspaced version of `\f` (which is `cmr10`) as a virtual font. This virtual font is accessible to the user via the control sequence `\fx`. Each character from `\fx` is typeset using its counterpart from `\f`, plus a kern of `50*quad(\f)/1000` at each side.

There are some issues with compensating for the kern at the beginning/end of letterspaced text. Since the kern amount is known, it is possible to compensate that kern manually if needed, for example when using `\fx` inside an `hbox`.

In a multiple-line paragraph, one can compensate for the kern at the margin using margin kerning like follows:

```
\pdfprotrudechars=2
\newcount\n
\n=0
\loop
    \lpcode\fx\n 50
    \rpcode\fx\n 50
    \advance\n 1
\ifnum\n<256\repeat
```

This is still not perfect, since you lose the effect of margin kerning (now all marginal kerns are the same, so the margins are aligned mechanically as in the case without margin kerning). If you want to have both letterspacing and margin kerning, you need to compensate for the margin kern as follows, given that you have set up margin kerning for `\f` already:

```
\newcount\n
\newcount\m
\n=0
\loop
    \m=\lpcode\f\n
    \advance\m 50
    \lpcode\fx\n \m
    %
    \m=\rpcode\f\n
    \advance\m 50
    \rpcode\fx\n \m
    %
    \advance\n 1
\ifnum\n<256\repeat
```

The current version of pdfTEX (1.40.3) still has a problem when using letterspacing with font expansion. This problem will be fixed soon (not hard to do).

Relevant primitive: `\letterspacefont`

3.2 Adjustment of interword spacing

TeX treats all interword spaces from input text as glue items, while sometimes people need finer control over interword spaces, since this is one of the most important elements in paragraph building. Instead of describing the topic using my own words, I find it more convenient to quote the conversation via email between me and people interested in this topic (Frank Mittelbach and Ulrich Dirr).

Frank: what TeX is missing is a way to kern with the white space between words. The Adobe fonts and others might have such kerns but they have been written for software which does use "space chars" not glue.

Thành: I also would like to see the space character to be handled in a different way than it is now. Turning it into glue is probably not the best solution. It disallows fine adjustment of interword spaces to make them optically even rather than mechanically. Kerning with respect to the space can be used to improve this, but it is certainly not sufficient. Moreover, the boundary char mechanism has its limitations.

Frank: It would be a very radical step if one would introduce real space characters which (perhaps) just before typesetting are replaced by glue not early on. But again, we have now stayed and worked with TeX as it is for 20 years and if certain areas and their underlying ideas prove to be insufficient, why not experiment with alternatives?

However one other comment, if you look at what some typographers write about making the white space visually even, it make me wonder if you really can do much good about having "kerns" if you then end up with

```
<last char><kern><interword glue><kern>
<first char next word>
```

i. e. with the middle part stretching or shrinking at a constant rate, or if you really need `<glue>` adjustments here.

At least this is what some typographers claim: that if you need to shrink the interword glue that this should not be a constant factor as done with TeX but rather differing depending on the letter shapes at each side of the interword space.

Ulrich: I have had a short conversation with Frank (Mittelbach) about an extension/improvement of the paragraph building algorithm. First I thought it would be maybe possible with the help of `\sfcode` or `\spaceskip` etc. but this will not really work.

The idea is — analogous to the tables for expansion and protrusion — to have tables for optical reduction/expansion of spaces in dependence of the actual character so that the distance between words is optically equal.

When reducing distances the (weighting) order is:

- after commas
- in front of capitals which optically have more room on their left side, e.g., 'A', 'J', 'T', 'V', 'W', and 'Y'
- in front of capitals which have circle/oval shapes on their left side, e.g., 'C', 'G', 'O', and 'Q'
- after 'r' (because of the bigger optical room on the righthand side)
- before or after lowercase characters with ascenders
- before or after lowercase characters with x-height plus descender with additional optical space, e.g., 'v', or 'w'
- before or after lowercase characters with x-height plus descender without additional optical space
- after colon and semicolon
- after punctuation which ends a sentence, e.g. period, exclamation mark, question mark

The order has to be reversed when enlarging is needed.

Note: The principle of how this works can be seen in figure 1, where the numbers indicate the preference/order of each interword space when it needs to be stretched/shrunk.

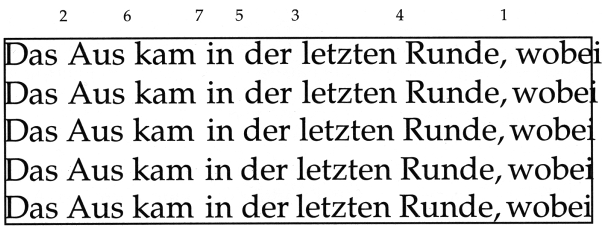
2 6 7 5 3 4 1
Das Aus kam in der letzten Runde, wobei
Das Aus kam in der letzten Runde, wobei
Das Aus kam in der letzten Runde, wobei
Das Aus kam in der letzten Runde, wobei
Das Aus kam in der letzten Runde, wobei

Figure 1: Interword spaces should be changed with respect to the adjacent characters.

Thành: I remember discussing this issue long time ago, when Frank also got involved. The problem with interword spaces in TeX is that Knuth decided to treat interword spaces like glue, while IMHO it needs special care because this is one of the most important factors in building a paragraph and hence we need a way to distinguish it from other glues.

From the experience with margin kerning, I think we should better make some small steps to see whether it makes sense, rather than start heavily changing the paragraph building engine.

Thành: I implemented an approach to allow more control on interword space as we discussed before. Sorry for the long delay.

I introduced three primitives:

`\knbscode` — kern before space code,

`\stbscode` — stretch before space code,
`\shbscode` — shrink before space code.

These primitives have the same syntax as `\rpcode` etc., i. e.

```
\knsbcode\font`\.=200
```

means that if a period sits before an interword space (glue), then the interword glue will be increased by an amount of `1em*200/1000`, i. e. the value is given in thousandths of an em as in the case of `\rpcode` etc. `\stbscode` and `\shbscode` are similar but adjust the stretch/shrink components of the interword glue.

Adjusting the interword glue only has effect when the space factor of the previous char is different from 1000.

For now I leave out ligatures and the case *after* the interword glue.

A minimal test file might look as follows:

```
\font\f=cmr10
\pdfadjustinterwordglue=1
\sfcode`\.=1000
\knbscode\f`\.=100
\shbscode\f`\.=200
\stbscode\f`\.=300
\f <text>
\bye
```

The above example would adjust every interword glue following a period by adding an amount of `.1em`, `.2em` and `.3em` to the glue width resp. shrink resp. stretch component. The primitive `\pdfadjustinterwordglue` is to switch the feature on/off at the global level, and setting `\sfcode` to 1000 is required to activate this feature (so they do not interfere with each other).

These features are available in pdfTEX since version 1.40, and are also supported by the `microtype` package. However, the predefined values are not yet optimal — probably more experimenting is needed to tune the parameters to get a good result. Please refer to the `microtype` documentation for further details.

There is no support to adjust the interword space with respect to the next character. The main reason is that it is not easy to do in current pdfTEX code. Hopefully when LuaTEX is ready, this can be changed.

Relevant primitives:

- `\knbscode` — "kern before space" code,
- `\stbscode` — "stretch before space" code,
- `\shbscode` — "shrink before space" code,
- `\pdfadjustinterwordglue` — turns on/off the feature.

4 Additional kerning

This is a feature that allows inserting a kern before or after a certain character from a font. A minimal example looks like this:

```
\font\f=cmr10
\pdfprependkern=1
\knbccode\f`\:=500
\f <text>
```

The above example prepends a kern of `.5em` before each colon. It is also possible to append a kern after a character:

```
\font\f=cmr10
\pdfappendkern=1
\knbccode\f`\;=100
\f <text>
```

These features are also supported by `microtype` already. However these new features are not flexible enough to get rid of the need to have active characters in babel/french, as shown in this email by Daniel Flipo:

> I have heard about new kerning facilities coming with pdfTEX 1.40 and started playing with them (through the microtype interface, latest version 1.37 2006-09-09 with `\betatrue`). I would love to get rid of the four active characters (:;!?) in babel/frenchb.
>
> Unfortunately, after discussing with Robert (in copy), it appears that these new kerning facilities do not quite fulfil what would be needed for French. I'd like to make a summary of the required specifications in case you can think of a possible solution for future developments of pdfTEX.
>
> 1) People who type correctly in French, are used to type a (normal) space before ';:!?'. pdfTEX 1.40 can add a kern before them, but cannot do an `\unskip` to remove the typed space. It is hopeless to try to convince French writers to change their habits and refrain from entering a space before ';:!?' `;-)`
>
> frenchb currently handles the four (active) punctuation chars ;:!? in two different ways:
>
> — with the option `\NoAutoSpaceBeforeFDP`, frenchb *replaces* the normal space with an unbreakable one of the correct width, *if and only if* a space (normal or '~') is present before ';:!?'. If no space is typed, frenchb does nothing and lets the punctuation mark stick to the preceding word. This avoids a spurious space in URLs (`http://...`), Windows paths (`C:/path`), etc.
>
> — with the option `\AutoSpaceBeforeFDP` (the default), you can carelessly type any of `bonjour!`, `bonjour !` or even `bonjour~!`; frenchb will always output it correctly — but then you cannot complain if you get a spurious space in URLs.

2) Another (minor) issue occurs with ':'. Again, there are currently two different options in frenchb:

— Most people agree with our « Imprimerie nationale » that ':' should be surrounded by two spaces of the same length, the first one being unbreakable, while the other three (;!?) get a thin space (kern in TEX) before and a normal space (glue) after. That's what frenchb does by default.

— Some typographers argue that ':' should be treated like the other three, so an option is provided in frenchb to satisfy them.

AFAIK pdfTEX 1.40 can add a kern before a character but not a glue, so the spaces around ':' might look asymmetrical in the first case if TEX stretches the second one.

3) Guillemets are less problematic because they are currently entered with commands (`\og` and `\cg`), not as characters. Spaces after the opening '«' and before the closing '»' should be unbreakable but stretchable (currently these are `.8\fontdimen2 plus .3\fontdimen3 minus .8\fontdimen4`). Moreover, a kern after '«' breaks hyphenation of the following word as Robert already pointed out on the pdfTEX bug list.

So the current situation still needs improvement, which is likely left to LuaTEX.

Relevant primitives:

- `\knbccode` — "kern before character" code,
- `\pdfprependkern` — toggle prepending of kerns,
- `\knaccode` — "kern after character" code,
- `\pdfappendkern` — toggle appending of kerns.

5 Support for ToUnicode map

ToUnicode map is a concept in the PDF specification that allows mapping from character codes in a font to corresponding Unicode numbers. The purpose is to allow PDF browsers to perform properly operations related to text contents like search, cut and paste. Support for this feature was added mainly to make PDF files produced with the MinionPro package [3] searchable. If you are having trouble with PDF produced by pdfTEX not being searchable with some fonts, give this feature a try (N. B.: this feature only works for Type 1 fonts). A minimal example:

```
\input glyphtounicode.tex
\pdfgentounicode=1
<text>
```

If `glyphtounicode.tex` is not available in your TEX distribution, it can be downloaded from [4]. This file covers most common cases. If you want to add your own entries, here is an example how it can be done:

Suppose that you have a font which has another variant of letter 'A', named e. g. 'myCoolA', and you wish that glyph to be found when you search for 'A'. Then you add to `glyphtounicode.tex` (or insert somewhere in your TEX file) a line saying:

```
\pdfglyphtounicode{myCoolA}{0041}
```

which means that the glyph with name 'myCoolA' has the corresponding Unicode number `0041` (which is the same as for the normal 'A'). This would make your 'myCoolA' behave like 'A' regarding operations like search, cut and paste.

6 Support for subfont

TEX was designed to work with 8-bit fonts only. CJK languages however use fonts with thousands of glyphs. To make those fonts work with TEX, a trick called 'subfont' was developed by Werner Lemberg for his CJK package. The subfont technique splits a huge font into smaller fonts, each of them containing up to 256 characters.

Explaining the subfont mechanism is out of scope for this paper, so I simply refer people with further interest in this topic to [5]. Here I try to give a simple example.

Suppose we want to use the Bitstream Cyberbit Unicode font. This font has about 30 000 glyphs and covers many languages. The fontfile is called `cyberbit.ttf`. We want to use this font to typeset CJK languages written in UTF-8 encoding.

The first step is to generate the TFM subfonts:

```
ttf2tfm cyberbit.ttf cyberb@Unicode.sfd@
```

The `ttf2tfm` program is part of the FreeType 1 bundle; it comes with all major TEX distributions like TEX Live or MiKTEX. `Unicode.sfd` is a *subfont definition* that comes with the CJK package. It is basically a text file containing instructions how to split a large font into subfonts. The above command will produce a bunch of TFM files with names in form `cyberbxx.tfm`, where *xx* are two lowercase hexadecimal digits. Copy the TFMs to a location where pdfTEX can find them.

The next step is to tell pdfTEX about the subfonts by adding to your TEX file a line saying:

```
\pdfmapline{+cyberb@Unicode@ <cyberbit.ttf}
```

The effect of the above command is that pdfTEX will be able to pick up the right glyphs for those TFMs from `cyberbit.ttf` and embed them as subsetted TrueType fonts in the PDF output. So it is no longer necessary to convert `cyberbit.ttf` to Type 1 subfonts and embed them, or to run `ttf2pk`. A complete minimal example:

```
\documentclass{article}
\usepackage{CJK}
\pdfmapline{+cyberb@Unicode@ <cyberbit.ttf}
```

```
\DeclareFontFamily{C70}{cyberbit}
    {\hyphenchar\font -1}
\DeclareFontShape{C70}{cyberbit}{m}{n}
    {<-> CJK * cyberb}{}

\begin{document}
\begin{CJK}{UTF8}{cyberbit}
\CJKnospace
<some CJK text in UTF-8 encoding>
\end{CJK}
\end{document}
```

There are many details that are not mentioned here, however the above example should give a good feeling about what can be done.

7 runpdftex — a wrapper to run pdfTEX

This section is not about a font-related topic in pdfTEX, but it is also relevant to pdfTEX so I take this opportunity to mention it.

runpdftex is a wrapper that allows applications to call pdfTEX via a well-defined API in C. The main intention is to hide TEX-specific details from the application that calls pdfTEX to generate a PDF file. A developer can call pdfTEX to convert a TEX file to PDF using library calls that are robust, easy to understand and use, and take care of error handling. This way, a Web developer who is not a TEX expert can set up a system that uses pdfTEX to create PDF output on-demand, for example some report, form, timetable or bank statement. The Web developer can ask or hire a TEX guru to write a TEX file or template that produces the required output.

This is still an experimental project, however I hope it will make pdfTEX more friendly to Web developers who need to create PDF on-demand but are too scared by TEX's complexity to give it a try. The API is available only for C at the moment, but support for other languages will be added. This wrapper has been designed with pdfTEX in mind, but can be used to run other TEX variants as well, for example LuaTEX when it is ready. For further information about runpdftex see [6].

References

[1] The manual of the LATEX microtype package by Robert Schlicht is available at http://ctan.org/tex-archive/macros/latex/contrib/microtype/microtype.pdf

[2] Hàn Thế Thành, *Micro-typographic extensions of pdfTEX in practice*, *TUGboat*, vol. 25 (2004), no. 1 — Proceedings of the Practical TEX 2004 Conference, pp. 35–38. (Online at http://www.tug.org/TUGboat/Articles/tb25-1/thanh.pdf)

[3] The MinionPro package containing LATEX support for Adobe MinionPro fonts is available at http://www.ctan.org/tex-archive/fonts/minionpro

[4] Definitions for ToUnicode entries can be downloaded from http://pdftex.sarovar.org/misc/glyphtounicode.zip

[5] The CJK package for LATEX is available at http://cjk.ffii.org/

[6] The runpdftex wrapper is available at http://runpdftex.sarovar.org/