Type 3 fonts and PDF search in `dvips`

Tomas Rokicki

Abstract

PDF files generated from the output of `dvips` using bitmapped fonts have not been properly searchable, indexable, or accessible. While a full solution is challenging, only minimal `dvips` changes are required to support English language text, changes that are at least two decades overdue. I will describe these changes and discuss their limitations.

1 Introduction

The Type 3 fonts generated by `dvips` for bitmapped fonts lack a reasonable encoding vector, and this prevents PDF viewers from interpreting those glyphs as text. This in turn prevents text search, copy and paste, screen readers, and search engine indexing from working correctly. Fixing this is easy, at least for English text, and comes with no significant cost.

This is not nearly a full solution to create accessible multilingual PDF documents. Support for eight-bit input encodings [2], explicit font encodings [3], and direct generation of PDF can yield better results. But if you want to use METAFONT fonts as-generated and `dvips`, this is an important change.

I describe how I generated reasonable encoding vectors for common METAFONT fonts, how `dvips` finds these encoding vectors and embeds them in the PostScript file, and how the current implementation allows for future experimentation and enhancement.

2 A little history

When `dvips` was originally written in 1986, the lone PostScript interpreter on hand was an Apple LaserWriter with 170K available memory. I treated PostScript as just a form of compression for the page bitmap, doing the bare minimum to satisfy the requirements for Level 1 Type 3 fonts. One of those requirements was to supply an `/Encoding` vector, despite the fact that at the time, the vector was completely unnecessary in rendering the glyphs. Not considering that people might someday use that encoding vector for glyph identification, on that fateful day in 1986 I generated a semantically nonsensical but syntactically acceptable vector (`/A0`–`/H3` in base 36) for all bitmapped fonts, and this vector remains to this day, subverting any attempt to search copy, or use screen readers.

Replacing this encoding vector with something more reasonable allows PDF viewers to properly understand what characters are being rendered, at least for English-language text.

3 A sample

The following TeX file, cribbed from `testfont.tex` but using only a single font, will be used for illustration.

```
\hsize=3in \noindent
On November 14, 1885, Senator \& Mrs.~Leland
Stanford called together at their San
Francisco mansion the 24~prominent men who
had been chosen as the first trustees of The
Leland Stanford Junior University.
?`But aren't Kafka's Schlo{\ss} and {\AE}sop's
{\OE}uvres often na{\"\i}ve  vis-\`a-vis the
d{\ae}monic ph{\oe}nix's official r\^ole
in fluffy souffl\'es?
\bye
```

When you run this through TeX and `dvips` (giving the `-V1` option to enforce bitmapped and not Type 1 fonts), and then `ps2pdf`, the resulting PDF does not support text search in most PDF viewers. In Acrobat with copy and paste it almost works; the c's are dropped throughout (San Francisco becomes San Fran is o). The c's are dropped because the original `dvips` encoding uses `/CR` as the name for this character, and it is apparently interpreted as a non-marking carriage return. Ligatures also don't work. In Mac OS X Preview (the default PDF viewer for the Mac), selecting text appears to fail (it actually works, but the selection boxes are too small to see that anything has actually been selected) and no characters are recognized as alphabetic. In Chrome PDF preview, selecting text gives a random note appearance with each word separately selected by its bounding box and no alphabetic characters recognized.

Conversely, when you process the file with Type 1 fonts, all text functions perform normally, except that accented characters are detected as two separate characters (the accent and the base character). The critical difference is not Type 3 (bitmaps) versus Type 1 (outline fonts), but rather the lack of a sensible encoding vector in the Type 3 font.

4 First attempts and failure

If I manually copy the `Encoding` vector from the output of dvips using Type 1 fonts and put that in the font definition for the Type 3 fonts, the situation improves; now Adobe Acrobat properly supports text functions (including ligatures but not accented characters). The other PDF viewers now recognize alphabetic characters, but they still have a number of problems.

With Preview, if you use command-A (to select all the text) and then command-C (to copy it), and then copy the result into a text editor (or a word processing program "without formatting"), you get the following mishmash of text:

> On Novemb er 14, 1885, Senator & Mrs. Leland Stanford called mansion the 24 together at their San Francisco prominent men who had b een cho- Stanford sen as the first trustees of The Leland Junior Æsop's University. ¿But aren't Kafka's Schloß and Œuvres often na"ıve vis-`a-vis the dæmonic phœnix's official r ˆole in fluffy souffl'es?

In addition to the broken words and split accented characters, if you look carefully you will notice some surprising and substantial word reordering! What could be going on?

5 Refinements and success

All PDF viewers use some heuristics to turn a group of rendered glyphs into a text stream. The heuristics differ significantly from viewer to viewer. The most important heuristic appears to be interpreting horizontal escapement into one of three categories: kerns, word breaks, and column gutters. Preview was failing so badly because it was recognizing rivers in the paragraph as separating columns of text. To satisfy the PDF viewers I had access to, I made two additional modifications to each bitmapped font.

First, I adjusted the font coordinate system, as defined by the so-called font matrix. The default Adobe font coordinate system has 1000 units to the em, while the original `dvips` uses a coordinate system with one unit to the pixel both for the page and for the font, and doesn't use the PostScript `scalefont` primitive. But not using `scalefont` apparently makes some viewers think all the fonts are just one point high, and they use spacing heuristics appropriate for such a font. By providing a font matrix more in line with conventional fonts, and using `scalefont`, PDF viewers make better guesses about the appropriate font metrics for their heuristics.

Second, I provide a real font bounding box. The original `dvips` code gives all zeros for the font bounding box, which is specifically allowed by PostScript, but this confuses some PDF viewers. So I wrote code to calculate the actual bounding box for the font from the glyph definitions.

With these adjustments, using `dvips` with bitmapped fonts and `ps2pdf` generates PDF files that can be properly searched with most PDF viewers — at least, for English language text.

6 Other languages: No success

I would have liked things to work with other languages as well, but was not able to get it to work. Clearly the PDF viewers are recognizing characters by the glyph names, but this appears to work only with a small set of glyph names. I hoped that those listed in the official Adobe Glyph List [1] would work, but in my experiments they (for the most part) did not. I also tried Unicode code point glyph names such as `/uni1234` and `/u1234` but neither of these formats worked in the PDF viewers I tried. I also experimented with adding a `cmap` to the font, with no success, and even tried some lightly documented GhostView hacks, but was able to achieve only distressingly partial success for most non-Roman characters.

Even if the individual glyphs are recognized, problems remain with accents, and more generally, virtual fonts. With a standard seven-bit encoding, accents are generally rendered as two separate characters, where the PDF viewer expects to see only a single composite character. Further, the entire virtual font layer would need to be mapped in some fashion, as the PDF contains the physical glyphs that are often combined in some way to provide the semantic characters. Supporting this would have required significantly more effort and heuristics, and there are already efforts in this direction from people much more knowledgeable and capable than I am. The most logical general solution is to use properly coded input, such as UTF-8, and where transformation to multiple glyphs is necessary, embed the appropriate mapping information directly in the PDF file.

The lack of success for other languages diminishes these proposed changes, but the changes are still important as they do provide reasonable support for English-language documents. Since PDF viewers are a moving target, as are the PostScript to PDF converters, the implementation provides for some future experimentation and extension.

7 Finding font encodings

In order to provide more than a proof of concept, I had to determine appropriate glyph names for the fonts provided with TeX Live, as well as provide a mechanism for end users to add their own glyph names for their own personal fonts.

Over the years others have translated nearly all of the METAFONT fonts provided with TeX Live, and as part of that process, reasonable encoding vectors have been created for the glyphs. I decided to leverage this work, so I wrote a script that located all the METAFONT sources in the TeX Live distribution, all the corresponding Type 1 fonts, and any encoding files used in the relevant `psfonts.map` file. A big Perl script chewed on all of this, extracting encoding vectors and creating appropriate files for `dvips`. Some of the encoding vectors use glyph names that are not particularly useful, and some use glyph names based on Unicode code points that are not currently recognized by the PDF viewers I tried. I did not want to edit the names in any way; I aimed for functional equivalence to using the Type 1 fonts. If improvements are made to the Type 1 font glyph names, or to the PDF viewers, I wanted to be able to pick up those improvements.

I considered having `dvips` read the encoding vectors directly from the Type 1 fonts, rather than extracting them and storing them elsewhere, but decided against this; I wanted `dvips` to use appropriate glyph names even if the Type 1 fonts didn't exist at all. This does introduce redundancy which can potentially lead to an inconsistency in the glyph names, but the fonts are currently mostly stable, and the glyph name extraction process can be repeated as needed if meaningful changes are made.

8 Storing and distributing encodings

After scanning all of the relevant METAFONT files and corresponding Type 1 files, I found there were 2885 fonts; storing the encodings separately one per font would require an additional 2,885 files in TeX Live, occupying about 5 megabytes. I felt this was excessive for the functionality added.

Karl Berry suggested combining all the encodings into a single file, along with a list of fonts using any particular encoding. Since there were only 138 distinct encodings, this gave tremendous compression, letting me store all of the encodings for all of the fonts in a single file of size 183K. This also enabled me to distribute a simple test Perl script that mimicked the changes so people could try them out without updating their TeX installation.

This combined file, called `dvips-all.enc`, provides the default encoding used by the 2885 distributed TeX Live METAFONT fonts. In every case that `dvips` looks for an encoding, e.g., for `cmr10`, it first searches for `dvips-cmr10.enc` and only falls back to the information in the combined file if the font-specific file is not found. This permits users to override the provided encodings, as well as define their own encoding for local METAFONT fonts.

The format of the encoding file is slightly different from that of other encoding files in TeX Live. The encoding file should be a PostScript fragment that pushes a single object on the operand stack. That object should either be a legitimate encoding vector consisting of an array of 256 PostScript names, or it should be a procedure that pushes such an encoding vector. It should not attempt to define the `/Encoding` name in the current dictionary, as some other encoding file formats do. A sample file, one that can be used for `cmr10` (and many other

Computer Modern fonts) is:

```
[/Gamma/Delta/Theta/Lambda/Xi/Pi/Sigma/Upsilon
/Phi/Psi/Omega/ff/fi/fl/ffi/ffl/dotlessi
/dotlessj/grave/acute/caron/breve/macron/ring
/cedilla/germandbls/ae/oe/oslash/AE/OE/Oslash
/suppress/exclam/quotedblright/numbersign
/dollar/percent/ampersand/quoteright/parenleft
/parenright/asterisk/plus/comma/hyphen/period
/slash/zero/one/two/three/four/five/six/seven
/eight/nine/colon/semicolon/exclamdown/equal
/questiondown/question/at/A/B/C/D/E/F/G/H/I/J
/K/L/M/N/O/P/Q/R/S/T/U/V/W/X/Y/Z/bracketleft
/quotedblleft/bracketright/circumflex
/dotaccent/quoteleft/a/b/c/d/e/f/g/h/i/j/k/l
/m/n/o/p/q/r/s/t/u/v/w/x/y/z/endash/emdash
/hungarumlaut/tilde/dieresis
128{/.notdef}repeat]
```

9 Deduplicating encodings

The encodings inserted in the fonts do use a certain amount of PostScript memory, and this memory usage is not presently accounted for in the memory usage calculation of `dvips`. The memory usage is small and modern PostScript interpreters have significant memory. Further, I doubt anyone actually sets the `dvips` memory parameters anymore anyway. So this is unlikely to be an issue. But to minimize the effect, and also to minimize the impact on file size, encodings that are used more than once are combined into a single instance and reused for subsequent fonts.

10 The `dvips` Changes

Almost all changes to `dvips` are located in the single new file `bitmapenc.c`, although a tiny bit of code was added to `download.c` to calculate an aggregate font bounding box, and the font description structure extended to store this information. I also added code to parse command line options and configuration file options to disable or change the behavior of the new bitmap encoding feature.

By default this feature is turned on. If no encoding for a bitmapped font is found, no change is made to the generated output for that font.

11 Testing the changes without updating

You can test my proposed changes to the `dvips` output files without updating your distribution or building a new version of `dvips`. The Perl script `addencodings.pl` [4] reads a PostScript file generated by `dvips` on standard input and writes the PostScript file that would be generated by a modified `dvips` on standard output. No additional files are required for this testing; the default encodings for the standard TeX Live fonts are built into the Perl script.

12 How to use a modified `dvips`

In general, `dvips` usage is unchanged. Warnings in the functionality of the bitmap encoding are disabled by default, so as to not disturb existing workflows; this may change in the future.

I add a single command line and configuration option, using the previously unused option character `J`. The option `-J0` disables the new bitmap encoding functionality. The option `-J` or `-J1` enables it but without warnings, and is the default. The option `-J2` enables it with warnings for missing encoding files.

13 Extension support

Remember that the encoding file is an arbitrary PostScript fragment that pushes a single object on the operand stack, and that object can be a procedure. I permit it to be a procedure to support experimenting with other changes to the font dictionary to improve text support in PDF viewers. For instance, if a technique for introducing Unicode code points for glyphs into a PostScript font dictionary is found and supported by various PostScript to PDF converters, such a procedure could introduce the requisite structures. The procedure will not be executed until the font dictionary for the Type 3 font is created and open.

To test this functionality, I created a `rot13.enc` file that defines a procedure that modifies the Encoding vector to swap single alphabetic characters much like the `rot13` obfuscation common during the Usenet days. With this modification, copying text from a PDF copies (mostly) content that has been obfuscated (except for ligatures). This brings us full circle to the current unreadable text copied from the original `dvips`.

References

[1] Adobe. Adobe glyph list specification. `https://github.com/adobe-type-tools/agl-specification`, August 2018.

[2] A. Jeffrey and F. Mittelbach. `inputenc.sty`. `https://ctan.org/pkg/inputenc`, 2018.

[3] R. Moore. Include CMap resources in PDF files from pdfTeX. `https://ctan.org/pkg/mmap`, 2008.

[4] T. G. Rokicki. Type 3 search code. `https://github.com/rokicki/type3search`, July 2019.

◇ Tomas Rokicki
Palo Alto, California
United States
`rokicki (at) gmail dot com`