Google Summer Of Code 2018 Proposal

For

Project:

Integrating VFlib's TeX format drivers into FreeType

Parth Wazurkar

Mentor

Werner Lemberg

Personal Information

Name	Parth Wazurkar
Country	India
University	Indian Institute of Information Technology, Nagpur, M.H, India
Degree and Major	B-Tech in Computer Science and Engineering
Email	parthwazurkar@gmail.com
Contact Details	+919552283766/+918668509601
Time Zone	Nagpur, India UTC+5:30
Website	https://parthw1.github.io
Github	https://github.com/parthw1

About Me

I am Parth Wazurkar a sophomore from Indian Institute of Information Technology, Nagpur, India pursuing Computer Science and Engineering. I am interested in open source software development and competitive coding. Apart from this I like to work on cutting edge technologies. I have worked on data science and have published a research paper in an IEEE conference on the same. I am well versed with C language and Unix build tools which are a requirement for the project.

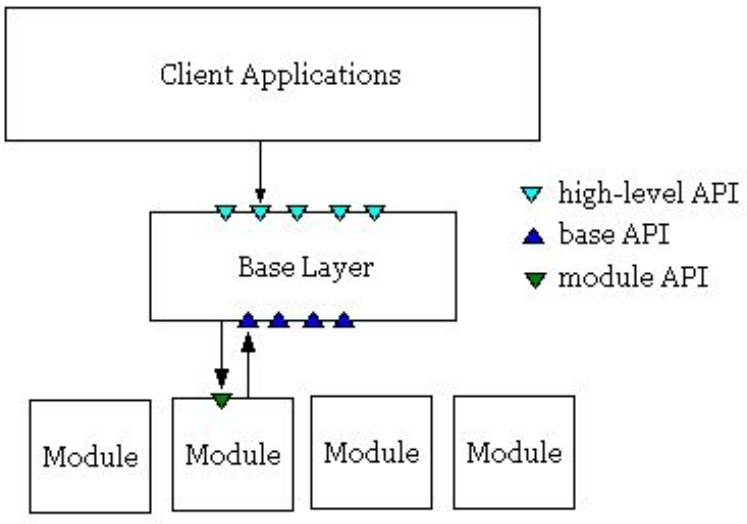
Project Proposal Information

Title: Integrating VFlib's TeX format drivers into FreeType.

Background: FreeType is a freely available software library to render fonts (More details can be found **here**). By default, FreeType supports the multiple font formats like TrueType, CFF, OpenType, BDF, etc. However, a need for the support to TeX's bitmap formats has been felt in the past (proof can be found **here**). Providing support to multiple formats also enhances the usability of the library.

Proposal Abstract: The goal of this project is to add the support of TeX's bitmap font formats (GF, TFM, PK and VF fonts) into FreeType by providing new modules to handle them. The VFLib library contains mature support for TeX's bitmap formats (in particular GF and PK fonts, together with TFM metric files and VF virtual fonts). This project aims at using the existing modules in VFlib as a reference to develop new modules for FreeType on the lines of existing bitmap drivers already available in FreeType like BDF, PCF and WINFNTS.

Detailed Project Description: FreeType can be described as a collection of different ***components***. Each one of them is a more or less abstract part of the library that is in charge of one specific task. Depending on the context or the task, the base layer of the codebase then calls one or more ***module*** components to perform the work.



Basic library design (details ommited)

FreeType recognizes and manages several kinds of modules like *Renderer modules*, *Font Driver modules*, *Helper modules* and *auto-hinder modules* for different tasks associated with them (as mentioned **here**).
Font driver modules are used to support one or more font formats. Each different font driver acts as a different module in the FreeType library. These drivers are governed by some set of functions provided by the FreeType API in the base layer. For the project these functions will need to be modified.

Relevant insights from the FreeType codebase:

- As mentioned here, the FT_Face object is a handle to the FT_FaceRec structure, which contains different font data required for rasterization and glyph loading. An FT_FaceRec object contains details about the dimensions like bounding box, character mappings, associated glyph details in a structure FT_GlyphSlotRec, it also has a handle to FT_CharMap determining the details about the current active charmap for this face in the FT_CharMapRec structure.
- When a pathname is provided for the font file to open to the FT_New_Face function, it determines the above mentioned parameters in the FT_Face object with the help of Font Driver modules. Every FT_Face object is *owned* by the corresponding font driver, depending on the original font file's format.
- The Font Driver extracts the information from the font file into FT_Face object. A Font Driver is an instance of the FT_DriverRec structure which is a derivation of FT_ModuleRec and which provides a handle to a given FreeType font driver object.
- FT_Driver_ClassRec which is a derivation of FT_Module_Class which provides an amalgamation of different pointers to driver methods.
- For any font driver for a particular font format, it we will have to define the variables which it provides for rasterization. A structure needs to be defined so as to encapsulate those variables according to their relations. After declaration of the structures methods have to be developed so that these variables can be brought to best use and then a renderer can render font.

For a Font Driver the methods defined in the FT_Driver_ClassRec are to be implemented so as to extract font data from the font file.
Thus, I will be implementing these methods with the help of the available source code for drivers for TeX font formats in VFlib (source can be found **here**).

About VFlib's Drivers: VFlib follows a completely different approach in initialising font data and glyph loading when compared to FreeType, it involves searching a font database file and handles kpathsea issues for searching. Thus, the implementation of

these drivers in FreeType will require to bypass this issues. Taking a note of this the functions below have been determined accordingly.

Details for the GF driver

GF Files: As mentioned here, A GF file is a stream of 8-bit bytes (opcodes) that may be regarded as a series of commands in a machine-like language. It has a fixed format in which these opcodes are written for a particular file.

VFlib's implementation of gf driver:

- Source can be found here in gf.h, gf.c and drv_gf.c files
- VFlib pre defines the above mentioned opcodes in different macros which when compared with the opcodes in the file directly returns the values. Similarly I will use these macros in our driver.
- The s_font_gf and s_gf_glyph structures stores parameters required like bounding box dimensions, charmaps, and other metrics in VFlib which is same as to what is required for the **GF_FaceRec** (from FT_FaceRec) structure.
- The gf_loader and gf_read_glyph functions together extracts glyph metrics from the file by appropriately matching the opcode macros in the file and will be used for the implementation of **GF_Face_Init** function.
- Implementing **GF_Face_Done** using GF_Close function.
- For the **GF_Glyph_load** function VFlib has similar GF_GetGF, GF_SetGFGlyph, GF_GetGFGlyph functions which will help in glyph loading.

Details for the TFM driver

TFM Files: As mentioned here, the information in a TFM file appears in a sequence of 8-bit bytes. The number of bytes is always a multiple of 4, we could also regard the file as a sequence of 32-bit word. The first 24 bytes (6 words) of a TFM file contain twelve 16-bit integers that give the lengths of the various subsequent portions of the file. The rest of the TFM file may be regarded as a sequence of ten data arrays having the informal specification.

VFlib's implementation of tfm driver:

- Source can be found here in tfm.h, tfm.c and drv_tfm.c files
- The s_tfm (defined in texfonts.h file in the source directory) and s_font_tfm structures contains font info, metric info, and bounding box dimensions in VFlib which will be used for defining **TFM_FaceRec** (from FT_FaceRec) structure.
- The vf_tfm_open alog with vf_tfm_metric, read_tfm functions together extracts metrics from the file appropriately, and will be used for the implementation of **TFM_Face_Init** function.

- The TFM files themselves only know about the sizes of characters and their interactions with each other, but not what characters look like. Although we can define a **TFM_Glyph_Load** function by using tfm_get_bitmap function, which will essentially interpret the TFM file as a file containing black (or white) rectangles of various sizes as glyph images. This is useful when some fonts in PK or VF formats are unavailable for some reason because character in non-existing fonts can still be printed as black rectangles.
- Implementing **TFM_Face_Done** using vf_tfm_free function.

Details for the PK driver

PK Files: As mentioned here, Similar to a GF file a PK file too is organized as a stream of 8-bit bytes. At times, these bytes might be split into 4-bit nybbles or single bits, or combined into multiple byte parameters.The packed file format is a compact representation of the data contained in a GF file. The information content is the same, but packed (PK) files are almost always less than half the size of their GF counterparts.

VFlib's implementation of pk driver:

- Source can be found here in pk.h, pk.c and drv_pk.c files
- VFlib pre defines the opcodes of PK file in different macros which when compared with the opcodes in the file directly returns the values. Similarly I will use these macros in our driver.
- The s_font_pk and s_pk_glyph structures stores parameters required like bounding box dimensions, charmaps, and other metrics in VFlib which is same as to what is required for **PK_FaceRec** (from FT_FaceRec) structure.
- The pk_loader and pk_read_glyph functions together extracts glyph metrics from the file by appropriately matching the opcode macros in the file and will be used for the implementation of **PK_Face_Init** function.
- For the **PK_Glyph_load** function VFlib has The PK_GetPK, PK_SetPKGlyph, PK_GetPKGlyph functions which will help in glyph loading.

Details for the VF driver

VF Files: As mentioned here, Similar to a GF file, a VF file is organized as a stream of 8-bit bytes(opcodes) that may be regarded as a series of commands in a machine-like language. It has a fixed format in which these opcodes are written for a particular file.

VFlib's implementation of vf driver:

- Source can be found here in vf.h, vf.c and drv_vf.c files

- VFlib pre defines these opcodes in different macros which when compared with the opcodes in the file directly returns the values. Similarly I will use these macros in our driver.
- The s_font_vf, s_vf structures stores parameters required like bounding box dimensions, charmaps, and other metrics in VFlib which is similar to the **VF_FaceRec** (from FT_FaceRec) structure.
- For the **VF_Glyph_load** function VFlib has The vf_vf_get_vf functions which will help in glyph loading.

Project Division: I have divided the project into 4 phases and 4 milestones.

- **Phase 1:** GF Driver Phase.
  Milestone 1: FreeType supports gf fonts!!
- **Phase 2:** TFM Driver Phase.
  Milestone 2: FreeType supports tfm fonts!!
- **Phase 3:** PK Driver Phase.
  Milestone 3: FreeType supports pk fonts!!
- **Phase 4:** VF Driver Phase.
  Milestone 4: FreeType supports vf fonts!!

Project Plan: This is the project plan, in which I have tried to schedule work as per the timeline of GSoC.

Days	Work to be done
April 23	Students Projects Announced
April 23 - May 14	Community Bonding Period
April 23 - April 27 (1 week)	● Get acquainted with the workflow of FreeType ● Set-up the required environment for the project ● Discussion on weekly status update method. ○ Blog/Wiki? ○ Email? ● Setup of wiki/blog/mailing list as per above. ● Establish clear timings for meetings, code reviews and discussions.
April 28 - May 5 (1 week)	● Fill in the gaps in the understanding of FreeType design. ● Fill in the gaps in FreeType code workflow. ● Recap through the bitmap font drivers code. ● Recap on how Type1 module attaches a metric file to a font(for pk and vf fonts).

May 6 - May 14 (1 week)	● Exploring VFlib. ● Recap through the VFlib's font drivers. ● Discussion with my mentor on implementation and getting doubts cleared. ● Set up a clear map for implementation of new drivers.
May 14	First Coding Period Starts
May 14 - June 4 (3 weeks)	**Phase 1: GF Driver phase**
May 14 - May 25 (~2 weeks)	● Defining structures for setting up the font configuration requirements. ● Implementation of GF_FaceRec, GF_CMapRec. ● Defining gf_driver_class as an object of FT_DriverClassRec. ● Implementation of the methods defined in the gf_driver class structure, ○ GF_FaceRec ○ GF_FaceInit ○ GF_Face_Done ○ GF_Glyph_load ○ GF_size_request ○ GF_size_select
	Progress Report and status update.
	● Integrating the new driver into FreeType ○ Implementing Makefiles for gf driver and refactoring existing Makefiles. ● Seek regular feedback from the mentor and make revisions accordingly..
May 26 - May 27 (2 days)	Refactoring the driver defining structures and files to accommodate new drivers ● Modifying the FT_MODULE_H file ● Modifying the ft_default_modules[] function in ftinit.c by refactoring FT_CONFIG_MODULES_H ● Refactoring of other required files. ● Refactoring for all the formats will be done here.
May 28 - June 4 (1 week)	● Setting up test cases and documentation. ● Changes according to the review comments. ● Testing of the new gf driver. ● Immediate bug fixes (if any).

	Progress Report and status update.
June 4 - June 11 (1 week)	● Midterm evaluation preparations
Milestone 1 Achieved!! *FreeType now supports gf fonts*	
June 11 - June 15	Midterm Evaluations
June 16	Second Coding Period starts
June 16 - June 30 (2 weeks)	**Phase 2: TFM Driver phase**
June 16 - June 24 (1 week)	● Defining structures for setting up the font configuration requirements. ● Implementation of TFM_FaceRec and setting up TFM_CMapRec. ● Defining tfm_driver_class as an object of FT_DriverClassRec. ● Implementation of the methods defined in tfm_driver class ○ TFM_FaceRec ○ TFM_Face_Init ○ TFM_Face_Done ○ TFM_Glyph_load ○ TFM_size_request ○ TFM_size_select ● Integrating the new driver into FreeType ○ Implementing Makefiles for tfm driver ○ Refactoring existing Makefiles
June 25 - June 30 (1 week)	● Setting up test cases and documentation. ● Changes according to the review comments. ● Testing of the new tfm driver. ● Immediate bug fixes (if any).
	Progress Report and status update.
Milestone 2 Achieved!! *FreeType now supports tfm fonts*	
July 1 - July 18 (2.5 weeks)	**Phase 3: PK Driver phase**

July 1 - July 10 (1.5 weeks)	● Implementation of PK_FaceRec and setting up PK_CMapRec. ● Defining pk_driver_class as an object of FT_DriverClassRec. ● Implementation of the methods defined in pk_driver class ○ PK_FaceRec ○ PK_Face_Init ○ PK_Face_Done ○ PK_Glyph_load ○ PK_size_request ○ PK_size_select ● In pk file format the metrics are provided by a tfm file. ○ Thus we will need to implement a method of FT_Face_AttachFunc in the FT_DriverClassRec. ○ Similar to the T1_Read_Metrics (for Type1 fonts) in t1afm.c file. ● Integrating the new driver into FreeType ○ Implementing Makefiles for pk driver ○ Refactoring existing makefiles
	● Second midterm evaluation preparation
July 9 - July 13	Second Midterm Evaluations
July 14 - July 21 (1 week)	● Setting up test cases and documentation. ● Changes according to the review comments. ● Testing of the new pk driver. ● Immediate bug fixes (if any).
	Progress Report and status update.
Milestone 3 Achieved!! *FreeType now supports pk fonts*	
July 21 - August 4 (2 weeks)	**Phase 4: VF Driver phase**
July 21 - July 30 (1.5 weeks)	● Implementation of VF_FaceRec and setting up VF_CMapRec. ● Defining vf_driver_class as an object of FT_DriverClassRec. ● Implementation of the methods defined in vf_driver class ○ VF_FaceRec

	○ VF_Face_Init ○ VF_Face_Done ○ VF_Glyph_load ○ VF_size_request ○ VF_size_select ● Similar to the pk file format, here too the metrics are provided by a tfm file. ○ Thus we will need to implement a method of FT_Face_AttachFunc type in the FT_DriverClassRec. ● Integrating the new driver into FreeType ○ Implementing makefiles for vf driver ○ Refactoring existing makefiles
July 31 - August 6 (1 week)	● Setting up test cases and documentation. ● Changes according to the review comments. ● Testing of the new vf driver. ● Immediate bug fixes (if any).
	Progress Report and status update.
Milestone 4 Achieved!! *FreeType now supports vf fonts*	
Aug 6 - Aug 14	Students Submit Code and Final Evaluations
Aug 14 - Aug 21	Mentors Submit Final Evaluations
August 22	Results Announced
Onwards	Keep contributing to FreeType and will keep resolving the issues faced by the users while using any of the above features.

Note:

- Open to change the timeline according to mentor.
- Holidays and other stuff are also considered while writing project plan.
- GSoC timeline events have been marked in grey, and merged, for better representation. Start of phases are marked as light grey and achievement of milestones has been marked green.

Why GSoc with FreeType?

I have always been passionate about projects which work on the middleware, which is surely the main motivation for me to work with FreeType, and to write this proposal. I am really excited to work with FreeType. While learning more about FreeType I was fascinated by the fact about the amount of rendering required to even display a simple text character, this acted as a catalyst to my interest in FreeType.

Regarding GSoC Timeline and Working hours

I have 24/7 internet access and do not have any work commitments during the summer and thus, I shall be able to work for 8 hours per day, thus completing the required target of 30 working hours per week during GSoC period. I will be working for 2 hours per day from 25 April - 4 May because of my college exams. I will compensate the time during the rest community bonding phase.

Interaction with the FreeType community

I have been active with the FreeType community for more than 2 months at the writing of this application, and feel confident about the codebase of FreeType. I have also been actively in touch with mentors on mailing lists. I am thankful for the warm support extended to me as a newbie into the open source community in the initial stages of my contribution.

Thanking Note to organisation

I would like to thank all the members of FreeType organization, to give me this opportunity to write proposal and of providing possibility to work with you.
I will look forward to every feedback from organisation members reviewing this document, and would be glad to discuss/change accordingly.