FreeType_MF_Module2: Integration of METAFONT, GF, and PK inside FreeType

Jaeyoung Choi, Saima Majeed,
Ammar Ul Hassan, Geunho Jeong

Abstract

METAFONT is a structured font definition system with the ability to generate variants of different font styles by changing its parameter values. It does not require creating a new font file for every distinct font design. It generates the output fonts such as Generic Font (GF) bitmaps and its relevant TEX Font Metric (TFM) file on demand. These fonts can be utilized on any size of resolution devices without creating a new font file according to the preferred size. However, METAFONT (mf), GF, and Packed Fonts (PK, a compressed form of GF) cannot be utilized beyond the TEX environment as they require additional conversion overhead. Furthermore, existing font engines such as FreeType do not support such fonts.

In this paper, we propose a module for FreeType which not only adds support for METAFONT, but also adds support for GF and PK fonts in the GNU/Linux environment. The proposed module automatically performs the necessary conversions without relying on other libraries. By using the proposed module, users can generate variants of font styles (by mf) and use them on devices of any desired resolution (via GF). The proposed font module reduces the creation time and cost for creating distinct font styles. Furthermore, it reduces the conversion and configuration overhead for TEX-oriented fonts.

1 Introduction

In recent times, technology has developed rapidly. In such environments, there is always a need for better and reliable mediums of communication. Traditionally, fonts were used as a means of communication, replacing pen and paper. A font was originally a collection of small pieces of metal manifesting a particular size and style of a typeface. This traditional technique was eventually replaced by a new concept of digital systems. Modern fonts are implemented as digital data files which contain sets of graphically related characters, symbols, or glyphs. Modern fonts are expected to provide both the letter shape as it is presented on the metal and the typesetter's information on how to set position and replace the character as appropriate.

The ability of science and technology to improve human life is known to us. With the rapid increase in development of science and technology, the world is becoming "smart". People are automatically served by smart devices. In such smart devices, digital fonts are commonly used, rather than analog fonts. A font is the representation of text in a specific style and size; therefore, designers can use font variations to give meaning to their ideas in text. Text is still considered the most common way to communicate and gather information. Although different styles of digital fonts have been created, still they do not meet the requirements of all users, and users cannot alter digital font styles easily [1]. A perfect application for the satisfaction of users' diversified requirements concerning font styles does not exist [2].

Currently, popular digital fonts, whether bitmap or outline, have limits on changing font style [3]. These limitations are removed by another type of fonts, parameterized fonts, e.g., METAFONT, which will be discussed in depth later. METAFONT provides the opportunity to font designers to create different font styles by merely changing parameter values. It generates TEX-oriented font files, namely Generic Font (GF) bitmaps and its equivalent TEX Font Metric (TFM) file. Thus, the usage of METAFONT directly in today's digital environment is not easy, as it is specific to the TEX-oriented environment. Current font engines such as the FreeType rasterizer do not support METAFONT, GF, or Packed Font (PK, a compressed form of GF) files. In order to use METAFONT, GF, or PK files, users have to specifically convert them into equivalent outline fonts.

When METAFONT was created, standard hardware was not fast enough to perform runtime conversion of METAFONT into outline fonts. Therefore, users were not able to take advantage of METAFONT's approach to get different font styles. Today, though, the hardware in typical systems is fast enough to perform such conversions at runtime. If such fonts were supported by the current font engines, the workload of font designers would be reduced, compared to the designers having to create a separate font file for every distinct style. This task of recreation takes considerable time, especially in case of designing CJK (Chinese-Japanese-Korean) characters due to their complex letters and shapes. Therefore, the benefits given by METAFONT can be applied to CJK fonts to produce high quality fonts in an efficient manner.

Our previous work, FreeType_MF_Module [10], has accomplished direct usage of METAFONT, excluding TEX-based bitmap fonts, inside the FreeType rasterizer. But the work was based on external software such as `mftrace` during the internal conversion. Such dependencies have disadvantages related to performance and quality. Hence, the purpose of this research is to present a module inside the FreeType that will directly use METAFONT, GF, and PK font files in a GNU/Linux environment.

In Section 2, the primary objective of this work is discussed. In Section 3, the METAFONT processing with its compiler/interpreter such as the `mf` program is explained. In Section 4, related research regarding the conversion of METAFONT is discussed along with their drawbacks. The implementation of the proposed module is discussed in Section 5. The experiments with the proposed module and performance evaluation along with other modules of the FreeType rasterizer are presented in Section 6. Section 7 gives some concluding remarks.

2 Objective of the research

With the continuing enhancement of technology, typography needs to keep pace. The primary focus of this work is to understand the TEX-oriented bitmap fonts and find ways to utilize them in the GNU/Linux environment using current font engines. Hence, the objective of this research is:

1. To save the time designers require to study the details of each font design from scratch and then create font files for each distinct design.
2. To generate variants of different font styles using a parameterized font system such as METAFONT.
3. To utilize the TEX-based bitmap fonts such as GF, ordinarily specific to the TEX environment, inside the FreeType font engine.
4. To increase the performance by using the compact form of GF, Packed Font (PK).
5. To automatically set the magnification and resolution according to the display.

3 METAFONT processing with the `mf` program

METAFONT, a font system to accompany TEX, was created by D. E. Knuth [4]. It is an organized font definition language which allows designers to change the style of a font per their requirements by changing values of parameters. METAFONT benefits the user in that they do not need to create a different font file for every unique style. It is considered a programming language which contains drawing guidelines for lines and curves which are later interpreted by the interpreter/compiler of METAFONT, notably the `mf` program, to render the glyph definitions into bitmaps and store the bitmaps into a file when done. The `mf` program determines the exact shapes by solving mathematical equations imposed by the author of the METAFONT program.

To process the METAFONT definitions using `mf`, users must understand how to invoke `mf` [5]. Figure 1 shows the proper way of processing the METAFONT using `mf`. (It can accept many other commands.) Therefore, to get the correct GF file, the given settings must be provided: `mode`, `mag`, and the METAFONT file to process. The `mode` setting specifies the printed mode; if this is omitted, a default of `proof` mode will be used, in which METAFONT outputs at a resolution of 2602dpi; this is not usually accompanied by a TFM file. The `mag` setting specifies a magnification factor to apply to the font resolution of the mode. As a result, `mf` generates the bitmap font GF file, its relevant TFM font metric file, and a log file.

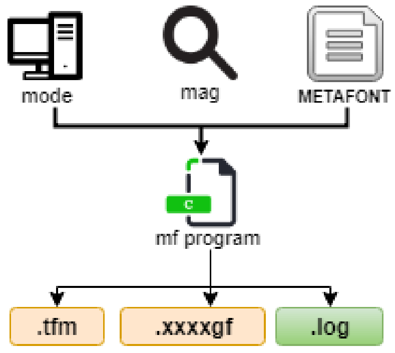


Figure 1: `mf` invocation

For example, if the given mode specifies a resolution of 600dpi, and the magnification is set to 3, the `mf` program will perform calculations internally and generate the output in the form of a GF file at 1800dpi, along with its corresponding TFM and a log file.

Generic Font (GF) format is a TEX-oriented bitmap font generated by the `mf` program by taking a METAFONT program as input along with other information related to the output device. GF font files are generated for a given output device with a specific scaled size. Such font files contain the character shapes in a bitmap form. However, the metric information relevant to the characters is stored in the TEX font metric (TFM) file. To make the GF font usable for typesetting, its corresponding TFM is required, as TEX reads only the font metric file, not the GF. These fonts are utilized in TEX-based typesetting systems.

To view or print, these fonts are converted into device-independent (`.dvi`) files (the same format that is output by TEX). Such a conversion is performed by the utility `gftodvi`. Later, a DVI driver is needed to interpret the `.dvi` file. In order to preview, a utility such as `xdvi` (for Unix systems) is utilized.

The Packed Font (PK) format is also a bitmap font format utilized in the TEX typesetting system. It is obtained by compressing the GF font; the size of a PK is usually about half of its GF counterpart. The content of a PK file is equivalent to a GF. The file format is intended to be easy to read and interpreted by the device drivers. It reduces the overhead of loading the font into memory. Due to its compressed nature, it reduces the memory requirements for those drivers that load and store each font file into memory. PK files are also easier to convert into a raster representation. This also makes it easy for a driver to skip a particular character quickly if it knows that the character is unused.

4 Related work

4.1 Existing font systems

VFlib [6] is a virtual font system that can handle a variety of font formats, e.g., TrueType, Type 1, and TeX bitmap fonts. It does not support METAFONT fonts directly. It provides a software library and a database font file which defines the implicit and explicit fonts. Although it supports different font formats, for some fonts it makes use of external libraries, as shown in Figure 2. The font searching mechanism utilized in VFlib is time consuming if the font does not appear in its database. Therefore, to handle such fonts, various font drivers are called to check whether the requested font can be opened or not. Hence, this font system is not suitable for adding METAFONT support because of the extra dependencies and need for database updates.

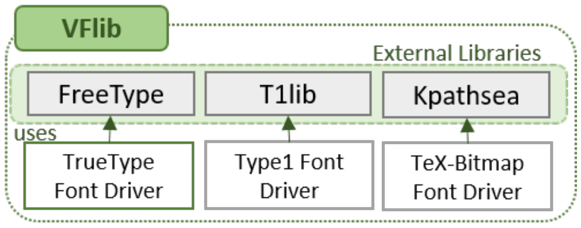


Figure 2: VFlib library dependencies

An alternative is the FreeType [7] font rasterizer. It has the ability to handle different font styles regardless of platform, unlike the T1lib [8] font rasterizer. It does not support the TeX-oriented bitmap fonts and METAFONT fonts, but it provides intuitive interfaces to allow users to add new font modules to enhance the functionality of the engine. Therefore, the FreeType font engine is the best choice for adding the TeX-oriented bitmap fonts because it has no dependency and database issues. If there is a module inside FreeType which supports the TeX-oriented bitmap fonts such as GF and PK, then users can take advantage of these fonts, which are normally specific to the TeX environment. No pre-conversion by utilizing DVI drivers will be required to preview TeX-oriented fonts.

4.2 Research on adding METAFONT support in existing font systems

As mentioned in Section 4.1, the FreeType font engine provides the capability of adding new font modules. MFCONFIG [2] adds indirect support for METAFONT inside FreeType. It provides an intuitive way to use METAFONT in the GNU/Linux environment. As shown in Figure 3, it allows users to utilize METAFONT fonts, but has some dependency problems in that it is built on the high-level font libraries Fontconfig [9] and Xft. These dependencies affect the performance of the module compared to the built-in font driver modules of FreeType. Also, it is unable to handle the TeX-oriented bitmap fonts such as GF and PK, and adding support for the TeX bitmap fonts would be inadequate as it's not directly implemented inside FreeType.

FreeType_MF_Module [10], a METAFONT module inside the FreeType font engine, resolves the dependency and performance issues which were seen in MFCONFIG. Its performance is much faster than MFCONFIG as it is implemented inside FreeType. Using METAFONT fonts requires transformation into an outline font. Hence, FreeType_MF_Module performs this conversion, relying on `mftrace`. Although this generates high-quality output, during conversion font file information is lost due to the reliance on `mftrace`.

As shown in Figure 4, when the request for a METAFONT font is received by FreeType, it sends it to FreeType_MF_Module. In its sub-module Transformation Module, it calls `mftrace`, which has its own drawbacks. It was specifically designed for translating METAFONT to Type 1 or TrueType formats by internally utilizing the `autotrace` and `potrace` libraries for conversion of bitmaps into vector fonts. This approximate conversion gives an approximate outline, and loses information about nodes and other control points [11]. Also, it processes the METAFONT font but is unable to process TeX-based GF and PK bitmap fonts. Therefore, to add support for GF and PK inside FreeType_MF_Module is inconvenient due to the dependency on the external libraries, which also decreases the performance of the module.

The proposed FreeType_MF_Module2 is intended to resolve the problems of FreeType_MF_Module, and is able to support TeX bitmap fonts along with METAFONT. The module can process METAFONT and GF independently without relying on any external software, e.g., `mftrace`. It can be easily installed

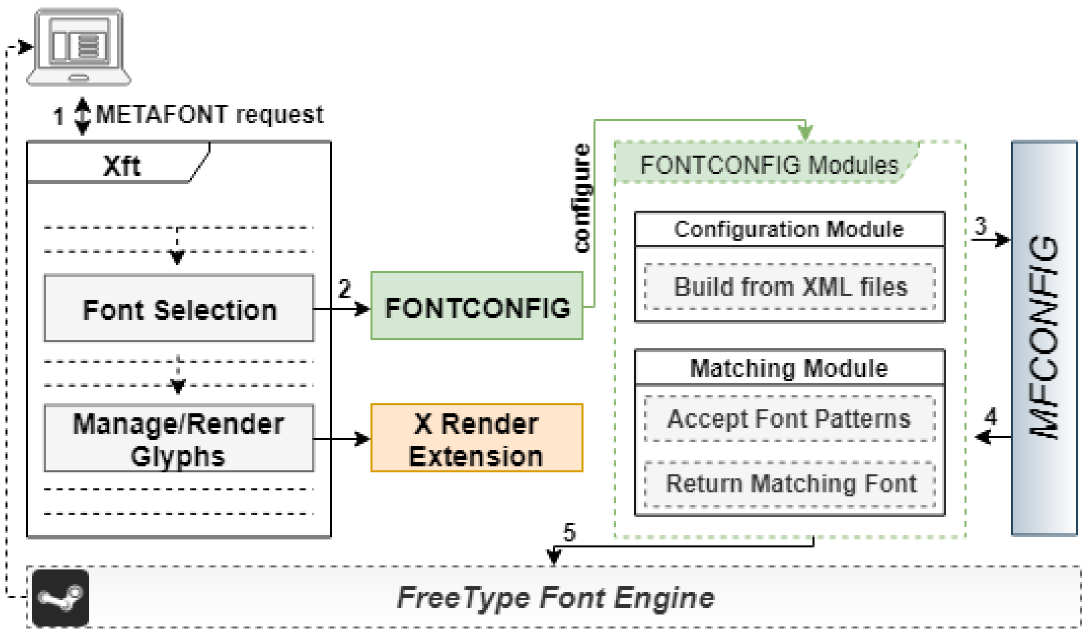


Figure 3: MFCONFIG internal architecture

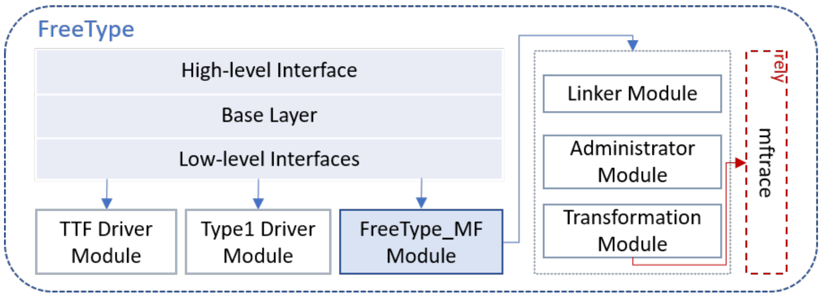


Figure 4: FreeType_MF_Module architecture

and removed, as it is implemented just like the default FreeType driver module. Therefore, METAFONT and TEX-oriented bitmap fonts can be used just like any existing digital font format using the proposed module.

5 Implementation of the module

To use digital fonts, FreeType is a powerful library to render text on screen. It is capable of producing high quality glyph images of bitmap and outline font formats. When FreeType receives a request for a font from the client application, it sends the font file to the corresponding driver module for the necessary manipulation. Otherwise, it displays an error message to the client that the requested font file is not supported. So, the proposed module is directly installed inside FreeType to process requests for METAFONT, GF, and PK fonts. As shown in Figure 5, when FreeType receives a request for one of these formats, it is sent on to FreeType_MF_Module2.

As shown in Figure 6, the MF Script module calls its submodule Font Style Extractor. This extracts the font style parameters from the METAFONT file. For example, if the METAFONT request given to the module has the italic style, this will extract the italic style parameters from the METAFONT file and apply them. Once it extracts the font style parameters, the corresponding outline will be generated, with the requested style, by utilizing the Vectorization submodule.

5.1 METAFONT (`mf`) request

When FreeType sends a METAFONT request to the proposed FreeType_MF_Module2, its submodule Request Analyzer API analyzes the font file to determine whether the requested is for a usable METAFONT file or an incorrect one, by analyzing its style parameters. After analyzing, it checks whether the requested font has already been manipulated by the font driver or if the new request has arrived via the Cache (again,

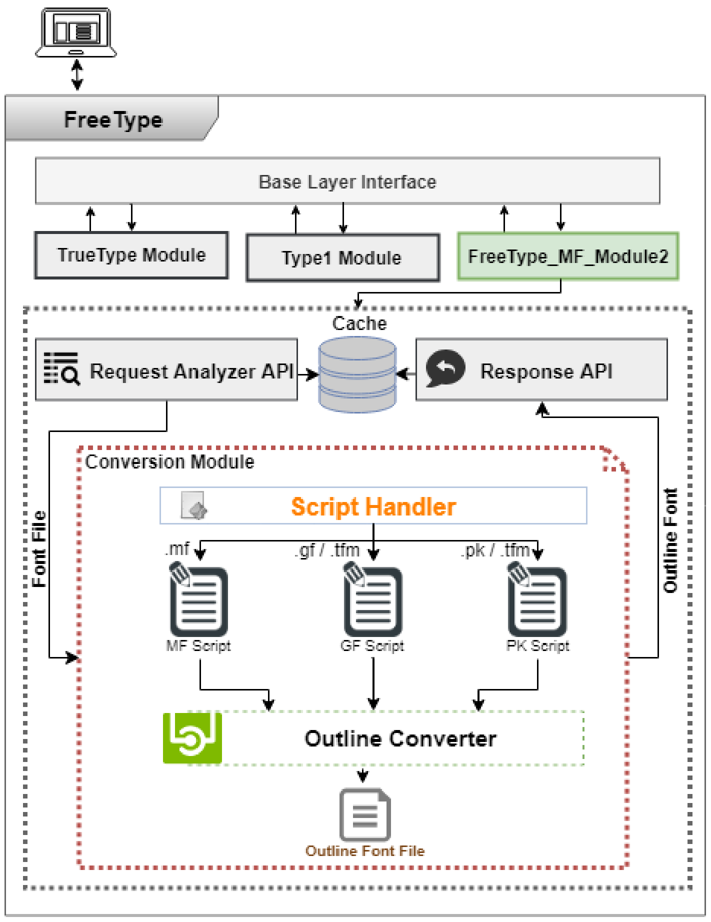


Figure 5: FreeType_MF_Module2 architecture

see Figure 5). If the requested font is found in the Cache, it is sent directly back to FreeType for manipulation. But if the font is not found in the Cache, it sends the METAFONT request to the Conversion Module. After receiving the request, this utilizes a submodule Script Handler. The core functionality of the module is performed in this module. It calls the scripting module based on the request. For a METAFONT request, it calls the MF Script module, passing the METAFONT file.

After extracting the character outlines, it is necessary to remove redundant nodes from the shapes to improve the quality. Therefore, a Node Redundancy Analysis step receives the transformed METAFONT, analyzes the outline contours, and removes the redundant nodes from the font to create the simplified outline. Once the simplification task is done, autohinting is performed on the font with the Hinting Module. After hinting, the corresponding outline font will be generated with the Outline Converter module and the outline font file sent to the module Response API. This updates the Cache with

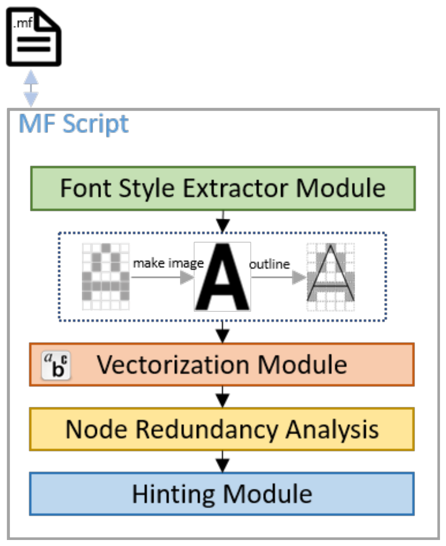


Figure 6: MF Script internal architecture

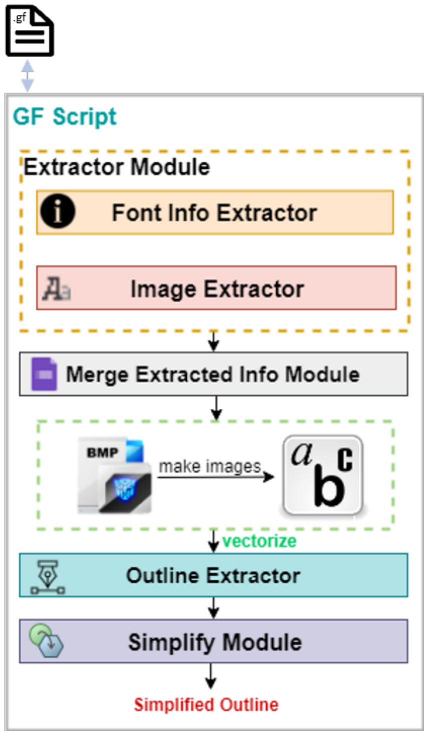


Figure 7: GF Script internal architecture

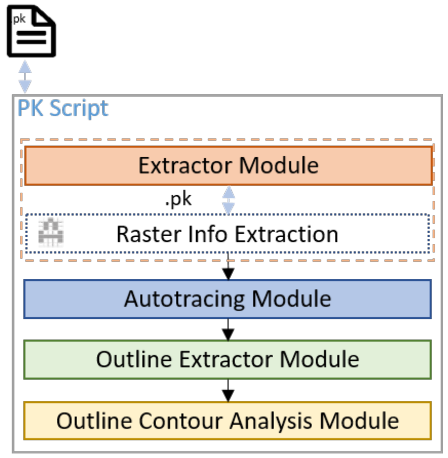


Figure 8: PK Script internal architecture

the newly generated outline font for reusability and high performance. After updating, FreeType renders this outline font that was created from the METAFONT with the requested style parameter values.

5.2 Generic Font (GF) request

When FreeType sends a GF request to the proposed module, again, the requested font goes first to the Request Analyzer API module. This checks whether the requested GF font has been converted with correct use of the `mf` compiler by analyzing the device specific information. If the requested GF file was not generated correctly, the Request Analyzer API module will not proceed, as it has to compute file names using the device resolution and magnification font parameters. On the other hand, if the GF font is generated by correct use of `mf`, then its TEX font metric file must exist.

For a GF request, its corresponding TFM must be provided for internal computations related to character shapes. (Similarly, TEX only reads the TFM instead of GF as all the relevant information is provided by the TFM). After the Request Analyzer API module analyzes the GF request, it checks in the Cache to see if the manipulated font exists. If the requested font does not exist in the Cache, the request is forwarded to the Conversion Module where the Script Handler submodule handles the GF request along with its companion TFM file. As shown in Figure 7, when GF Script receives the GF file, its submodule Extractor Module contains the main functionality. Its internal module Font Info Extractor extracts the font-related information from the TEX font metric file as well as a sequence of bitmaps at a specified resolution from the GF file.

After extraction, it merges the extracted information and makes the GF file usable in the form of character images via Merge Extracted Info module. From the bitmap font, it makes character images. After merging and creating the vector images, it extracts the outline of the characters via the Outline Extractor module. After extracting the outline, it sends the extracted outline characters to the Simplify module, which, as described above, analyzes the font and removes the redundant nodes from the font to make high quality outlines. It then outputs the simplified outline using the Outline Converter module internally. The newly created outline font is sent to the Response API module, which updates the Cache with the generated outline font for later reusability. Once the Cache is updated, it sends back the response to the core FreeType module for further processing. Lastly, FreeType renders this outline font that was designed from the requested GF with the styled parameter values at a specified resolution.

5.3 Packed Font (PK) request

A PK font request is handled with the same process as described in Sections 5.1 and 5.2, up until the Conversion Module. Once the Script Handler receives the requested PK font, it passes control to PK Script. As shown in Figure 8, the Extractor module extracts the raster information from the packed file. After extraction, it performs autotracing on the merged font via Autotracing Module, which outputs the character images. The Autotracing Module not only uses an autotracing program, it improves the basic result with additional functionality such as auto-hinting and eliminating redundant nodes from the font image. These enhancements result in high quality output. Once done, it sends the transformed output to the Outline Extractor Module where it obtains the outline of the characters. After getting the outline character images, it performs the outline contour analysis and removes the redundant nodes from the outlines using the submodule Outline Contour Analysis. As before, it sends the simplified output to the Outline Converter, and the generated outline font file is sent to the Response API which updates the Cache and sends to the corresponding FreeType module for rendering.

The proposed module provides direct support for METAFONT, GF, and PK. It is perfectly compatible with FreeType's default module drivers. It can manipulate the request with the desired style parameters and scale size. As a result, it provides better quality outline fonts without needing external libraries.

6 Experiments and performance evaluation

To test the proposed module, an application server is utilized. The application server is responsible for rendering the text on the screen by receiving the font file from FreeType along with the text requested to be displayed. FreeType can only process those fonts in formats which it supports. When the client application sends the METAFONT, GF, or PK request to FreeType, it internally processes the requested font using the proposed module and sends the newly generated outline font file, along with the input text, to the application server to display on screen.

For testing purposes, the METAFONT font Computer Modern is used. The Computer Modern fonts are examined with the usual four styles: Normal, Italic, Bold, and Bold+Italic. (We chose to use the slanted roman instead of the cursive italic styles, due to resolution considerations.) These styles are generated by changing the METAFONT parameters. To verify the quality of the proposed module results, the authors used the same four styles of another font family, FreeSerif. The sample text is composed of words and characters, including the space character.

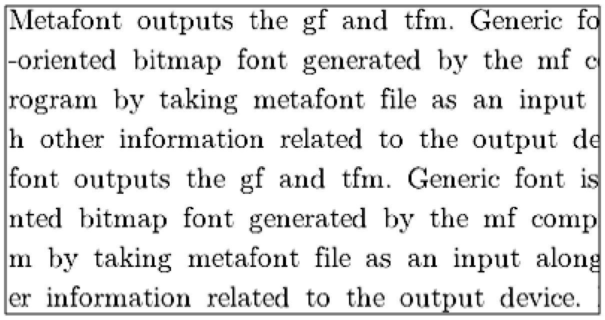
The same font family was used to test the original FreeType_MF_Module, with the same four font styles. Thus, changing the parameter values and generating new styles are explained in [10]. The same concept is applied to the proposed module for experiments. The only difference comes in the cases of GF and PK fonts. To manipulate such fonts, information about the printer device and resolution is required. In the proposed module, the GF and PK fonts are directly manipulated by the module without requiring any DVI driver or previewer. It accepts the input text by the client application and internally calculates the font resolution in pixels per inch. Afterwards, it internally processes the GF and PK file as described in Sections 5.2 and 5.3 respectively, and generates the necessary output with the desired style.

When FreeType sends the METAFONT request to the proposed module, it internally manipulates the request by extracting the styled parameters from the source file. The default style of Computer Modern METAFONT is generated by extracting the default parameters. The four font styles Normal, Bold, Italic, and Bold+Italic are generated by the module, and it generates output similar to that shown in Figure 9(a–d), respectively. Using one Computer Modern METAFONT file, different font styles can be generated according to desires and requirements.

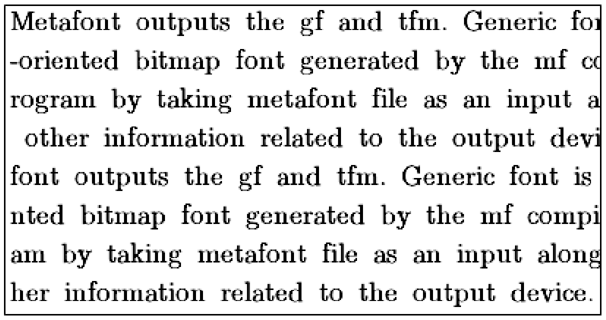
When FreeType receives a Generic Font request from the client application server, it sends it to the proposed module along with the input text, where it extracts the font-related information from the TFM file and resolution information from the GF file. Then it internally calculates the font resolution in pixels per inch by referring to a device definition. Later, it generates the output for the resulting resolution, as shown in Figure 9. The default style of Generic Font is generated by extracting the default style parameters at 1200dpi. The remaining font styles such as Bold, Italic, and Bold+Italic are generated by the module at the calculated resolution, with results as shown in Figure 9(a–d), respectively. The GF results differ from METAFONT slightly, due to the variations in the resolution — the authors tested the GF font with different magnifications at the time of manipulation.

The GF font created by METAFONT has a rather large size which takes considerable memory during the manipulation. To reduce memory consumption, it is converted into packed form using the utility `gftopk`. PK files contain exactly the same information and style parameters as the GF files. Therefore, their resulting output differs only in performance, rather than quality; again, Figure 9 shows the results.

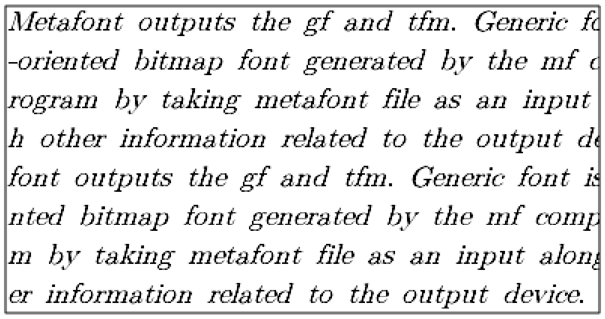
The authors compared the obtained results with the first FreeType_MF_Module. It is concluded that the results are quite similar and the proposed module handles the TEX-oriented bitmap fonts along with METAFONT format inside FreeType, without reliance on external software for the conversions.


Metafont outputs the gf and tfm. Generic fo
-oriented bitmap font generated by the mf c
rogram by taking metafont file as an input
h other information related to the output de
font outputs the gf and tfm. Generic font is
nted bitmap font generated by the mf comp
m by taking metafont file as an input along
er information related to the output device.

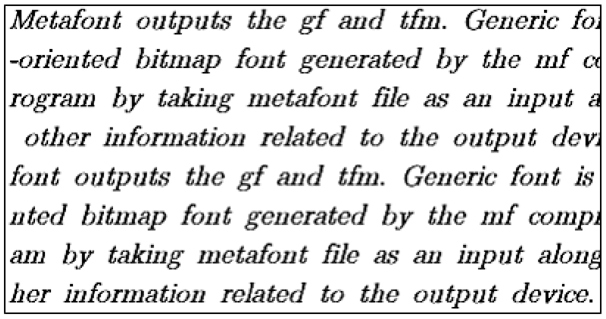
(a) Normal style Packed Font output


Metafont outputs the gf and tfm. Generic fo
-oriented bitmap font generated by the mf co
rogram by taking metafont file as an input a
other information related to the output devi
font outputs the gf and tfm. Generic font is
nted bitmap font generated by the mf compi
am by taking metafont file as an input along
her information related to the output device.

(b) Bold style Packed Font output


Metafont outputs the gf and tfm. Generic fo
-oriented bitmap font generated by the mf c
rogram by taking metafont file as an input
h other information related to the output de
font outputs the gf and tfm. Generic font is
nted bitmap font generated by the mf comp
m by taking metafont file as an input along
er information related to the output device.

(c) Slanted style Packed Font output


Metafont outputs the gf and tfm. Generic fo
-oriented bitmap font generated by the mf co
rogram by taking metafont file as an input a
other information related to the output devi
font outputs the gf and tfm. Generic font is
nted bitmap font generated by the mf compi
am by taking metafont file as an input along
her information related to the output device.

(d) Bold-Slanted style Packed Font output

Figure 9: Text printed with Packed Font (PK) format

Table 1: Average time of rendering (in milliseconds)

Style	Time efficiency of font modules (Average Time)				
	TrueType Font Driver	FreeType_MF_ Module	FreeType_MF_Module2		
			METAFONT	GF	PK
Normal	4.5 ms (3-6)	6 ms	5 ms	6 ms	4 ms
Bold	4 ms (4-6)	7 ms	6 ms	6 ms	6 ms
Slanted	4 ms	6 ms	6 ms	5 ms	4 ms
Bold + Slanted	5 ms (5-7)	8ms	8 ms	7 ms	6 ms

The authors have not only considered the quality of the generated font using the proposed module, but also performance. As shown in Table 1, the performance of FreeType_MF_Module is slightly slower in processing the Bold and Bold+Italic font styles of METAFONT. This takes time due to the dependency on the external software such as `mftrace`. Therefore, the proposed module overcomes such performance and dependency issues by adding the functionality integrating the bitmap font formats. GF fonts take a little more time compared to PK, but less time than METAFONT, as it is already in a compiled form. PK fonts take less time than either METAFONT or GF, as it is the compressed form of GF.

The proposed FreeType_MF_Module2 provides parameterized font support. The proposed module does not require any preconversion before submitting such fonts to the FreeType rasterizer. Client applications which utilize FreeType can thus now also utilize the TEX-oriented bitmap font formats GF and PK, as well as METAFONT fonts, using the proposed module. Such fonts can be used just as TrueType or other font formats supported by FreeType, with similar performance. The proposed module can be utilized in the FreeType font engine as a default driver module. The proposed module works in the same fashion as the other driver modules in FreeType. It is able to support real-time conversion in a modern GNU/Linux environment.

7 Conclusion

In this paper, a new module is proposed for the FreeType font rasterizer which enhances its functionality by adding support for TEX-oriented parameterized (METAFONT) and bitmap (GF and PK) fonts. FreeType supports many font formats, but not these, which originated in the TEX environment.

Although our recent studies provided a way to utilize METAFONT fonts inside FreeType, it had dependency issues which affected the performance of the module. Furthermore, it could only handle METAFONT requests. The proposed module overcomes these issues and adds TEX-oriented bitmap font support as well. With the proposed module, users can use METAFONT, GF, and PK fonts without needing other drivers for conversion. Therefore, with the proposed module, users can now utilize these fonts outside the TEX environment.

Furthermore, the proposed module overcomes the disadvantage of outline fonts requiring users to change font styles using only existing font files, thus requiring a different font file to be created for every distinct font style and size. Creating a new outline font file for CJK fonts consumes significant time and cost, as they have rather complicated shapes compared to alphabet-based fonts. Various studies have been conducted to implement CJK fonts, such as Hongzi [14] and the use of a structural font generator using METAFONT for Korean and Chinese [15]. It might take a longer time to process CJK METAFONT fonts, which have complicated shapes and several thousands of phonemes. The proposed module optimization and utilization for the CJK fonts will be considered in the future.

Acknowledgement

This work was supported by an Institute of Information & Communications Technology Planning and Evaluation (IITP) grant funded by the government of Korea (MSIP) (No. 2016-0-00166, Technology Development Project for Information, Communication, and Broadcast).

References

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

[2] J. Choi, S. Kim, H. Lee, G. Jeong. MFCONFIG: A METAFONT plug-in module for FreeType rasterizer. *TUGboat* 37(2):163–170 (TUG 2016 conference proceedings). `tug.org/TUGboat/tb37-2/tb116choi.pdf`

[3] 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`

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

[5] Web2c: A TEX implementation. `tug.org/web2c`

[6] H. Kakugawa, M. Nishikimi, N. Takahashi, S. Tomura, 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`

[7] D. Turner, R. Wilhelm, W. Lemberg. *FreeType.* `freetype.org`

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

[9] K. 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`

[10] J. Choi, A. Hassan, G. Jeong. FreeType_MF_Module: A module for using METAFONT directly inside the FreeType rasterizer. *TUGboat* 39(2):163–170 (TUG 2018 conference proceedings). `tug.org/TUGboat/tb39-2/tb122choi-freetype.pdf`

[11] H.-W. Nienhuys. `mftrace` — Scalable fonts for METAFONT. 2017. `lilypond.org/mftrace`

[12] M. Weber. Autotrace — converts bitmap to vector graphics. 2002. `autotrace.sourceforge.net`

[13] K. Píška. Creating Type 1 fonts from METAFONT sources: Comparison of tools, techniques and results. Preprints for the 2004 Annual TUG Meeting. `tug.org/TUGboat/tb25-0/piska.pdf`

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

[15] J. Choi, G. Gwon, M. Son, G. Jeong. Next Generation CJK Font Technology Using the Metafont. LetterSeed 15:87–101, Korea Society of Typography, 2017.

◇ Jaeyoung Choi
Saima Majeed
Ammar Ul Hassan
Geunho Jeong
369 Sangdo-Ro, Dongjak-Gu
Seoul 06978, Korea
`choi (at) ssu.ac.kr`
`saimamajeed089 (at) gmail.com`
`ammar (at) ssu.ac.kr`
`ghjeong (at) gensolsoft.com`