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

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

Abstract

METAFONT is the structured font definition that has the ability to generate variants of different font styles by changing its parameter values. It doesn't require to create a new font file for every distinct font design. It generates the output fonts such as Generic Font (GF) and its relevant TeX Font Metric (TFM) file on demand. These fonts can be utilized on any size of the resolution devices without creating new font file according to the preferred size. However, METAFONT (MF), Generic Fonts (GF), and Packed Fonts (PK compressed form of GF) cannot be utilized beyond the TeX environment as it requires the additional conversion overhead. Furthermore, existing font engine such as FreeType doesn't support such fonts.

In this paper, we have proposed a module for FreeType which not only adds the support of METAFONT, but also adds the support of GF and PK font under Linux environment. The proposed module automatically perform such conversions without relying on other libraries. By using the proposed module, users can generate variants of font styles (by MF) and use it on the desired resolution devices (by GF). The proposed font module reduces the creation time and cost for creating the distinct fonts styles. Furthermore, it reduces the conversion and configuration overhead for TeX-oriented fonts.

1 Introduction

In the recent era, development in technology is increasing rapidly. In such environments, there is always a need of better and reliable medium for communication. Traditionally, fonts were used as means of communication. A font was collection of small pieces of metal which has particular size and style of the typeface. With the enhancement, modern fonts were introduced which were expected to sum up both the letter shape as it is presented on the metal and the ability of the typesetter by providing information that how to set position and replace the character as appropriate. Such technique was not reliable as the concept of pen and paper was considered the slow and inefficient way of communication. This traditional technique was replaced by the modern fonts. A new concept of digital systems aroused where these modern fonts are utilized which replaced the pen and paper usage. Therefore, modern font is implemented as digital data file which contains set of graphically related characters, symbols or glyphs.

The ability of science and technology to improve human life is known to us. With the rapid increase in development of science and technology, world is becoming "smart". People will automatically be served by smart devices. In such smart devices, digital fonts are commonly used than analog fonts. As font is the representation of text in a specific style and size, therefore, designers can use various font setting to give meaning to their ideas in text. Text is still considered the most appropriate and an elective source to communicate and gather information, respectively. Although a different styles of digital fonts have been created but still they do not meet the requirements of all the 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, either bitmap or outline, have limits on changing font style [3]. These limitations are removed by another type of fonts such as parameterized fonts e.g. METAFONT which will be discussed later in depth. METAFONT provides the opportunity to the font designers to create a different font styles by just changing some of its parameter values. It generates the TeX-oriented bitmap font such as Generic Font (GF) and its equivalent TeX Font Metric (TFM) file. However, the usage of METAFONT directly in the digital environment is not easy as its specific to TeX oriented environment and the current font engines, the FreeType rasterizer doesn't support the METAFONT, Generic Font (GF), and Packed Font (PK). In order to use the METAFONT, GF, and PK font, users have to specifically convert them into its equivalent outline font. When METAFONT was created, the hardware of the PCs was not fast enough to perform the runtime conversion of METAFONT into outline font. Therefore, users are not able to get advantage from the METAFONT to get different font styles. Currently, the hardware which are being utilized in system are fast enough to perform such conversions on runtime. If such fonts will be supported by the current font engines, then the workload of the font designers will be reduced. As the font designers have to create a separate font file for every distinct style. Such recreation task is time taken especially in case of designing the CJK (Chinese-Japanese-Korean) characters due to its complex letters and shape. Therefore, such benefits given by METAFONT can be applied to the CJK font to produce high quality font in an efficient manner. Our previous work, FreeType_MF_Module[10] have been accomplished for the direct usage of METAFONT excluding TeX-based bitmap fonts, inside FreeType rasterizer. But the work was somehow based on the external library such as mftrace during the internal conversion. Therefore, such library has disadvantages related to the performance and quality. Hence, the purpose of this research is to present a module inside the FreeType that will directly use the METAFONT, GF, and PK font in 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 mf program are explained. In Section 4, the 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 on the proposed module and performance evaluation along with other modules of FreeType rasterizer is presented in Section 6. Section 7, describes the concluding remarks.

2 Objective of the Research

With the enhancement in development and technology, typography also get the fame. The primary focus of this work is to understand the digital fonts, TeX-oriented bitmap fonts and find out the ways how to utilize it in Linux environment using the current font engines. Hence, the objective of this research is:

1. To save time of the designer to study the details of each font design from scratch and then create font file for every distinct design
2. To generate variants of different font styles using parametrized font such as METAFONT
3. To utilize the TeX-based bitmap fonts such as GF which is specific to TeX environment inside Freetype font engine
4. To increase the performance by using compact form of GF such as Packed Font (PK)
5. To set the automatic magnification and resolution according to the display in case of Generic Font

3 METAFONT processing with mf program

METAFONT, a TEX font system, had been introduced by D. E. Knuth [4] is an organized font definition which allows the font designers to change the style of font as per their requirements by changing values of parameters. METAFONT benefits the user in a way that they don't need to write the different font file for every unique style. It is considered as programming language which contain lines and curves drawing guidelines which are later interpreted by the interpreter/compiler of METAFONT such as mf program to draw the glyphs into a bitmaps and keeping the bitmaps into a file when done. Mf program determines the exact shapes by solving mathematical equations imposed by METAFONT. To process the METAFONT using mf program, users must have the knowledge of mf invocations [5]. Figure 1 shows the proper way of processing the METAFONT using mf program. It can accept plenty of other commands. Therefore, in order to get the correct GF file, these commands are provided e.g. mode, mag, and METAFONT file to process. The mode command specify the printed mode, if leave this out the default will be used such as proof mode where METAFONT will outputs at a resolution of 2602dpi; this is not usually required without TFM. The mag command takes the font resolution in pixels per inch along with the METAFONT file. In result, mf program generates the TeX-oriented bitmap font file such as GF, its relevant Font metric file named: TFM, and log file.

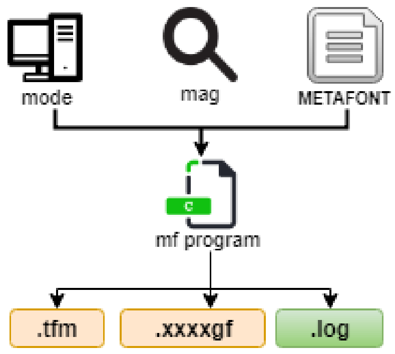


Figure 1: mf invocations

For example, if the device is 600dpi and specify the magnification 3 along with mode then mf program will perform calculations internally and will generate the output in the form of GF at 1800dpi, along with its corresponding tfm and log file.

Generic Font (GF) is TeX-oriented bitmap font generated by the mf program by taking METAFONT as an input along with other information related to the output device. GF font files are generated for each output device with specific scaled sizes. Such font files contain the character shapes in a bitmap form. However, the information relevant to the characters shape are stored in the TeX font metric (TFM) file. To give meaning to the GF font, its corresponding TFM is required as TeX only reads the font metric file instead of GF. These fonts are utilized in TeX typesetting systems. To view or print, these fonts are converted into device-independent (.dvi) files. Later, DVI drivers are required to interpret the data given in device independent files as .dvi files cannot be read directly by the TeX. Such conversions are performed by the utility named; gftodvi. It reads binary generic font and convert them into device-independent files. In order to preview, utility named xdvi is being utilized. As GF files are unreadable, therefore, such conversions are required in order to view.

The Packed Font (PK) is bitmap font format utilized in the TeX typesetting systems. It can be obtained by compressing the GF font. As GF files are larger in size, therefore, the size of the PK is half of their GF counterparts. The content stored in PK files are same as GF. Such file format is intended to be easy to read and interpreted by the device drivers. It reduces the overhead of loading the font on memory. Due to its compression nature, it reduces the memory requirements for those drivers that loads and stores the each font file on memory. They are also easier to convert into a raster representation. (This also makes it possible for a driver to skip a particular character quickly if it knows that the character is unused).

4 Related Works

4.1 Existing Font Systems

VFlib [6] is a virtual font system that can handle the variety of font formats e.g. TrueType, Type1, and TeX-bitmap fonts. It handles the library itself and the database font file where it defines the implicit and explicit fonts. Although it supports different font formats but for some fonts it make use of the external libraries, as shown in Figure 2. Furthermore, it doesn't support the METAFONT but it has the ability to handle the TeX-bitmap fonts. The font searching mechanism utilized in VFlib is time consuming, if the font doesn't appear in the database. Therefore, to handle such fonts, various font drivers will be called to check whether the requested font can be opened or not. Hence, such font systems are not suitable to add the METAFONT support because of reliance and taking care of database.

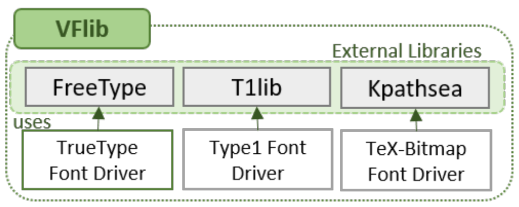


Figure 2: VFlib Reliance

An alternative to such font engines is FreeType [7] font rasterizer. It has the ability to handle different font styles regardless of platform dependency unlike T1lib [8] font rasterizer. However, it doesn't support the TeX-oriented bitmap fonts and METAFONT. But it provides the intuitive interfaces which allows the end-users to add the new font module to enhance the functionality of the engine. Therefore, selection of 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 will support the TeX-oriented bitmap fonts such as GF and PK, then, users can get advantage of such fonts that are only specific to TeX-environment. No pre-conversion by utilizing the DVI drivers will be required to preview TeX-oriented fonts.

4.2 Researches on adding METAFONT support in existing font systems

As mentioned in Section 4.1, FreeType font engine provides the capability to add the new font module. MFCONFIG [2] added an indirect support of METAFONT inside FreeType. It provides an intuitive way to use METAFONT on Linux environment. As shown in Figure 3, it allows the users to utilize the METAFONT but it has some dependency problem as it is built on high-level font libraries such as FONTCONFIG [9] and Xft. Due to such dependencies it affects the performance of the module compared to font driver modules of FreeType. It is unable to handle the TeX-oriented bitmap fonts such as GF and PK. Therefore, adding the functionality of TeX-bitmap fonts is 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 stimulated in MFCONFIG. Its performance is relatively faster than MFCONFIG as it is implemented inside the FreeType. In order to use the METAFONT, it requires to transform it into outline font. Hence, FreeType_MF_Module performs such conversions but relying on mftrace. Although, it generates a high-quality output but during conversion font file information is vanished due to reliance on mftrace. As shown in Figure 4, when the request of METAFONT is received by the FreeType, it sends it to FreeType_MF_Module. When it comes to its submodule named: Transformation Module, it utilizes mftrace. Mftrace has its own drawbacks. It was specifically designed for translating METAFONT fonts to Type1 or TrueType formats by internally utilizing the autotrace and potrace libraries for vectorization purpose. Approximate conversion gives approximate outline and lost information about nodes and other control points [11]. Although, it processes

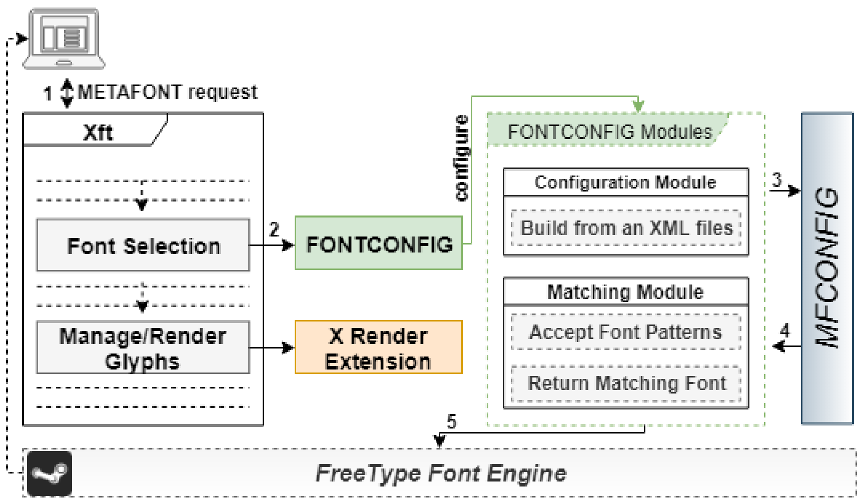


Figure 3: MFCONFIG Internal Architecture

the METAFONT but is unable to process TeX-based bitmap fonts such as Packed Font (PK) and Generic Fonts (GF). Therefore, to add a support of GF or PK inside FreeType_MF_Module is inconvenient due to dependency on external library which slower down the performance of the module.

The proposed FreeType_MF_Module2 intends 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 external library e.g. mftrace. It can be easily installed and removed, as it is implemented just like the default FreeType driver module. Therefore, METAFONT and TeX-oriented bitmap fonts can be used as the existing digital font formats using the proposed module.

5 Implementation of the Module

To use the digital fonts, FreeType is a powerful library to render the text on screen. It is capable of producing the high quality glyph images of the bitmap and outline font formats. When FreeType receives a request of font from the client application, it sends the font file to the responsible module driver for the manipulation. Otherwise, it displays an error message to the client if the requested font file is not supported. Similarly, the proposed module is directly installed inside the FreeType to process the request of METAFONT and TeX bitmap font such as Generic Font (GF) and Packed Font (PK). As shown in Figure 5, when FreeType receives the METAFONT or GF request it directs into FreeType_MF_Module2.

5.1 METAFONT (MF) Request

When FreeType sends the METAFONT request to FreeType_MF_Module2, its submodule Request Analyzer API analyzes the font file. It analyzes that the requested font file is the exact METAFONT file or the wrong one by analyzing its style parameters. After analyzing, it checks whether the requested font is already manipulated by the font driver or the new request is arrived via Cache. If the requested font is found in the Cache, it sends directly to the engine 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, it utilizes its submodule named: Script Handler. The core functionality of the module is performed in this module. It calls the scripting module based on the request. On METAFONT request, it calls the MF Script module by passing the METAFONT file.

As shown in Figure 6, MF Script Module calls its submodule named: Font Style Extractor Module. It extract the font style parameters from the METAFONT file. For example, the METAFONT request given to the module with the italic style, this will extract the italic style parameters from the METAFONT file and apply into it. Once it extracts the font style parameters, its corresponding outline will be generated with the requested style by utilizing Vectorization Module. After extracting the characters outline, it is necessary to remove the redundant nodes from the characters shapes to make the better quality. Therefore, Node Redundancy Analysis will receive the transformed METAFONT and analyze the outline contours and remove the redundant nodes from the font to create the simplified outline. Once simplification task is done, auto-hinting will be

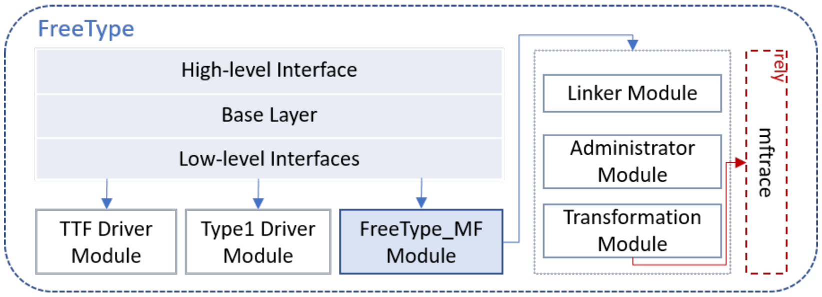


Figure 4: FreeType_MF_Module Architecture

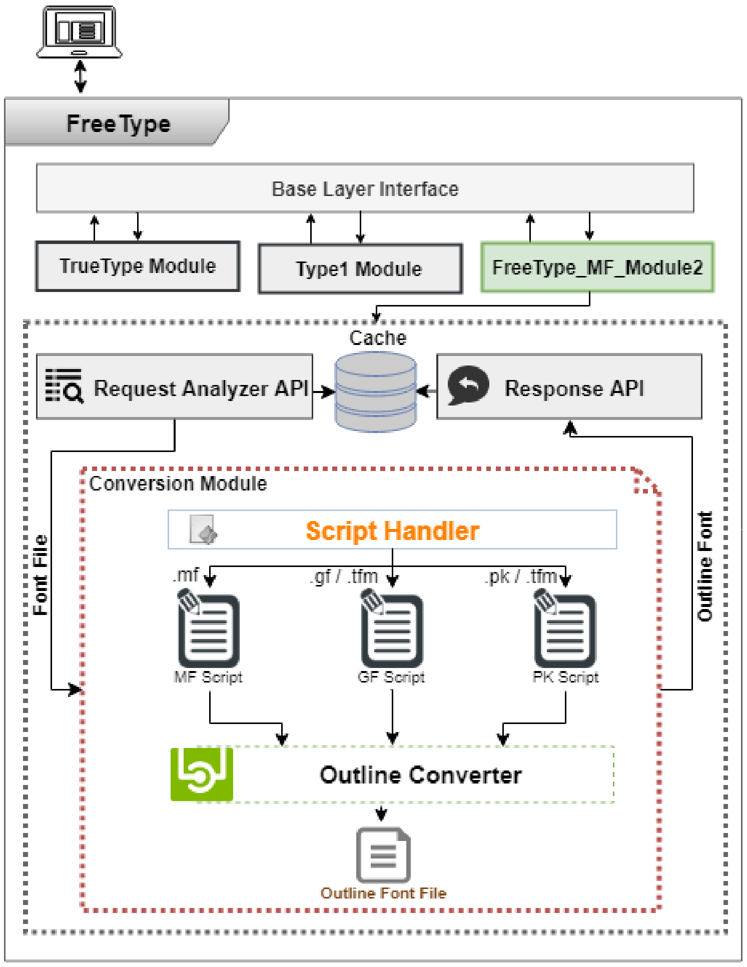


Figure 5: FreeType_MF_Module2 Architecture

performed on the font using Hinting Module. After hinting, the corresponding outline font will be generated with the Outline Converter module and sends the outline font file to the module named: Response API. It updates the Cache with 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 the GF request to the proposed module, it sends the requested font to the Request Analyzer API module. It checks whether the requested GF font is converted with correct use of mf compiler or not by analyzing the device specific information. If the requested GF file is not generated by the correct use of mf compiler, then Request Analyzer API module will not proceed as it has to

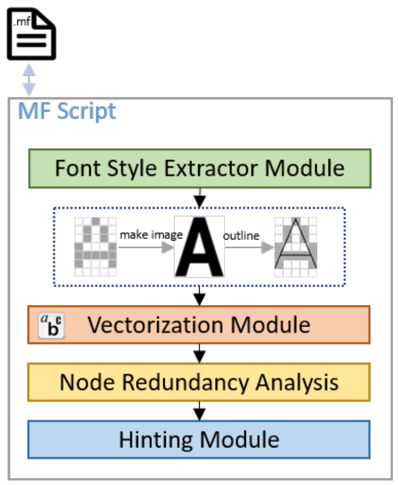


Figure 6: MF Script Internal Architecture

compute file name by using font parameters such as device resolution and magnification. But if the GF font is generated by the correct use of mf compiler, then its TeX font metric file must exist.

On GF request, its TFM must be provided for internal computation related to character shapes. Furthermore, TeX only reads the TFM instead of GF as all the font relevant information is provided by the TFM. Once Request Analyzer API module analyzes the GF request, then it checks in the Cache to get the manipulated font if exist. If requested font doesn't exist in the Cache, then the request will be forwarded to the Conversion Module where its submodule named: Script Handler handles the GF request along with its relevant tfm file.As shown in Figure 7, when GF Script receives the GF file, its submodule Extractor Module plays the main functionality. Its internal module of Font Info Extractor extracts the font related information from the TeX font metric file and extracts a sequence of bitmaps at a specified resolution from GF file.

After extraction, it merges the extracted information and gives meaning to the unreadable gf file in the form of characters images via Merge Extracted Info Module. From such bitmap relevant font, it makes character images. After merging and creating the vectorize kind of images, it extracts the outline of the characters via Outline Extractor Module. After extracting the outline, it sends the extracted outlined characters to the Simplify Module, which is capable of analyzing the font and removes the redundant nodes from the font in order to make the good quality outline. As a result, it outputs the simplified outline using the Outline Converter module internally. The newly created outline font is sent to the Response API, which updates the Cache with the generated outline font for later reusability. Once Cache updated, it sends back the response to the core FreeType module for further processing. Lastly, FreeType renders this outline font that was made from the requested GF with the styled parameter values at a specified resolution.

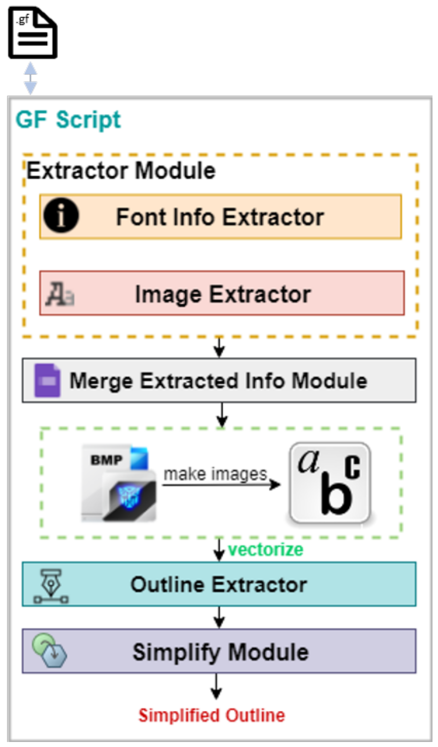


Figure 7: GF Script Internal Architecture

5.3 Packed Font (PK) Request

On PK font request, FreeType performs the same functionality till Conversion Module as it performs in Sections 5.1 and 5.2. Once Script Handler receives the requested PK font, it utilizes PK Script. As shown in Figure 8, Extractor Module extracts the raster information from the packed file. It internally utilizes the GF Script for extracting the font information from the relevant tfm file using it submodule Font Info Extractor. After extraction, it performs the autotracing on the merged font via Autotracing Module, which outputs the character images. Once done, it sends the transformed output to the Outline Extractor Module where it obtains the outline of the characters. After getting the outlined character images, it performs the outline contour analysis and remove the nodes redundancy from the outlines using the submodule named: Outline Contour Analysis Module. It sends the simplified output to the Outline Converter which creates the good quality outline font file. The generated outline font file is send to the Response API which updates the Cache and sends to the corresponding FreeType module for rendering.

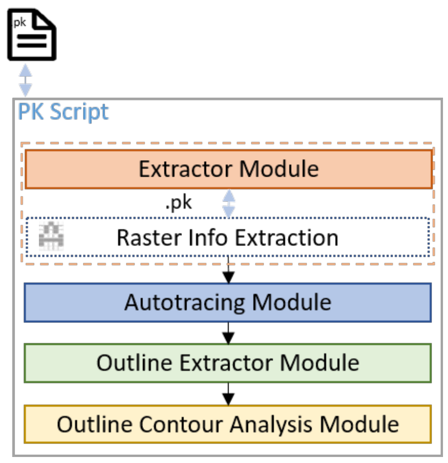


Figure 8: PK Script Internal Architecture

The proposed module provides the direct support of METAFONT, GF, and PK. It is perfectly compatible with the default module drivers of the FreeType. It can manipulate the request with the desired style parameters and scale size. In result, it provides the better quality outline font without utilizing the external libraries.

6 Experiments and Performance Evaluation

In order to test the proposed module, an application server is being utilized. The application server is responsible for rendering the text on the screen by taking the font file from the FreeType along with the requested text to be printed. FreeType can only process those fonts which are supported by it. When the client application sends the METAFONT, GF, or PK request to the 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 it on screen.

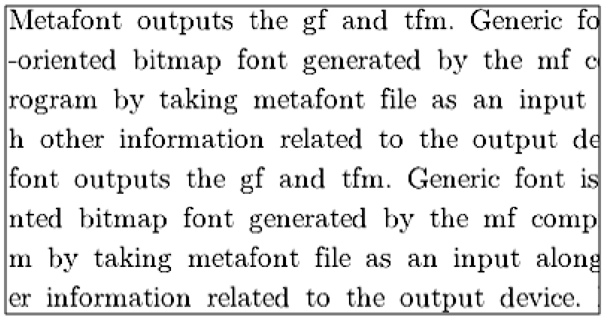
For testing purpose, the METAFONT font Computer Modern is used. The Computer Modern fonts are examined with the four unique styles: Normal, Italic, Bold, and Bold+Italic. These styles are generated by tweaking the METAFONT parameters. In order to verify the quality of the proposed module results, authors used the same four styles of another font family named: FreeSerif. The sample text comprises of words and characters, including the space characters.

The same font family is utilized to test the FreeType_MF_Module with the same four font styles. Changing the parameter values and generating new styles are explained in [10]. The same concept is applied on the proposed module for experiments. The only difference comes in case of GF and PK fonts. In order to manipulate such fonts, information of the printer device and font resolutions of the specific device is required. Furthermore, such TeX-oriented bitmap fonts cannot be directly viewed on the screen, it requires DVI drivers which makes proof sheets from a GF bitmap file where characters from the gf appear one per page in the form of .dvi file. Therefore, in the proposed module the GF and PK fonts are directly manipulated by the module without requiring the DVI drivers and previewers. 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, and generates the resultant output with the desired style.

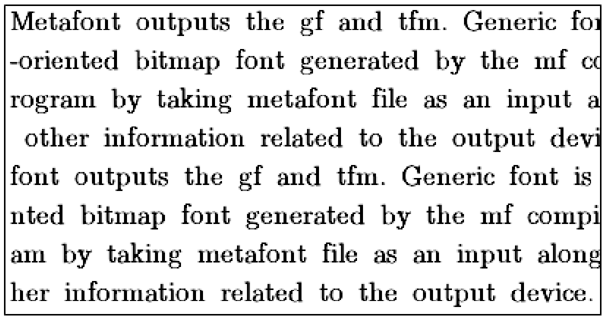
When FreeType sends the METAFONT request to the proposed module, it internally manipulates the request by extracting the styled parameters from the source file. Default style of Computer Modern METAFONT is generated by extracting the default parameters. The four font styles such as Normal, Bold, Italic, and Bold+Italic are generated by the module, and it generates the similar output in Figure. 9(a), (b), (c), (d). Using one Computer Modern METAFONT file, user can generate different font styles based on the desire and requirement.

When FreeType receives the Generic Font request by 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. After that it internally calculates the font resolution in pixels per inch by referring to a device definition. Later, it generates the output on the resulted font resolution, as similar to 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 similar to the results in Figure 9(b), (c), (d). The GF results differ slightly due to the variations in resolution than METAFONT. The authors tested the GF font with different magnifications at the time of manipulation.

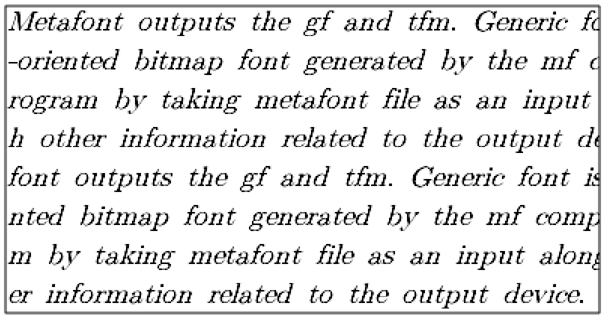
Once GF font is obtained by the METAFONT, it has a larger size which takes a lot of memory during the manipulation. In order to reduce the memory consumption, it's converted into packed form using


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.

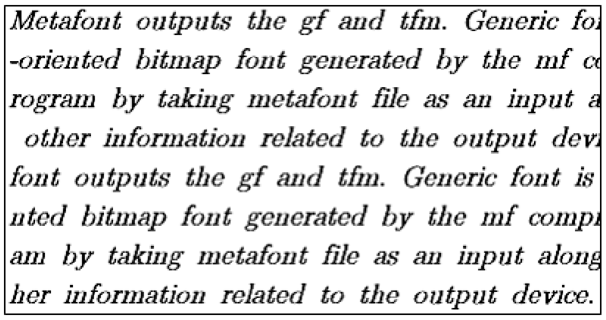
(a) Normal Style Packed Font


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


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) Italic Style Packed Font


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-Italic Style Packed Font

Figure 9: Text printed with Packed Font (PK)

Table 1: Average time of Rendering (in milliseconds)

Style	**Time efficiency of font modules (Average Time)**			
	FreeType_MF_Module	***FreeType_MF_Module2***		
		METAFONT	**GF**	**PK**
Normal	6 ms	5 ms	6 ms	4 ms
Bold	7 ms	6 ms	6 ms	6 ms
Italic	6 ms	6 ms	5 ms	4 ms
Bold+Italic	8 ms	8 ms	7 ms	6 ms

the utility gftopk. It contains the same information and style parameters which were utilized at the time of GF experiment. Therefore, their resultant output only differs at the performance level rather than quality. The resultant output for the PK request is similar like GF at the same font resolution. As shown in Figure 9,font styles such as Normal, Bold, Italic, and Bold+Italic are generated by the module. The authors compared the obtained results with the FreeType_MF_Module. Therefore, it is concluded that the results are quite similar and proposed module handles the TeX-oriented bitmaps fonts along with the METAFONT inside the FreeType without reliance related to the conversions.

The authors have not only considered the quality factor of the generated font using the proposed module, but also the performance factor. As shown in Table 1, the performance of FreeType_MF_Module is relatively slower on processing the Bold and Bold+Italic font style of METAFONT. It takes time due to the dependency on the external library such as mftrace. Therefore, the proposed module overcomes such performance and dependency issues and added the dual functionality by integrating the TeX-oriented fonts. The GF font takes a little more time compared to PK font but less time than METAFONT font as it is already in the compiled form. The PK font takes a less time than METAFONT and GF, as it is the compressed and compiled form of GF.

The proposed FreeType_MF_Module2 provides the parameterized font support to the users. The proposed module doesn't require the preconversion before giving it to the FreeType rasterizer. The client applications which utilizes the FreeType internally can utilize the METAFONT and TeX-oriented bitmap fonts such as GF and PK using the proposed module. Users can utilize such fonts as it utilizes the TrueType fonts using the FreeType. The proposed module can be utilized in the FreeType font engine as a default driver module. The proposed module will work the same as the other driver modules works in the FreeType. It is able to support the real time conversions on a modern Linux environment.

7 Conclusion

In this paper, a module is proposed for the FreeType font rasterizer which enhanced its functionality by adding the parameterized and TeX-oriented bitmap fonts. FreeType supports many different font formats but doesn't support the fonts which are utilized only in the TeX environment such as GF and PK. It is unable to support the parametrized font such as METAFONT. Although the recent studies provided a way to utilize the METAFONT inside FreeType, however, it has the dependency issues which effects the performance of the module. Furthermore, it can only handle the METAFONT request. Therefore, the proposed module overcome these issues and added the TeX-oriented bitmap support as well. Using the proposed module, users can use the METAFONT, GF, and PK fonts without using other drivers for conversion purpose. Such fonts are specific to the TeX environment, therefore, using the proposed module users can utilize these fonts outside the TeX environment.

Furthermore, the proposed module overcome the disadvantages of the outline fonts which limits the users to change the font style using the existing font file. It requires to create the different font file for every distinct font style with the different sizes as well. Therefore, for creating a new font style in outline fonts for the CJK fonts consumes time and cost, as these are complicated in shapes as compared to the alphabet-based fonts. A various studies have been conducted to implement the CJK fonts, such as Hongzi[14], including 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 have more than several thousands of phonemes. The proposed module optimization and utilization for the CJK fonts will be considered in future.

References

[1] Donald E. Knuth, *S. Song. Development of Korea Typography Industry,* Appreciating Korean Language, 2013.

[2] Jaeyoung Choi, Sungmin Kim, Hojin Lee, Geunho Jeong, *MFCONFIG:* METAFONT plug-in module for Freetype rasterizer TUG 2016 (TUGboat, 2016): 163170.

[3] Y. Park., *Current status of Hangeul in 21th century.* Type and Typography magazine The T, 7th.

[4] Donald E.Knuth, *Computers and typesetting.* Volume c: The Metafontbook. TUGboat, 1986.

[5] Web2c: *A TeX implementation.* http://tug.org/texinfohtml/web2c.html

[6] 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

[7] David Turner, Robert Wilhelm, Werner Lemberg, *FreeType,* www.freetype.org.

[8] Rainer Menzner, *A library for generating character bitmaps from Adobe Type 1 fonts.* http://www.fifi.org/doc/t1lib-dev/t1lib_doc.pdf.gz

[9] Donald E.Knuth Metafont: The Program. Addison-Wesley, 1986.K. Packard, *Fontconfig,* Gnome User's and Developers European 2002.

[10] Jaeyoung Choi, Ammar Ul Hassan, Geunho Jeong, *FreeType_MF_Module,* 2016. https://tug.org/tug2018/preprints/choi-freetype.pdf

[11] Scalable Fonts for MetaFont, *mftrace* http://lilypond.org/mftrace/

[12] Autotrace library. http://lilypond.org/mftrace/

[13] Karel Piska, *Creating Type 1 Fonts from METAFONT Sources, Comparison of Tools, Techniques and Results,* 2004.

[14] Javier Rodrıguez Laguna: *Hong-Zi: A Chinese METAFONT,* Communications of the TEX Users Group, TUGboat, Vol 26, No.2 pp.125-141,2005.

[15] Jaeyoung Choi, Gyeongjae Gwon, Minju Son, Geunho Jeong, *"Next Generation CJK Font Technology Using the Metafont",* LetterSeed 15, pp.87-101, Korea Society of Typography, 2017. 6.