FreeType_MF_Module: A module for using METAFONT directly inside the FreeType rasterizer

Jaeyoung Choi, Ammar Ul Hassan, Geunho Jeong

Abstract

METAFONT is a font description language which generates bitmap fonts for the use by the TEX system, printer drivers, and related programs. One advantage of METAFONT over outline fonts is its capability for producing different font styles by changing parameter values defined in its font specification file. Another major advantage of using METAFONT is that it can produce various font styles like bold, italic, and bold-italic from one source file, unlike outline fonts, which require development of a separate font file for each style in one font family. These advantages are especially applicable when designing CJK (Chinese-Japanese-Korean) fonts, which require significant time and cost because of the large number of characters used in Hangeul (Korean character) and Hanja (Chinese character). However, to use METAFONT in current font systems, users need to convert it into its corresponding outline font. Furthermore, font rendering engines such as FreeType don't support METAFONT.

In this paper, we propose *FreeType_MF_Module* for the FreeType rasterizer. The proposed module enables direct usage of METAFONT just like any other font (outline or bitmap) supported in the FreeType rasterizer. Users of METAFONT don't need to pre-convert METAFONT fonts into corresponding outline fonts as FreeType_MF_Module automatically performs this. Furthermore, FreeType_MF_Module allows the user to easily generate multiple font styles from one METAFONT source file by changing parameters.

1 Introduction

In today's information society, much traditional pen and paper usage for communication between people has been increasingly replaced by computers and mobile devices. Text has become an effective source for gathering information and a means of communication between people. Although people commonly use smart devices these days with effective resources like media and sound, text generally plays the key role of interaction between user and device. Text is composed of characters, and these characters are physically built from specific font files in the digital environment's system.

Fonts are the graphical representation of text in a specific style and size. These fonts are mainly categorized in two types: outline fonts and bitmap fonts. Outline fonts are the most popular fonts for producing high-quality output used in digital environments. However, to create a new font style as an outline font, font designers have to design a new font with consequent extensive cost and time. This recreation of font files for each variant of a font can be especially painful for font designers in the case of CJK fonts, which require designing of thousands individual glyphs one by one. Compared to alphabetic scripts, CJK scripts have both many more characters and generally more complex shapes, expressed by combinations of radicals [3]. Thus it often takes more than a year to design a CJK font set.

A programmable font language, METAFONT, has been developed which does not have the above disadvantages of outline fonts. METAFONT is a programming language created by D.E. Knuth [1] that generates TEX-oriented bitmap fonts. A METAFONT source file is radically different from an outline font file: it consists of functions for drawing characters and has parameters for different font styles. By changing the parameters defined in a font specification file, various font styles can be easily generated. Therefore, a variety of font variants can be generated from one METAFONT source.

However, in practice users are unable to use METAFONT on modern systems, because current font engines like FreeType [2] do not provide any direct support of METAFONT. Unlike standard bitmap and outline fonts, METAFONT is expressed as a source code that is compiled to generate fonts. To use METAFONT in a general font engine like the FreeType rasterizer, users have to convert each metafont into its corresponding outline font. When it was developed in the 1980s, standard PC hardware was not fast enough to do real-time conversion of METAFONT source into a corresponding outline font. Current PC hardware, however, is fast enough to do such real-time conversion.

In this paper, a METAFONT module for the FreeType rasterizer (FreeType_MF_Module) is proposed. The proposed module enables direct use of METAFONT in FreeType, just like any other outline and bitmap font modules. With FreeType_MF_Module, users don't need to pre-convert a METAFONT font into its corresponding outline font before using it with the FreeType rasterizer, as FreeType_MF_Module automatically performs this. It allows users to easily generate variants of font styles by applying different parameter values. This module is directly installed in the FreeType rasterizer just like

its default font modules, thus minimizing any reliability and performance issues. We have tested our proposed module by generating different font styles with METAFONT and compared its performance with default FreeType modules and our previous research.

This paper is organized as follows. In Section 2, related research regarding font modules and libraries is discussed. The architecture of FreeType_MF_Module is explained in Section 3. Section 4 demonstrates how FreeType can support METAFONT, via some testing of FreeType_MF_Module. The performance of FreeType_MF_Module is also compared with FreeType default modules and other researches in this section. Section 5 gives concluding remarks.

2 Previous research and its problems

MFCONFIG [4] is a plug-in module for Fontconfig [5]. It enables the use of METAFONT on GNU/Linux and other Unix font systems. Figure 1 shows the architecture of the MFCONFIG module linked with Fontconfig.

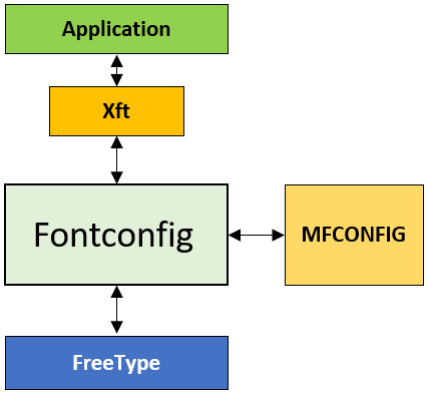


Figure 1: Basic architecture of MFCONFIG module

Although MFCONFIG does provide support for METAFONT in Fontconfig-based font systems, it has performance and dependency problems. Since MFCONFIG is plugged into a high-level font system, i.e., Fontconfig, and not at the low-level FreeType rasterizer, its performance is very slow compared to the font-specific driver modules supported by FreeType. Whenever the client application sends a METAFONT file request, Fontconfig communicates with MFCONFIG, performs operations, and then sends input to FreeType for rendering text. This whole process becomes slow because of the high-level operations before FreeType receives its input.

Other than the performance problem, MFCONFIG also has a dependency problem. As it works with the Fontconfig library, this means that in a font environment not using Fontconfig, this module cannot be used. Fontconfig is mainly used in the font system for GNU/Linux and some other Unix operating systems, so MFCONFIG cannot be supported in other environments, such as Windows and Mac OS X.

VFlib [6], a virtual font library, is a font rasterizer developed for supporting multilingual fonts. VFlib can process fonts which are represented in different font formats and outputs glyphs as bitmap images from various font files. VFlib supports many font formats like TrueType, Type 1, GF, and PK bitmaps [7], et al. It provides a unified API for accessing different font formats. A new module can be added in this font library for adding support for METAFONT but this library has its own drawbacks: as it supports many different font formats, and requires support from a database, it can be too heavy for embedded systems. It is also dependent on additional font libraries, such as the FreeType engine for TrueType font support and T1lib [8] for Type 1 font support, so it has its own dependency problems as well. Therefore, VFlib is not suitable for adding METAFONT support.

FreeType is a font rasterizer. It can produce high quality output for mainly two kinds of font formats, both outline and some bitmap formats. FreeType mainly supports font formats such as TrueType, Type 1, Windows, and OpenType fonts using the same API, independent of the font format. Although FreeType supports many different font formats, it doesn't provide any support for METAFONT directly. If there were a module for FreeType that directly supports METAFONT, users could take advantage of the METAFONT features above, e.g., generating variants of font styles by just changing parameter values. MFCONFIG's problems can also be resolved using such a module.

The proposed FreeType_MF_Module in this paper reuses the process for printing METAFONT from the MFCONFIG module. FreeType_MF_Module intends to solve the two problems of the MFCONFIG module. METAFONT can be used with any system having FreeType using the proposed module. As it is implemented like any other default FreeType module, it can be easily installed or uninstalled.

3 Implementation of FreeType_MF_Module in the FreeType rasterizer

3.1 FreeType_MF_Module as an internal module of FreeType

FreeType can support various font formats. Processing a font file corresponding to its format is done by an internal module in FreeType. This internal module is called a font driver. FreeType contains a configuration list of all driver modules installed, in a specific order. When FreeType receives a request

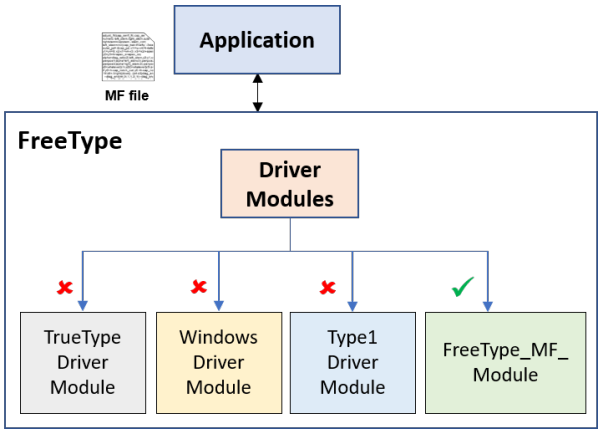


Figure 2: Process of selecting a module in FreeType

for a font file from an application, it passes this request to the driver module at the top of the list for processing. This module performs some internal operations to check if this font format can be processed or not. If this driver module supports the request, it performs all other operations to process the font file request. Otherwise the request is sent to the second driver module mentioned in the list. This process continues until a font driver is selected for processing the font file request. If no font driver can process the request, an error message is sent to the client application.

In our case, FreeType_MF_Module is directly installed inside FreeType just like its other internal modules. When the client application sends a request for a METAFONT file, FreeType_MF_Module receives this request and processes it. Figure 2 shows how FreeType will select a driver module for processing a METAFONT file request.

FreeType_MF_Module consists of three submodules: Linker module, Administrator module, and Transformation module.

3.2 Linker module

Linker module is the starting point of FreeType_MF_Module. It is mainly responsible for linking FreeType internal modules with FreeType_MF_Module. It is divided into two parts: inner meta interface and outer meta interface. The inner meta interface receives font file requests from internal modules and delivers it to the Administrator module for processing. After processing by the Administrator module, outer meta interface delivers the response to internal modules for further operations. The process of Linker module is shown in Figure 3.

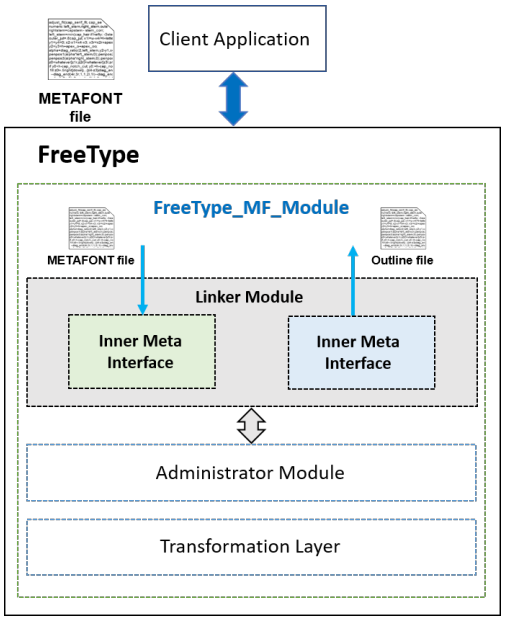


Figure 3: Linker module

3.3 Administrator module

The core functionality of FreeType_MF_Module is performed in the Administrator module. This module is divided into two layers: Search layer and Management layer.

The Search layer is responsible for finding all the installed METAFONT fonts in a table. This table contains a list of all the METAFONT fonts installed and how to fetch information related to them. The Search layer is implemented in Meta scanner and Meta table.

The Management layer mainly performs the following tasks:

1. Checking whether or not the requested font file is METAFONT.
2. Checking the cache to determine if the corresponding outline font for the METAFONT request is already stored. If yes, it sends the response directly from the cache. This functionality is implemented to achieve better performance and reusability.
3. If the outline font is not prepared in the cache, this request is sent to the Transformation layer. The outline font prepared by the Transformation layer is stored in the cache.
4. The response is sent back to FreeType internal modules by the Management layer.

The Management layer is implemented in three parts: Meta analyzer, Meta request, and Meta cache. Figure 4 shows the Administrator module and its sublayers.

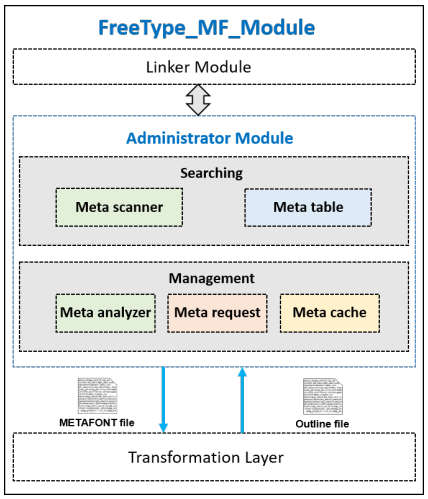


Figure 4: Administrator module

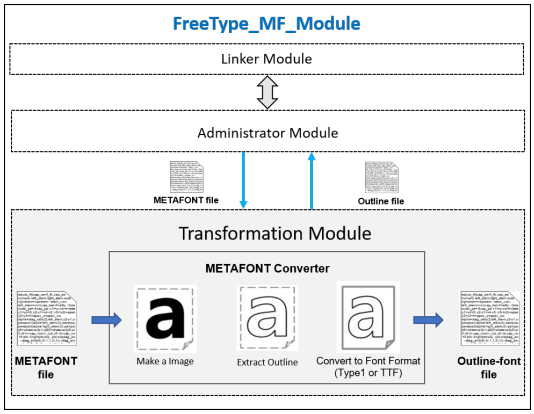


Figure 5: Transformation module

3.4 Transformation module

The Transformation module is mainly responsible for converting the METAFONT file into its corresponding outline font file. If the outline font file for a requested METAFONT file doesn't exist in the table then the Administrator module sends the request to the Transformation module. This module processes the request and returns the corresponding outline font file to the Administrator module. Figure 5 shows how the Transformation module converts METAFONT files into corresponding outline files.

3.5 METAFONT support in FreeType using FreeType_MF_Module

The overall architecture of FreeType_MF_Module is shown in Figure 6. As seen there, FreeType_MF_Module is an internal module of FreeType which is responsible for processing METAFONT file requests.

- First an application sends a font file request to FreeType (step 1).
- If all other driver modules fail to process this font file request, the request is sent to FreeType_MF_Module through the Linker module. Inner meta interface delivers this request to the Administrator module (step 2).
- A Meta request in the Administrator module receives all the information in this font file request and sends it to the Meta Analyzer to check if this font file is METAFONT or not (step 3). If this font file is not METAFONT this request is sent back to FreeType (step 3a). If this request is METAFONT, The Meta analyzer checks if this METAFONT file is installed or not by scanning the Meta table. If not found in the Meta table, an error is sent back to FreeType internal modules (step 3b).
- There can be a scenario in which the METAFONT font is installed but its corresponding outline font is not stored in the cache. In this case, the Meta cache is scanned to check if the corresponding outline file is stored in it (step 4). If it is already stored in the Meta cache with the same style parameters as requested, it is directly sent to FreeType (step 4a). If it is not stored in Meta cache, the request is sent to the Transformation layer (step 4b).
- The Transformation layer converts the METAFONT file into its corresponding outline font by applying the requested style parameters (step 5).
- An outline font is returned from the Transformation module to the Administrator module where the Meta cache is updated for future reuse (step 6).
- The outer Meta interface returns this outline font to core FreeType for further processing (step 7).

Lastly, FreeType renders this outline font that was made from the requested METAFONT with the style parameter values.

The FreeType_MF_Module is perfectly compatible with the standard FreeType rasterizer. FreeType_MF_Module provides direct support of METAFONT in FreeType rasterizer just like its default Type1 driver module, TrueType driver module, etc. The module manages the METAFONT font and its conversion to the corresponding outline font. Client applications can request any style parameters of METAFONT; FreeType_MF_Module processes them and the result is displayed on the screen as usual. As it is directly implemented inside the FreeType rasterizer, it has no dependency problems as discussed in Section 2. FreeType_MF_Module can easily generate multiple font families like bold, italic, and bold-italic depending on the style parameter values passed to it.

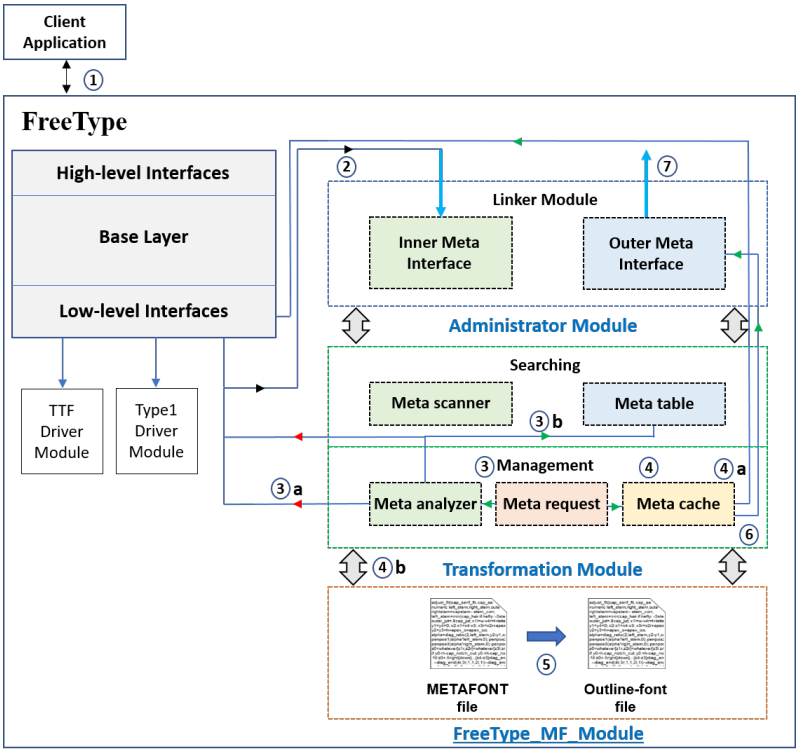


Figure 6: FreeType_MF_Module architecture

Table 1: FreeSerif font family

Style	Styled Output	Font files
Normal	FreeSerif	FreeSerif.ttf
Bold	**FreeSerifBold**	FreeSerifBold.ttf
Italic	*FreeSerifItalic*	FreeSerifItalic.ttf
Bold + Italic	***FreeSerifBoldItalic***	FreeSerifBoldItalic.ttf

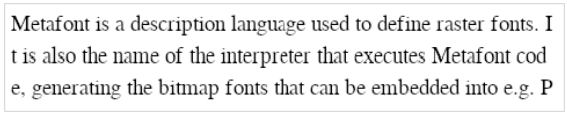
Table 2: Various font styles with Computer Modern

Style	Styled Output	Parameter values
Normal	ComputerModern	Default values of stem, hair, curve, slant
Bold	**ComputerModern**	stem+20, hair+20, curve+20, slant default
Italic	*ComputerModern*	Default values of stem, hair, curve, slant= 0.4
Bold + Italic	***ComputerModern***	stem+20, hair+20, curve+20, slant = 0.4

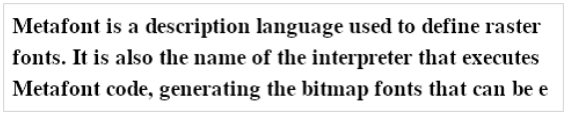
4 Experiment and performance evaluation of FreeType_MF_Module

For the experiment of using FreeType_MF_Module to generate different font styles from METAFONT source, the authors used a font viewer application in GNU/Linux. This application directly uses FreeType to render fonts. It takes a font file and text as input and displays the styled text on the screen using the X Windows System. For testing, the authors have used all four styles of FreeSerif font family as TrueType fonts, i.e., normal, bold, italic, bold-italic, comparing with the Computer Modern fonts in METAFONT.

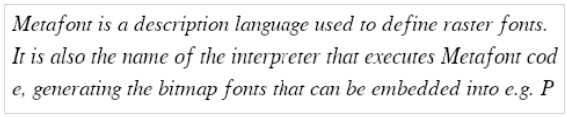
Table 1 shows the FreeSerif font family in four different styles. These styles are generated by using four different font files. Table 2 shows Computer Modern in the same four styles, made using different parameter values. These styles are made from one single METAFONT source file. The parameter values which are modified for generating these font styles are hair, stem, curve, and slant. The three parameters hair, curve, and stem are related with the bold style. Increasing their value increases the boldness of text. These parameter values are different for lowercase and uppercase characters. The slant parameter is related to the italic style. As shown in Table 2, for the normal style the default values of all four parameters are used. For bold style, the values used are stem+20, hair+20, curve+20, and slant parameter default value. Default values of stem, hair, curve, and slant= 0.4 are used for italic style. Whereas,



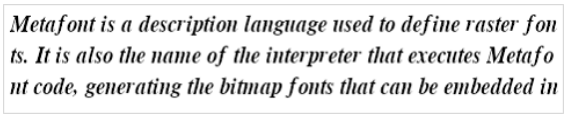
(a) FreeSerif normal style



(b) FreeSerif bold style

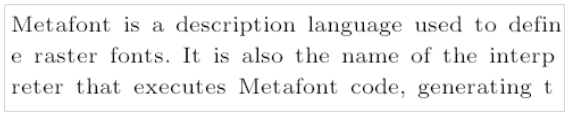


(c) FreeSerif italic style

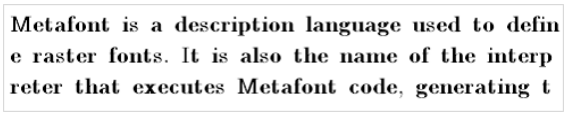


(d) FreeSerif bold-italic style

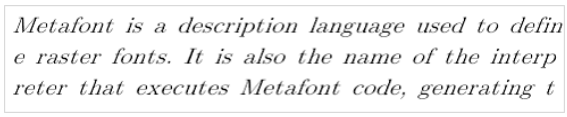
Figure 7: Dataset rendered in FreeSerif (enlarged from screen resolution)



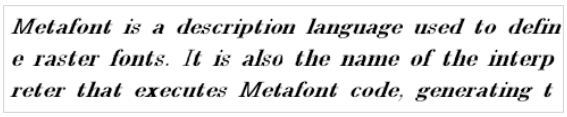
(a) default values of stem, hair, curve, slant



(b) stem+20, hair+20, curve+20, slant default



(c) default values of stem, hair, curve; slant = 0.4



(d) stem+20, hair+20, curve+20, slant = 0.4

Figure 8: Dataset rendered in Computer Modern (enlarged from screen resolution)

stem+20, hair+20, curve+20, slant = 0.4 values are used for bold-italic style. Similarly, many other font styles can be generated with this single METAFONT source file by changing parameter values.

To test the performance of FreeType_MF_Module compared to FreeType default driver modules and the MFCONFIG module, another experiment was performed using the same font viewer application. All four font files of FreeSerif in Table 1 were used for testing the TrueType driver module of FreeType; Computer Modern source files were used with four different parameter values to generate four different styles in Table 2. For the text input, a sample dataset was used which consisted of 2,000 words and over 8,000 characters, including space characters. The average time in milliseconds between the font style request from application and the successful display of styled text on the screen was computed and compared.

Figure 7 shows the result of printing four FreeSerif fonts and Figure 8 shows the result of four Computer Modern METAFONT fonts. Table 3 shows the average time to print the dataset using the FreeSerif font by the TrueType driver module, the Computer Modern font by FreeType_MF_Module, and Computer Modern with the MFCONFIG module. The default TrueType driver module in FreeType takes 3ms to 7ms to print the dataset with all four FreeSerif families. FreeType_MF_Module takes 4ms to 10ms to print this dataset with Computer Modern, whereas the MFCONFIG module took 50ms to 120ms to display a similar size dataset.

The performance of FreeType_MF_Module is comparatively slower than default FreeType driver module because it takes extra time to convert a METAFONT into its corresponding outline font by applying the style parameters. On the other hand, FreeType_MF_Module has very good performance relative to the MFCONFIG module, because it is directly implemented inside the FreeType rasterizer and not dependent on other font libraries like Fontconfig and Xft [9] etc. Hence, we can conclude that FreeType_MF_Module in the FreeType rasterizer can provide direct support of METAFONT usefully in practice, in almost real time on a modern PC.

Performance can be further improved by optimizing the METAFONT converter in the Transformation layer. Currently, the METAFONT converter works with the mftrace and autotrace programs. Future work will consider proposed module optimization and direct usage of TEX bitmap fonts like GF and PK which are not supported by the FreeType rasterizer.

FreeType_MF_Module is a suitable module for providing users with parameterized font support on the screen by applying style parameters directly to the METAFONT font. To reiterate, users don't need to pre-convert METAFONT fonts into outlines before

Table 3: Average time to display dataset with the TrueType driver, FreeType_MF_Module, and MFCONFIG

Style	TrueType driver Avg. Time	FreeType_MF_Module Avg. Time	MFCONFIG Avg. Time
Normal	4.5 ms (3-6)	6 ms (5-8)	70 ms (50-80)
Bold	4 ms (4-6)	7 ms (6-9)	85 ms (70-100)
Italic	4 ms (4-6)	6 ms (4-7)	105 ms (70-110)
Bold + Italic	5 ms (5-7)	8 ms (6-10)	100 ms (90-120)

using with FreeType, as FreeType_MF_Module automatically performs this step. Users see no difference between METAFONT fonts and TrueType fonts using FreeType with this module. FreeType_MF_Module has also overcome the performance and dependency problems of the MFCONFIG module.

5 Conclusion

In this paper, we have proposed a module, named FreeType_MF_Module, enabling direct support of METAFONT in the FreeType rasterizer. Outline fonts such as TrueType and Type 1 do not allow users to easily change font styles. For every different font style in outline fonts, a new font file is created, which can be a time consuming and costly process for CJK fonts consisting of large numbers of complex characters. METAFONT fonts do not have this disadvantage.

FreeType supports many different font formats, including TrueType, Type 1, Windows fonts, etc., but does not provide any support for METAFONT. The proposed module is installed directly inside FreeType and can be used like any other internal module to support METAFONT fonts. The authors have reported on experiments demonstrating that a variety of styled fonts can be generated from METAFONT by adjusting parameter values in a single METAFONT source file by FreeType_MF_Module in almost real time.

Acknowledgement

This work was supported by an Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korean government (MSIT) (No. R0117-17-0001, Technology Development Project for Information, Communication, and Broadcast).

References

[1] Donald E. Knuth, *Computers and Typesetting,* Volume C: *The METAFONTbook*. Addison-Wesley, 1996.

[2] David Turner, Robert Wilhelm, Werner Lemberg, *FreeType*. `freetype.org`

[3] Jinpyung Kim, et al. *Basic Study of Hangul Font,* Seoul: Korea Publishing, Research Institute, 1988.

[4] Jaeyoung Choi, Sungmin Kim, Hojin Lee, Geunho Jeong, *MFCONFIG: A METAFONT plug-in module for the FreeType rasterizer*, *TUGboat* 37:2, pp.163–170, 2016. `tug.org/TUGboat/tb37-2/tb116choi.pdf`

[5] *Fontconfig*. `freedesktop.org/wiki/Software/fontconfig`

[6] Hirotsugu Kakugawa, *VFlib — a general font library that supports multiple font formats,* EuroTEX conference, March 1998. `www-masu.ist.osaka-u.ac.jp/~kakugawa/VFlib/vflib35.ps`

[7] Tomas Rokicki, *The GFtoPK processor Version 2.4,* 06 January 2014. `ctan.org/pkg/mfware`

[8] Rainer Menzner, *A Library for Generating Character Bitmaps from Adobe Type 1 Fonts*. `inferiorproducts.com/docs/userdocs/t1lib/t1lib_doc.pdf`

[9] 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/xft.pdf`

◇ Jaeyoung Choi
Ammar Ul Hassan
Geunho Jeong
369 Sangdo-Ro, Dongjak-Gu
Seoul 06978, Korea
choi@ssu.ac.kr
ammar@ssu.ac.kr
ghjeong@gensolsoft.com