MFCONFIG: A METAFONT plug-in module for the Freetype rasterizer

Jaeyoung Choi, Sungmin Kim, Hojin Lee and Geunho Jeong

Abstract

One of METAFONT's advantages is its ability to create font variants by changing values of parameters representing font characteristics. This advantage can be applied not only to Latin alphabetic characters, but also to complicated CJK (Chinese-Japanese-Korean) characters. Second, font families like bold, italic, and bold-italic do not need to be created separately for METAFONT, because it can automatically generate a variety of styled fonts via changing parameter values. Therefore, METAFONT can reduce the development time and cost for production of a font family. It is not possible, however, to directly use METAFONT in modern font engines; the output must be changed to an outline font format if it is to be used in a current computing environment.

In this paper, a module named MFCONFIG, enabling direct usage of METAFONT on Linux is proposed. It is a plug-in module for the FONTCONFIG library, and must also be installed with the popular rasterizer Freetype. FONTCONFIG and Freetype are already compatible with other digital font types, both bitmap and outline; MFCONFIG adds METAFONT support. Furthermore, by setting various parameters, the proposed module supports a variety of font styles, all generated from METAFONT.

1 Introduction

Text is an effective way to communicate and record information. With the growing use of smart devices, digital fonts are more commonly used than analog fonts. Although many styles of digital fonts have been created, they still do not meet the requirements of all users, and users cannot change digital font styles freely [10]; for instance, if a user wants to use a thinner outline font, either he/she has to find a thinner styled font, or an in-application function to change the font thickness. As several different features of font style are needed, though, such searching or changing of font style of an existing font is typically not easy. A perfect application satisfying users' diverse requirements regarding font styles does not exist. Also, it is impossible to provide all styles of all fonts in accordance with users' preferences.

Currently, popular digital fonts, either bitmap or outline, have limits on changing font style [8]. However, METAFONT is a structured font definition that allows users to change the font style freely. METAFONT, a font system for TEX, was created by D. E. Knuth [4]. It has functions for drawing characters and parameters to determine the font styles. When the user changes the parameters, the font style is changed automatically. Therefore, a variety of styled fonts can be generated from one METAFONT font. Figure 1 shows a variety of styled fonts created by the changing of the thickness and slant; examples of two thickness and slant styles for the Latin letter "A" and the Chinese character "漢" are shown. If other features such as serif and pen are applied together, a greater variety of styled fonts can be generated.

Most users, however, are unable to use METAFONT on their PCs because current font engines do not support METAFONT. METAFONT fonts are expressed as program source code, completely different from standard digital bitmap and outline fonts. If a user wants to use a specific METAFONT font in a general font engine such as Freetype, then he/she needs to convert the METAFONT font into the needed outline font format.

In the case of Roman characters, the design of "only" several hundred characters is required; moreover, their shapes are generally simpler than those of CJK (Chinese-Japanese-Korean) characters. In the mid-1980s, when METAFONT was introduced, hardware was not fast enough for real-time conversion of the METAFONT fonts into the corresponding bitmap or outline fonts. Moreover, outline fonts are more commonly used than METAFONT.

Current PC hardware, however, has sufficient performance for the real-time execution of METAFONT. If METAFONT could be used directly in a

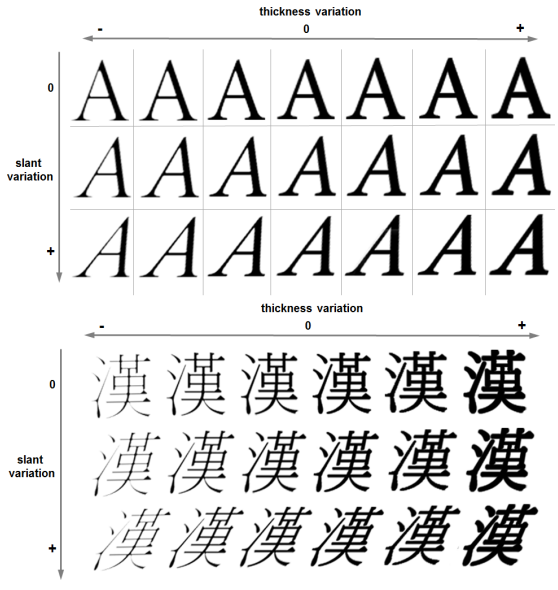

Figure 1: METAFONT style variations

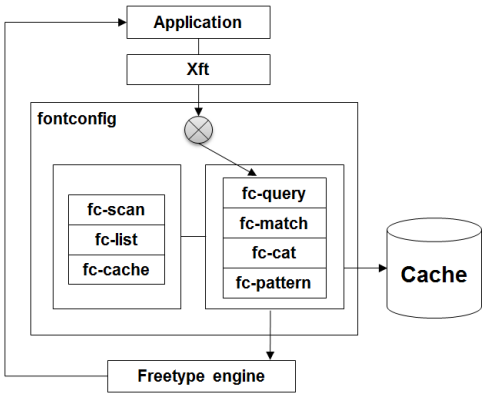

Figure 2: Architecture of FONTCONFIG

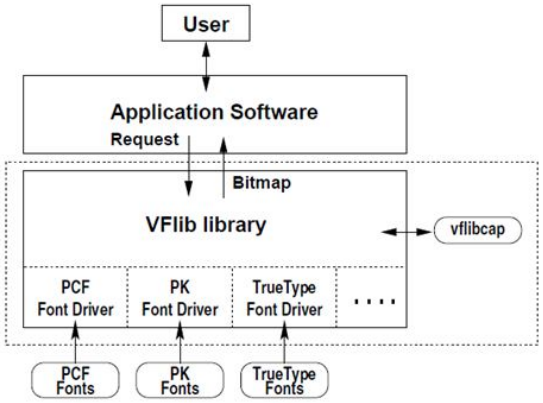

Figure 3: Architecture of VFlib

PC, then users could easily make and use a variety of styled fonts by themselves. As we previously saw, one METAFONT font can generate a variety of styled fonts by changing style parameter values. Therefore, METAFONT can save great amounts of time and repeated effort in terms of font design to make font families of plain, italic, bold, and bold-italic fonts. In particular, in the case of CJK usage, METAFONT could be an effective way to make and display a variety of font styles — because, compared to alphabetic scripts, CJK characters are both complicated in shape and expressed by combinations of radicals.

In this paper, a METAFONT module that enables direct METAFONT usage on Linux is proposed. It is possible to plug this module into FONTCONFIG to provide digital font information to the FreeType engine. When the MFCONFIG module is used, conversion of a METAFONT into corresponding outline fonts becomes unnecessary. It is simple to change font styles by applying new parameter values. Also, this module can interact with most existing FONTCONFIG functions without modification of either FONTCONFIG or Freetype. The MFCONFIG module therefore has good usability and compatibility for the support of METAFONT in the Freetype engine.

2 Existing font systems

FONTCONFIG [7] provides extended font configuration for the Freetype rasterizer, and the Xft (X–FreeType) library [6] has been developed to provide interfaces between applications and Freetype. These font libraries are able to collect font information on the current PC system such as font paths, style information, extra meta information, and so on. Figure 2 shows a font-output sequence that is required for applications using the X Window system under Linux. After an application sends a font request according to name and style to the Xft library, it also delivers the request information to FONTCONFIG. FONTCONFIG uses its internal commands to check the following conditions: (1) whether the requested font is installed, (2) whether the style of the user's request has been applied to the stored font, and (3) whether the requested font has already been stored with the printing format in the cache. (4) If the requested font is not stored in the cache, it needs to be converted into the requested printing format and stored in the cache, and (5) the requested font in the cache is selected and then delivered to Freetype. Lastly, the requested font is printed with the font styles.

FONTCONFIG is a library for Freetype, and capable of supporting general digital font formats that can also be processed in Freetype. The architecture of FONTCONFIG is shown in Figure 2. It can support TrueType, OpenType, Type1, CFF, PFR, and BDF, but it does not support METAFONT. For the direct support of METAFONT in FONTCONFIG, it might be necessary to change the internal implementation of FONTCONFIG; for instance, changing the overall processes in FONTCONFIG from `fc-scan` (for font searching) to `fc-pattern` (for the matching of the styled font pattern). This is not a simple task.

Additionally, the Xft library interface exists between an application and FONTCONFIG, intended for providing font information such as font name and size. It would not be a good approach to modify Xft to support METAFONT, as it would likely reduce Xft's performance.

VFlib [2, 3] is a font driver system for supporting a variety of font types. The system supports virtual fonts like BDF, PCF, and TrueType, as shown in Figure 3. It provides a database of general font information, and a useful API for the supported font

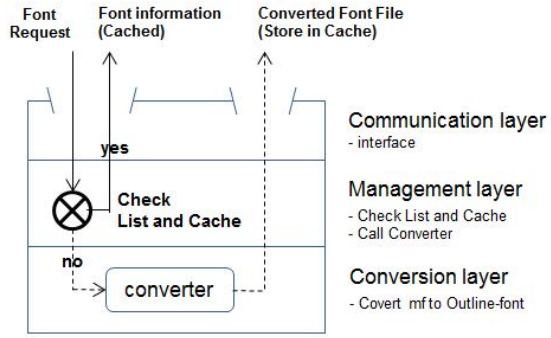

Figure 4: Three layers of the MFCONFIG module

types. VFlib includes separate modules for each font type, so a new module could be added to support METAFONT. But VFlib is a complicated system consisting of many different kinds of font drivers and an information dataset of default font information. In addition, the VFlib interface is required for an application to use the VFlib library. Therefore, to add a METAFONT module to VFlib, additional functions must be implemented for every relevant application, which is not practical. So, VFlib is not suitable to add support for METAFONT.

The proposed MFCONFIG module in this paper combines the following two features: (1) the process for the printing of digital fonts in FONTCONFIG, and (2) the font driver architecture of VFlib. The module can process METAFONT independently, and it can be easily installed or removed since it is implemented as a plug-in module. Also, the steps that are used for its implementation are similar to FONTCONFIG's internal commands, so METAFONT can be used along with the existing digital font formats.

3 Implementing the MFCONFIG module

As shown in Figure 4, the MFCONFIG module consists of the following three layers: communication, management, and conversion. The communication layer provides an interface between FONTCONFIG and MFCONFIG. The management layer checks whether the requested METAFONT is ready in the cache. If not, it sends a request message to the conversion layer to convert the METAFONT font. The conversion layer makes a new outline font file by using the requested METAFONT and the customized style values. The resulting outline font is stored in the cache.

As shown in Figure 5, MFCONFIG can be used as a plugin for FONTCONFIG. First, an application requests a font from the Xft library (step 1). Next, FONTCONFIG sends the font information and the values of the style parameters to MFCONFIG through the interface of the communication layer (step 2). This interface checks if the requested font is a METAFONT font or not.

In the case of METAFONT, *mf-query* analyzes the requested information, and *mf-match* tries to find this METAFONT from *mf-list*. If *mf-list* does not have the information, the requested METAFONT font is not installed. In this case, *mf-query* returns a "not found" flag to FONTCONFIG (step 3). Otherwise, if the METAFONT font is installed and is already stored in the cache, *mf-query* returns a "found" flag to FONTCONFIG (step 3).

One more case: the requested METAFONT font is installed, but the corresponding outline font is not yet in the cache. In this case, *mf-converter* in the conversion layer needs to convert the METAFONT font into the corresponding outline font (step 2a). In this step, the METAFONT font and the styled parameter values from the application are required for the conversion. After conversion, the outline font is stored in the cache (step 2b), and *mf-query* sends the "found" flag to FONTCONFIG (step 3).

After step 3, the remaining steps are the default steps of FONTCONFIG. The font information is sent to the internal programs of FONTCONFIG (step 4) that try to find the corresponding outline font in the cache (step 5); then, this outline font is sent to the Freetype rasterizer (step 6). If MFCONFIG returns the "not found" flag, then FONTCONFIG uses a default font file. Lastly, the Freetype engine renders the outline font that was made from the requested METAFONT with the styled parameter values.

The details of the three layers in MFCONFIG are presented below. First, the communication layer, which is an interface between FONTCONFIG and MFCONFIG, is the starting point of the MFCONFIG module. Therefore, the Freetype engine receives METAFONT font information from FONTCONFIG just as with existing font formats. The main functions of the interface are as follows: (1) delivery of the requested METAFONT information to the management layer, (2) returning results to FONTCONFIG, and (3) storage of the outline-font file from *mf-converter* in the cache memory.

The major programs of the MFCONFIG module operate in the management layer. This layer is in charge of "searching" and "managing". "Searching" is an independent function, finding all installed METAFONT fonts and storing the information in a list. This list is used for checking whether a specific font is installed or not, and for fetching its information quickly. The searching is implemented in *mf-scan* and *mf-list* which, as shown in Figure 6,

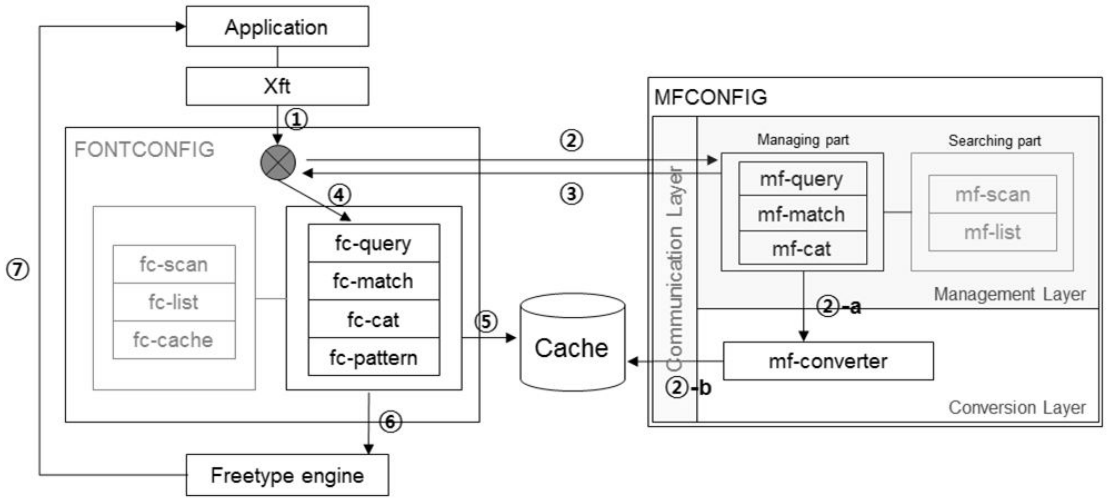

Figure 5: MFCONFIG architecture linked into fontconfig

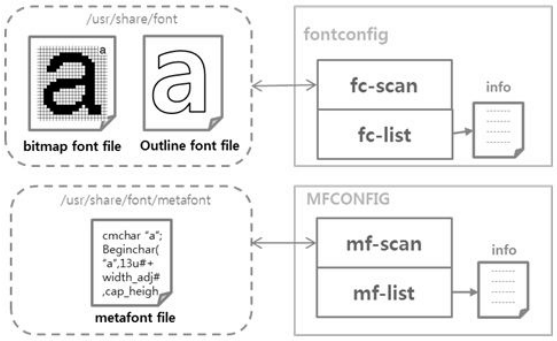

Figure 6: Management layer, cf. FONTCONFIG

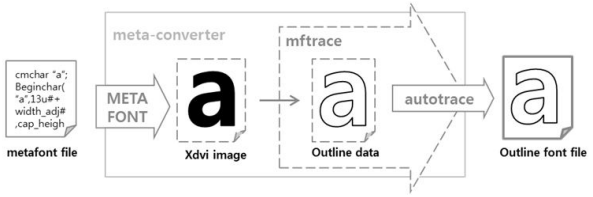

Figure 7: Conversion layer operation

work similarly to FONTCONFIG's *fc-scan* and *fc-list*.

"Management" is a core process of the MFCONFIG module that is responsible for the following actions: (1) checking if the requested METAFONT font is prepared in the list, (2) checking if the corresponding outline font is stored with the requested style in the cache, and (3) if the outline font is not so stored, check whether conversion of the METAFONT font into the corresponding outline font is needed. If the outline font has already been prepared in the cache, then a notification is sent directly from MFCONFIG to FONTCONFIG to use it, and FONTCONFIG sends the cached outline font to Freetype. If the outline font is not stored in the cache, then the conversion layer converts the METAFONT font into the corresponding outline font by applying the style parameters, as shown in Figure 7. The resulting outline font is then stored in the cache, and a notification from the management layer through the communication layer tells FONTCONFIG to use the font.

Thus, the work of the MFCONFIG module is perfectly compatible with the standard FONTCONFIG, and it can provide new functions to support METAFONT. The module handles management of METAFONT fonts and their conversion to corresponding outline fonts in real time. When a different style of an METAFONT font is requested, MFCONFIG can conveniently display the resulting font on the screen by applying the style values to the METAFONT fonts. Therefore MFCONFIG provides good usability for METAFONT. In addition, it is not necessary to generate the font family set of plain, bold, italic, and bold-italic in advance with respect to MFCONFIG, because the font styles can be generated easily.

4 Examining the MFCONFIG module

For performing the experiments of this study, an application for the use of the X Window system in Linux was developed, and the display of a text file was attempted with the use of a variety of font files. The TrueType font family FreeSerif was used in the usual four font styles (*normal, bold, italic, bold-italic*) and Computer Modern was used for METAFONT. The Computer Modern fonts were examined with the four similar styles *normal, thickness, italic,* and *thickness+italic*. The sample text comprises over 2,000 words and over 8,800 characters, including

Styles	Output	Font files
Normal	Computer	FreeSerif.ttf
Bold	**Computer**	FreeSerifBold.ttf
Italic	*Computer*	FreeSerifItalic.ttf
Bold+Italic	***Computer***	FreeSerifBoldItalic.ttf

Table 1: The FreeSerif font family

Style (variable)	Style 1	Style 2	Style 3
Normal	Computer (basic)	Computer (width x2)	Computer (width / 3)
Stroke (hair, stem, curve)	Computer (+20,+10,+10)	Computer (+30,+10,+10)	Computer (+20,+20,+20)
Slant (slant)	*Computer* (1/4)	*Computer* (1/2)	*Computer* (1/3)
Stroke + Slant (slant, Hair, Stem, Curve)	*Computer* (1/4,+20,+10,+10)	*Computer* (1/2,+30,+10,+10)	*Computer* (1/3,+20,+20,+20)

Table 2: Computer Modern fonts in various styles, made by changing style variables

space characters. For performance analysis of the Freetype rasterizer, the time between the requesting of a font with styles from an application and the successful display of text on screen was measured and compared.

Table 1 shows the FreeSerif font family in the four styles, and Table 2 shows 12 styles for the Computer Modern METAFONT. These styles were all made from one original METAFONT font by simple changes of the style parameters. Therefore, the METAFONT font has a good capability of generating various font styles.

For displaying text in an application, the four FreeSerif files from Table 1 and Style 1 of Computer Modern from Table 2 were used. For the CM style, the four parameter values are *hair, stem, curve, and slant*. The three parameters of *hair, stem*, and *curve* are related to the *bold* style, but these parameters are different for lowercase and uppercase. The *slant* parameter is related to the *italic* style. The chosen parameter values for the representation of a bold style are *hair*+20, *stem*+10, and *curve*+10, while *slant* is 0.25 for a representation of the italic style.

Table 3 shows the average time to print both the FreeSerif and Computer Modern contents, and Figures 8 and 9 show the displayed results. In this experiment, with the FreeSerif TrueType fonts, results from 10 ms to 30 ms were obtained, and the

Type	FreeSerif	Computer Modern
(a) Normal	15 ms (10~30)	70 ms (50~80)
(b) Bold	18 ms (10~30)	85 ms (70~100)
(c) Italic	16 ms (10~30)	105 ms (70~110)
(d) Bold+italic	16 ms (10~30)	100 ms (90~120))

Table 3: Average time for display of TrueType and METAFONT fonts (milliseconds)

average time is 16 ms. Therefore, extra time was required for the conversion of the TrueType fonts. In the case of the Computer Modern METAFONT font, the result is much slower than for FreeSerif, because of the additional time needed for the conversion of the METAFONT font into the corresponding outline font. The obtained results are from 50 ms to 120 ms, and the average time is 90 ms. Even though this is 10 times slower, 90 ms is still a reasonable time for rendering text to a display. Thus, we can conclude that the MFCONFIG module can be used with FONTCONFIG to support METAFONT in (almost) real time on a modern Linux PC.

The MFCONFIG module is a convenient system to provide users with various styled fonts on screen by applying style parameters directly to the METAFONT font. An METAFONT font can be used in just the same way as a TrueType font.

In this paper, we discuss the METAFONT font Computer Modern, which provides alphanumeric values and symbols. It is possible to perform tests with other METAFONT fonts from CTAN (Comprehensive TEX Archive Network) directories that support languages such as Russian and Thai. Unfortunately, difficulty was experienced when complicated CJK fonts are used.

CJK font definitions are very complicated compared to alphabet-based fonts, and they are composed of several thousand phonemes. A number of studies have been conducted to partially implement CJK fonts, such as Hóng-zì [5, 11, 12] and Tsukurimashou [9], including the use of a structural font generator using METAFONT for Korean and Chinese [1], and others. However, CJK fonts created with METAFONT have still not nearly reached the level of quality and practicality reached by commercial offerings. The authors expect that using the MFCONFIG module for the generation of CJK fonts will take more time. It may, however, be possible to solve this problem by optimizing the meta-converter in the conversion layer. Currently, the meta-converter works with *mftrace* and *autotrace* programs, which take a long time to generate outline fonts.

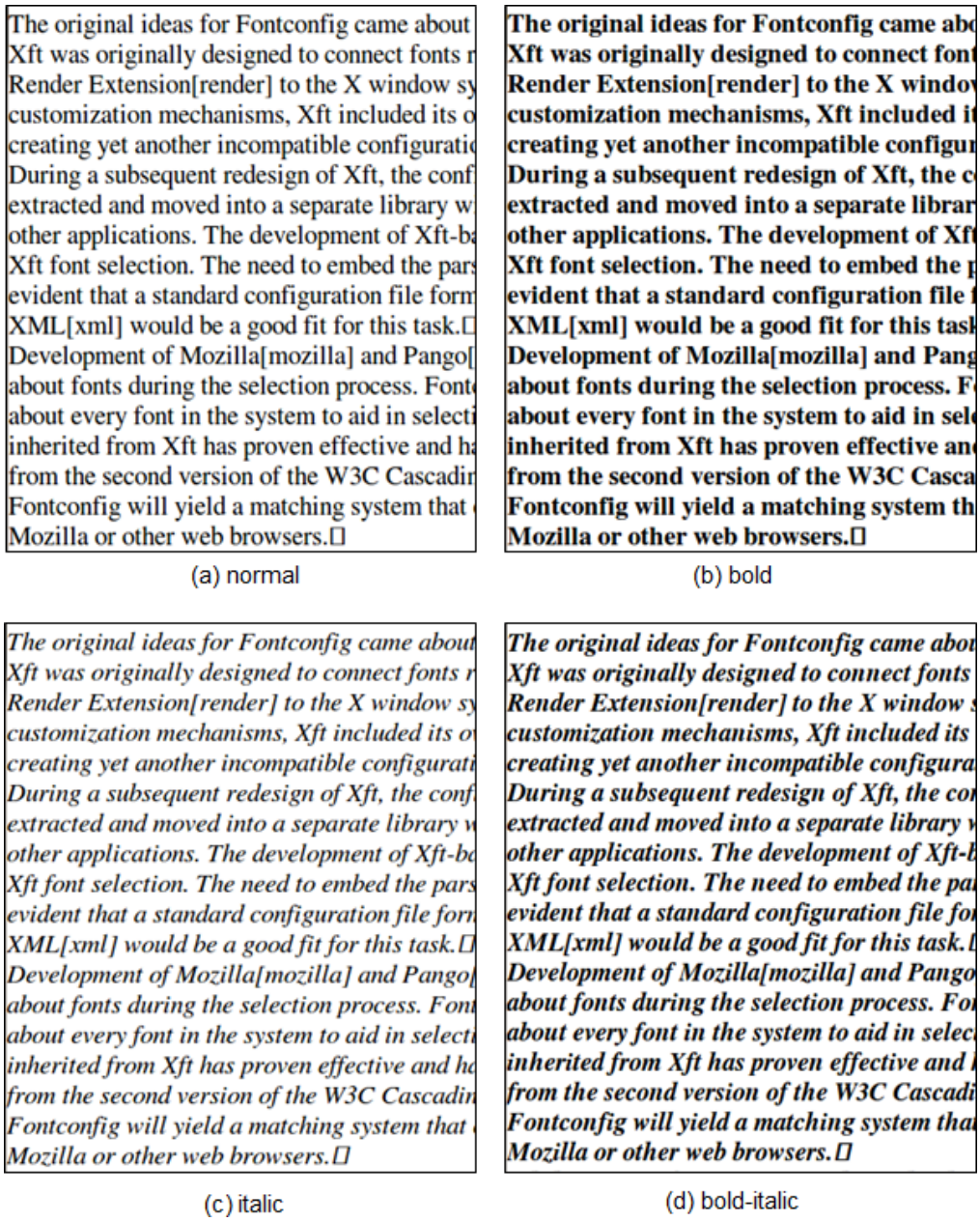

Figure 8: Displayed text in FreeSerif in the usual four styles

5 Conclusion

In this paper, the MFCONFIG module, enabling the direct use of METAFONT on Linux, is proposed. It is installed and used with the popular Freetype rasterizer. MFCONFIG is a plug-in module for the FONTCONFIG library The module supports a variety of the styled fonts that are generated from METAFONT by setting different parameters.

Existing digital font formats — notably the outline fonts of Type 1, TrueType, and OpenType — either do not typically allow users to change their styles, aside from font size scaling. From the experiments of the present study, it has been demonstrated that a variety of fonts can be directly generated on screen by applying different style parameters to a prototype METAFONT font using a Freetype rasterizer that is installed with MFCONFIG. Furthermore, the fonts could be seen within an average time of 90 ms, which is a barely noticeable duration.

The MFCONFIG module targets METAFONT fonts to be used with Freetype, a well-known rasterizer. MFCONFIG can be used effectively with

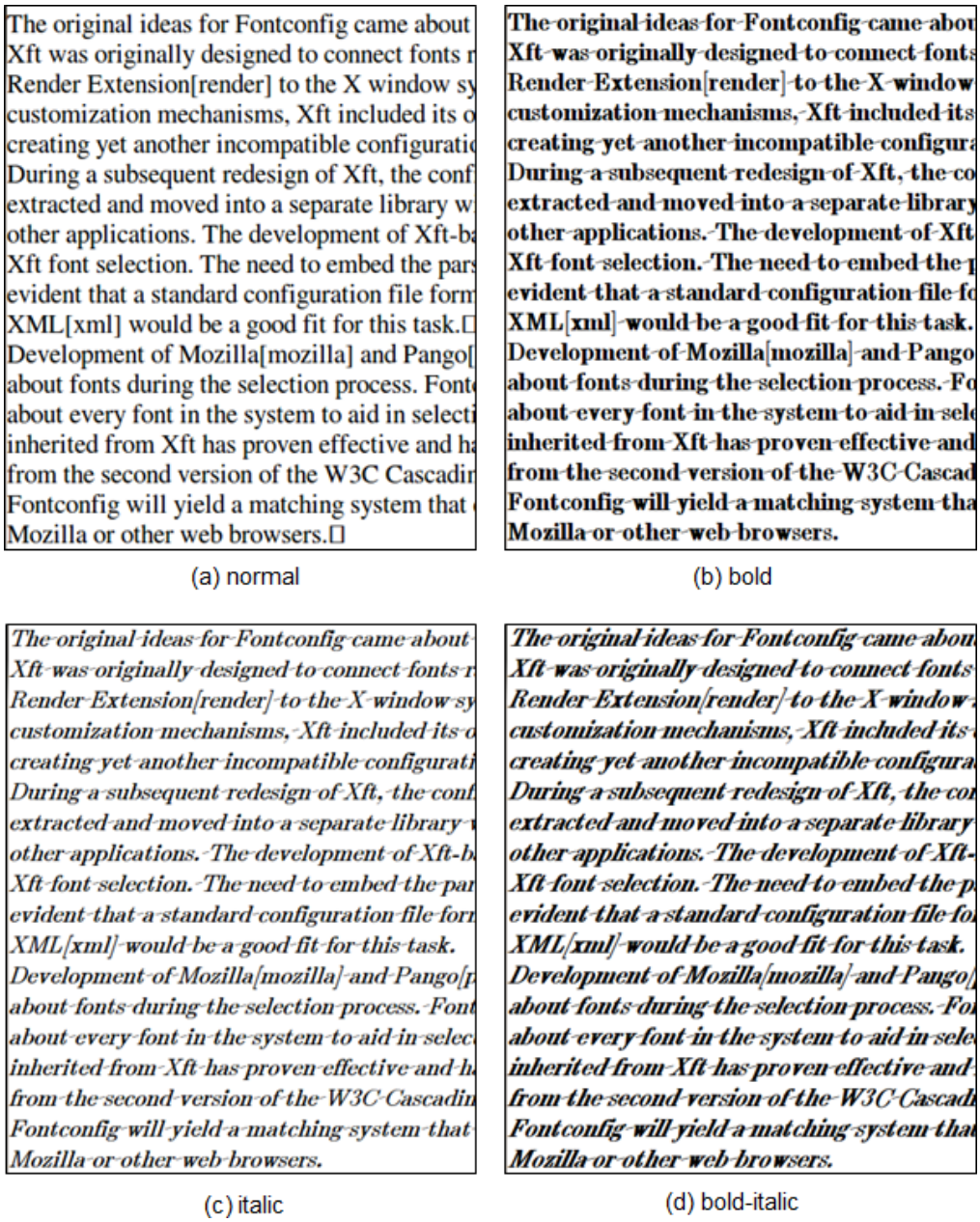

Figure 9: Displayed text in Computer Modern in the usual four styles

alphabet-based fonts, which are relatively simple and have a limited number of characters. However, there are only a few METAFONT fonts for various languages. It will likely take a longer time to process CJK METAFONT fonts, which have complicated shapes and more than several thousand phonemes. Further work will focus on these CJK METAFONT fonts to improve performance, and otherwise optimize the MFCONFIG module. In addition, this module will be experimented with as a font driver in the Freetype rasterizer.

Acknowledgements

This work was supported by an Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korean government (MSIP) (No.R-20160301-002987, Technology Development Project for Information, Communication, and Broadcast).

References

[1] Gyungjae Gwon, Minju Son, Geunho Jeong, and Jaeyoung Choi. Structural font generator using METAFONT for Korean and Chinese, 2016. In preparation.

[2] H. Kakugawa, M. Nishikimi, N. Takahashi, S. Tomura, and K. Handa. A general purpose font module for multilingual application programs. *Software: Practice and Experience*, 31(15):1487–1508, 2001. dx.doi.org/10.1002/spe.424.

[3] Hirotsugu Kakugawa. VFlib: A general font library that supports multiple font formats. *Cahiers GUTenberg*, iss. 28–29:211–222, March 1998. cahiers.gutenberg.eu.org/cg-bin/article/CG_1998___28-29_211_0.pdf.

[4] Donald E. Knuth. *Computers and Typesetting, Volume C: The METAFONTbook*. Addison-Wesley, 1986.

[5] Javier Rodríguez Laguna. Hóng-zì: A Chinese METAFONT. *TUGboat*, 26(2):125–128, 2005. tug.org/TUGboat/tb26-2/laguna.pdf.

[6] Keith Packard. The Xft font library: Architecture and users guide. *Proceedings of the 5th annual conference on Linux Showcase & Conference*, 2001. keithp.com/~keithp/talks/xtc2001/paper/.

[7] Keith Packard, Behdad Esfahbod, et al. Fontconfig. fontconfig.org.

[8] Y. Park. Current status of Hangul in the 21st century [in Korean]. ⟨*The T*⟩ *Type and Typography magazine*, vol. 7, August 2012. www.typographyseoul.com/news/detail/222.

[9] Matthew Skala. Tsukurimashou: A Japanese-language font meta-family. *TUGboat*, 34(3):269–278, 2013. tug.org/TUGboat/tb34-3/tb108skala.pdf.

[10] S. Song. Development of Korean Typography Industry [in Korean]. *Appreciating Korean Language*, 2013. www.korean.go.kr/nkview/nklife/2013_3/23_0304.pdf.

[11] Candy L.K. Yiu and Jim Binkley. Qin notation generator. *TUGboat*, 26(2):129–134, 2005. tug.org/TUGboat/tb26-2/yiu.pdf.

[12] Candy L.K. Yiu and Wai Wong. Chinese character synthesis using MetaPost. *TUGboat*, 24(1):85–93, 2003. tug.org/TUGboat/tb24-1/yiu.pdf.

◇ Jaeyoung Choi
Soongsil University, Seoul, Korea
choi (at) ssu.ac.kr

◇ Sungmin Kim
Soongsil University, Seoul, Korea
sungmin.kim (at) ssu.ac.kr

◇ Hojin Lee
Soongsil University, Seoul, Korea
hojini (at) ssu.ac.kr

◇ Geunho Jeong
Gensol Soft, Seoul, Korea
ghjeong (at) gensolsoft.com