T1part: Printing TeX Documents with Partial Type 1 Fonts

Sergey Lesenko
Institute for High Energy Physics
Scientific Information Department
Protvino, Moscow Region, Russia, P.O. Box 35
Email: `lesenko@desert.ihep.su`

Abstract

A large fraction of the scientific papers available on the Internet have been created by TeX and dvips. Most of these papers use Computer Modern fonts at 300 dpi, and thus neither preview well with a typical screen resolution of less than 100 dpi, nor take advantage of the higher resolution of current laser printers. This paper presents T1part, a set of subroutines that decompose Type 1 PostScript fonts, including only those characters needed in a particular document. This package, in conjunction with a high-quality freely available set of Computer Modern fonts, can provide for the distribution of compact PostScript files that preview legibly and print beautifully. In addition, this package allows the printing of much more complex documents using more fonts than have previously been available in dvips. The T1part subroutines are modular and can be easily incorporated into other drivers.

Introduction

Using Type 1 PostScript fonts in TeX documents has been problematic for a number of reasons. Not least among these problems is the requirement that either the fonts be included in the document itself, or that the fonts be available to all potential recipients of the document. Including such fonts in the document itself raises serious copyright issues, as well as causing the resulting file to be large and slow to print. In this paper, we present T1part, which allows such fonts to be included in PostScript output in a partial form. It is currently integrated with Tomas Rokicki's dvips program (Rokicki, 1994).

While commercial programs with this feature have been available for a number of years, this is the first freely available integrated implementation of this functionality.

The idea for this paper was suggested by Basil Malyshev's paper (Malyshev, 1994). The program is based on information available in the Adobe "Black Book" (Adobe Systems, 1990). In addition, I found work by Rajeev Karunakaran (1994), Chris B. Sears (1991), and Al Stevens (1994) to be useful.

This paper discusses the interface, algorithms, and finally the efficiency of using the T1part program.

Interface

T1part, as a subroutine, needs to be told what fonts and what characters in each font to include. The fonts are indicated by a file name pointing at a PFB or PFA Type 1 font. The set of characters to include can be specified either by a list of glyph names or by a set of character codes. In the latter case, the character codes must be translated into glyph names through the font's encoding vector.

The resulting output of the program is a partial font in PFA format. As integrated into dvips, the program inserts this font directly into dvips' output stream.

Algorithm

Before we present how T1part works, we must describe the format of a Type 1 font. We will first consider a font in PFB format; the PFA format is a simple modification of this. A Type 1 font in PFB format has the following structure and relevant keywords in each part:

- ASCII part
  - *keyword* **/Encoding**
- BINARY part (eexec encryption)
  - *keyword* **/lenIV**
  - *keyword* **/Subrs**
  - *keyword* **/CharStrings**
- ASCII part

By "ASCII", we mean the portion of the file that is not eexec encrypted.

Since the font contains three different parts, each part has its own parsing process. The minimum set of keywords listed above allows us to quickly parse the input at a high speed and with a simple parser.

The parsing process is as follows. First, we read the initial ASCII portion and search for the **Encoding** keyword. After finding it we define its type with the help of the next token. If the next token is **StandardEncoding**, we will assume that the Adobe Standard Encoding is the default font encoding vector.

If a reencoding has been specified for this font in the dvips `psfonts.map` file, then we use that reencoding to translate the character codes to glyph names. Otherwise, we try to parse an encoding from the input stream by searching for index and glyph name pairs. If we do not find such an encoding, we search for and parse an AFM file associated with the font.

Next, we parse the BINARY portion of the font. We start by performing the eexec decryption and loading the result directly and entirely into memory. We then scan the decrypted section for the keywords **/lenIV**, **/Subrs**, and **/CharStrings**. Each of the sections separated by these keywords has a similar structure.

When initially parsing the **Subrs** section, we simply identify what subroutines exist by number and keep track of the number of tokens in each subroutine definition. We do not initially send out any subroutines.

When we parse the **CharStrings** section, we initially identify what subroutines are associated with each character. We parse the subroutines for each used character, recursively diving into the **Subrs** entries as necessary, marking which subroutines are actually used. If there are multiple **Subrs** sections, as is the case with some hybrid fonts, we consider all such sections.

If a required character is a composite character, then the component characters will be marked and processed as above.

For efficiency, all selected subroutines are decrypted only once; a flag associated with each subroutine is used to indicate whether that subroutine has already been decrypted.

After finishing the scanning phases, the size of the **Subrs** and **CharStrings** arrays must change to reflect the deleted subroutines. The new values are:

- **Subrs** size — largest selected subroutine plus one
- **CharStrings** size — number of selected charstrings

We retain the indices of the subroutines for simplicity (so we do not need to rescan and modify each subroutine) and to easily guarantee that each subroutine only refers to those with a lower index (an Adobe requirement that prevents recursive subroutines).

Finally, the selected portions of the scanned memory are eexec encrypted and directed to the output in hexadecimal (PFA) form.

To finish up the font, the final ASCII portion is sent to the output without changes.

If the input font is in PFA format, the keywords that define the beginning of the hexadecimal eexec section are **currentfile eexec**; a line of all zeros marks the end of the section.

Results

T1part was tested by integrating it into dvips and running it over a number of files using the BaKoMa collection of Computer Modern fonts, as well as the Acrobat Reader selection of Adobe fonts. The results of these tests are:

- *Size of output file.* The total size of the resulting PostScript file when using T1part and PS fonts was compared with that when using bitmap PK fonts under both 300 dpi and 600 dpi. Usually, the bitmapped font output was slightly smaller at 300 dpi, but slightly larger at 600 dpi. Thus, using partial Type 1 fonts is practical from the perspective of final file size.

- *Reduction in PS fonts.* In our tests, we found that usually less than half of the characters from body text fonts were used, and a very small fraction of special fonts were used. On average, the size of the partial PostScript font created by T1part was less than 30% of the size of the original font.

The figures are given in detail in tables 1–3.

The output of the program has been tested with GhostScript (Deutsch, 1993), and the time and memory required to view documents created with partial fonts was less than those with the whole fonts.

It is clear that the popular paradigm of TeX — dvips — GhostScript will be more efficient when using the integrated T1part functionality.

Acknowledgements

I would like to thank Tomas Rokicki for his helpful discussion and suggestions.

References

Adobe Systems. *Adobe Type 1 Font Format, version 1.1.* Addison Wesley, 1990.

L. P. Deutsch. "Aladdin Ghostscript version 2.6.1". Electronic distribution from `ftp.cs.wisc.edu` via CTAN, 1993.

R. Karunakaran. "PFB2PFA program". Electronic distribution via `comp.sources.postscript`, volume 03 issue 51, 1994.

B. K. Malyshev. "Problems of the conversion of Metafont fonts to PostScript Type 1". In *Proceedings of TUG94*, edited by M. Goossens, Santa Barbara, CA. 1994.

T. Rokicki. "Dvips: A TEX Driver". Electronically distributed from `labrea.stanford.edu` via CTAN with `dvips`, version 5.58, 1994.

C. B. Sears. "CHARS program". Electronic distribution via `comp.sources.misc` volume 19 issue 94, 1991.

A. Stevens. "Quincy: a C interpreter". *'Dr. Dobb's' journal* (09), 1994. Electronic distribution as `ftp://ftp.mv.com/pub/ddj/1994/1994.09/qnc41.zip`.

Table 1: Dvips' output* with various modes of used fonts (dvips.dvi, version 5.58, used as input).

Font type	PS Type 1		PK				
Mode of font embedding	Full	Partial	Partial				
Output resolution (dpi)			300	600	1016	1200	1800
Output size (bytes)	762 981	398 358	424 529	1 012 195	3 082 215	4 487 168	14 933 095
Output with compressed bitmap fonts (bytes)	–	–	274 205	335 981	418 895	480 083	622 405
Disk space required for fonts** (bytes)	277 579	277 579	104 036	251 756	479 080	588 696	947 684

* Dvips' output size without embedding of fonts — 228 918 bytes

** PFB format is used for PS Type 1 fonts

Table 2: Efficiency of partial font downloading for each used font (dvips.dvi used as input)

Font name	cmbx10	cmr10	cmr8	cmsl10	cmti10	cmtt10	cminch	cmmi10	cmsy10	cmsy7	logo10	Total
Font version	*	*	*	*	*	*	**	*	*	*	**	
Percentage characters used	55	75	14	27	28	72	19	6	7	7	99	32
Partial font (bytes)	28 757	40 675	7 314	14 903	18 382	37 908	4 438	3 628	3 862	3 929	5 301	169 097
Full font (bytes)	52 768	53 960	53 360	54 870	65 948	52 915	23 211	63 752	52 613	54 951	5 372	533 720

Table 3: Results of using DC instead of CM fonts

Font name	dcbx10	dcr10	dcr8	dcsl10	dcti10	dctt10	cminch	cmmi10	cmsy10	cmsy7	logo10	Total
Font version	***	***	***	***	***	***	**	*	*	*	**	
Percentage characters used	34	44	9	18	17	44	19	6	7	7	99	23
Partial font (bytes)	25 766	34 039	6 732	14 555	15 827	36 450	4 438	3 628	3 862	3 929	5 301	154 527
Full font (bytes)	75 311	77 211	76 871	80 687	92 105	83 261	23 211	63 752	52 613	54 951	5 372	685 345

* BaKoMa/CM Fonts Collection (1.3, (Level-C), January 95)

** Paradissa Fonts Collection (1.0-prerelease, 1993)

*** BaKoMa/DC Fonts Collection (1.0, (Level-B), 1994)

This topic contains packages with fonts in Adobe Type 1 format. accanthis. Accanthis fonts, with LaTeX support. adforn. OrnementsADF font with TeX/LaTeX support. adfsymbols. SymbolsADF with TeX/LaTeX support.Â Calligraphic fonts for use with LaTeX in T1 encoding. bakoma-fonts. Computer Modern and AMS fonts in outline form. baskervaldadf. Baskervald ADF fonts collection with TeX/LaTeX support. baskervaldx. Extension and modification of BaskervaldADF with LaTeX support.Â Using Utopia fonts in LaTeX documents. fpl. SC and OsF fonts for URW Palladio L.Â A partial implementation of the old msym10 font. musixtex-fonts. Fonts used by MusixTeX. musixtex-t1fonts. Adobe Type 1 versions of MusiXTeX fonts. mxedruli. I use Adobe Postscript Type 1 fonts regularly in my documents, and I have found that Word 2010 (and presumably all of the Office 2010 applications) has several problems working with PostScript fonts. Most notably, the "Save as PDF" function will create documents with an odd gradient effect applied to all PostScript fonts. Secondarily, the Word editor frequently glitches and renders PostScript fonts using a sans-serif font that looks to be Arial. The following are a demonstration of the issue: This is an original document typed in Minion, an Adobe PostScript Type 1 font.Â When saving documents using fonts with PostScript outline data, Word 2010 only supports outputting the text as a static image, and does not support outputting the text as selectable text. Andrew pointed me to the type 1 specification and the True Type specification, and I started learning about the Type 1 fonts. Unfortunately the Type 1 specification was full of references to the Postscript Language Reference Manual, which was only available in book format. The chapter on creating the Type 1 outline was full of references: rlineto behaves the same as the rlineto postscript command or.