Vol.2011-DD-83 No.2
2011/11/18

電子文書中に埋め込まれたフォントの描画プログラムによる特定手法の考察

鈴木俊哉†

電子文書に埋め込まれたフォントは、文書のデータ容量を削減するため、様々な付加情報が削除される。特に PDL として発達してきた PDF は、フォントのファミリ名、バージョン、著作権表示なども削除されてしまう。PDF 処理系によってはこれらを残すものもあるが、規格上は必須ではないので、これによってフォントを特定することには困難が伴う。特に、人名・地名など字形の詳細が重視される場合など、ファミリ名だけでは特定が不十分な場合も多い。本研究では、電子文書の表示の際のグリフ描画プログラムによってグリフ単位でフォントを特定する方法について考察する。

A Study On the Per-Glyph Font Identification of the Embedded TrueType

suzuki toshiya†

The TrueType fonts embedded in the electronic documents may lack its metadata to reduce the size of the data size of the document. Especially, PDF (ISO/IEC 32000) was originally developed as a Page Description Language (PDL), thus the embedded TrueType fonts in PDF lack the family name, version number and copyright acknowledgement that are included in the original TrueType fonts. It is possible for some PDF production system to retain such information, but it is not required and most PDF browser does not retrieve them, it is difficult to identify the fonts used in the document. In some office document dealing with the special glyphs for personal or place names, the identification of the glyph (which font is used for the glyph) is important, and the family name may be insufficient to identify (e.g. there are many fonts that their family names were not changed during the switching from JIS X 0213:2000 to JIS X 0213:2004). In this report, the method to identify the font by the TrueType glyph instruction embedded in PDF is discussed.

1. はじめに

従来、電子文書はレンダリング結果の処理系依存性を積極的に認め、同時に固定的なレンダリング結果を保証するためのデータは外部参照とし、文書の肥大化を避けてきた。しかし、近年では処理系の高機能化を背景とし、標準的で汎用性があるデータであれば、電子文書内部のデータとして埋め込み、処理速度が重要な処理系や、埋め込まれたデータを処理できない場合はそれを無視しても良いという方針が受け入れられつつある。

固有名詞を扱う一部の行政文書では、一般には区別されない字形差を指定した文書が作成されることがあるが、通常は、使用されている字形がどのようにデザインされているかを具体的に記述しないため、行政文書に示された字形にどの程度従えば良いのか不明な場合が多い。これは特殊文字に限らず、文字符号の規格、たとえば JIS X 0208:1997 や JIS X 0213:2004 の漢字表で示されている字形はあくまでも参考情報であり、字形を指定するものではないので、これらの規格によって漢字表と同一字形を持つフォントを指定することはできないという問題がある。字形情報交換の規格として、ISO 10036 が存在するが、やはり字形をどのようにデザインするのかを具体的には記述せず、例示字形によって示す規格である。近年では Ideographic Variation Database などが普及し始めているが、これも適合性準拠の試験は定義されておらず、参照文書と同一の字形を使いたいという需要に対する完全な解にはなっていない。

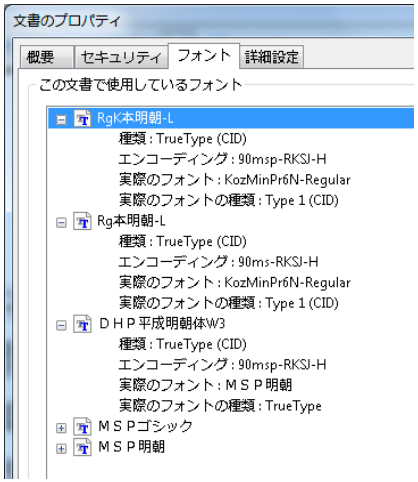


図 1: 人名用漢字見直し案の PDF でのフォント参照例

このような問題の例のひとつを図 1 に示した。人名用漢字見直し案の PDF がフォントを埋め込まずに複数のタイプフェイスを参照しており、しかも Adobe CID ではなく ShiftJIS 符号によって文字を指定しているため、Adobe CID の粒度での字形指定すらできていない例などである。

ISO/IEC 32000 として標準化された PDF は、基本的には再編集を念頭に置いていないデータ形式であるが、フォントを(単なるグラフィクスではなく、フォントと符号化テキストとして)埋め込むことができる。

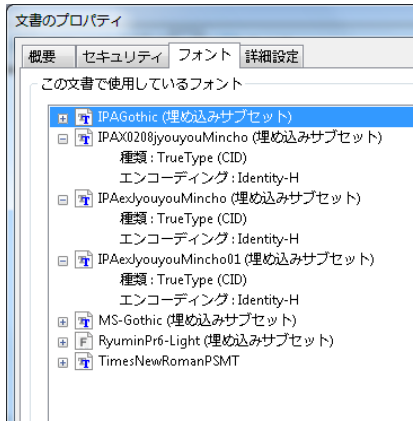


図 2:改定常用漢字表の内閣答申

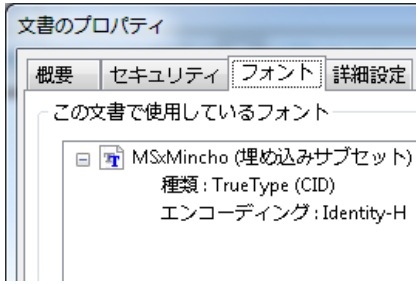


図 3:改訂常用漢字表の内閣訓令

PDF に埋め込まれたフォントについて、図 2、図 2 に示すようにフェイス名と若干のメタデータが取得可能だが、これによって十分なフォントが特定できるとは言い難い。日本でよく知られている例は、Microsoft 製品にバンドルされる明朝体フォントが表外漢字字体表や JIS2004 などによって少しずつ字形が変更されている例だが、もっとも極端な例としては、Microsoft 製品にバンドルされる繁体字フォント(MingLiU)が 2008 年に台湾の国字標準字体に準拠するために行った変更である。

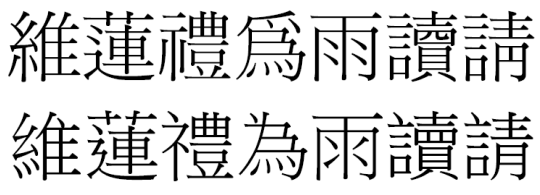


図 4: MingLiU version 3。21(上)と 7。00(下)の字形差

図 4 に示すように変更前後では字形が大きく異なるが、ファミリ名は同一なため、国字標準字体に準拠しているかどうかを判断することが困難である。本稿では、これらを機械的に区別するため、PDF に埋め込まれた TrueType[1]描画命令によるグリフ単位でのフォント特定手法と、それに要する処理時間の検討を報告する。

2. 本稿で扱う問題と解決方法

電子文書中へのフォントの埋め込みは Adobe 社の PostScript[2]にはじまり、同社の PDF[3]、Hewlett-Packerd の PCL[4]、Microsoft の OOXML[5]、XPS[6]、あるいは先ごろ第 3 版の標準化が完了した ePub[7]など様々なデータ形式で行われるようになった。また、XML 系の汎用的な埋め込みフォント形式として WOFF 形式が定められ[8]、さらに、動画像コンテンツにおいても MPEG-4[9]でフォントの埋め込みが想定されている。このうち、近年設計された OOXML、 XPS、 ePub は Unicode テキストレンダリングに関して処理系に任せる方向であり、フォントも表示に必要なグリフのみにサブセット化することは推奨されるが、汎用テキストレンダリングに使えないほどデータを削除することはできない。この意味で、TrueType フォントを単なるグリフ番号とグリフ描画プログラムのペアにまで削減する(文字符号や Unicode テキスト整形のための補助情報は全て削除される)PostScript や PDF は特異的な仕様である 。言い換えれば、PostScript や PDF に埋め込まれたフォントを特定することができれば、その他のコンテナの埋め込みフォントの特定には問題はないと言える。

2.1 PostScript/PDF 中の TrueType フォントの特徴

PostScript や PDF に TrueType フォントを埋め込む場合のデータ形式は PostScript Type42 フォーマットである。以下に TrueType フォントと Type42 形式の概略を述べ、埋め込みに際して削減される情報について整理する。

TrueType フォントはいくつかのサブテーブルを並べた構造をしており、そのヘッダにはサブテーブルへのアクセスを補助するため、サブテーブル名タグ、サブテーブルへのオフセット、サブテーブルのデータ長、サブテーブルのコンテンツのチェックサム(全て 32 ビットである)が記述されている。サブテーブルは大別して以下のグループがある。

a) 文字符号・グリフ番号対応(cmap)
b) グリフ
   1) アウトライン描画プログラム、共用定数表、共用サブルーチン(loca、 glyf、 gasp、 cvt、 prep、 fpgm、 LTSH)
   2) ビットマップデータ(EBDT、 EBLC、 EBSC)
c) レイアウト制御(hdmx、 hmtx、 VDMX、 vmtx、 kern、 その他 OpenType レイアウトテーブル)
d) フォント特性、メタデータ(post、 name、 head、 hhea、 vhea、 OS/2、 PCLT、 DSIG)

太字で書いたものは ISO OpenType 仕様で必須とされるサブテーブルである。これらのテーブルの依存関係を図 5 に示した。PS/PDF 処理系はグリフ番号を指示してグリフのラスタ画像を得るだけであるので、一部のテーブルしか必要でなく、Type42 形式でも一部のテーブルのみ必須とされている。言い換えれば、Type42 形式で埋め込まれたデータを抽出しても単体で利用可能な TrueType フォントとはならない。

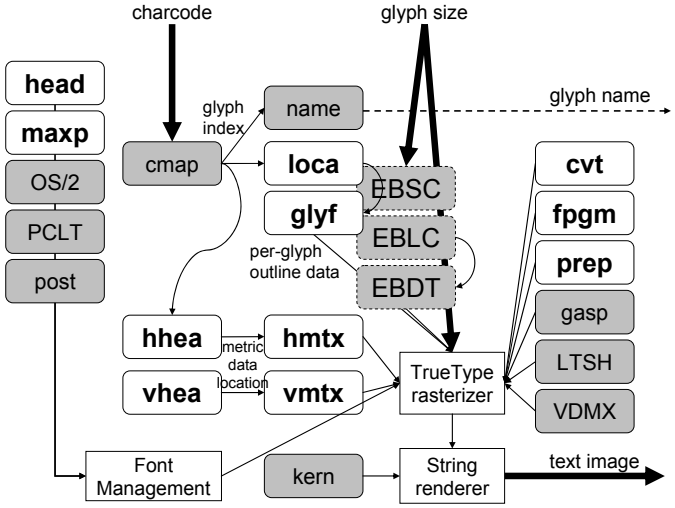


図 5: TrueType フォント内部のサブテーブルと、想定されているデータフロー

PostScript 処理系におけるマルチバイト符号化テキスト用のフォントの構造を図 6 に示す。図中に太線で示したブロックは PostScript または PDF オブジェクトとしてインタプリタが任意にアクセスできる単位である。CID-keyed Type42 の sfnt は sfnt 全体は PostScript/PDF のオブジェクトであるが、その内部の loca、 glyf テーブルなどは PostScript/PDF オブジェクトではない。純粋な PostScript フォントである CID-keyed Type1 は PostScript/PDF オブジェクトに対するアクセスでグリフ 1 個の描画命令を抽出できるが、CID-keyed Type42 は PostScript/PDF オブジェクトに対するアクセスでは TrueType のグリフ ID までしか抽出できず、TrueType フォントの構造を解釈し、グリフ 1 個の描画命令を抽出する操作が必要であることがわかる。

PostScript はスタック型の言語で、インタプリタによって処理するが、フォント等の資源は PostScript インタプリタが実行時に書き込み・読み出しができるテンポラリなメモリに格納される。このメモリは資源を名前によって管理するため、辞書と呼ばれる(辞書は同じ名前の異なるオブジェクトを格納することはできないため、PostScript 処理系の ROM に書き込まれているフォントと同名のものを文書中に埋め込むには一時的に書き込み可能な辞書を作成して、その中でフォント資源を上書きするなどしなければならない)。資源が名前で管理されるため、PostScript インタプリタでは全てのフォントオブジェクトに PostScript フォント名が書き込まれている。PostScript は辞書の鍵となる文字列について制限がある(ASCII 文字で、/と空白を含んではならない)ため、TrueType フォントなどのフェイス名とは別になっていることが多い。また、必須ではないが、ファミリ名(Bold、Italic などの修飾子がついていないもの)、フェイス名、バージョン、著作権表示などを制限のない文字列で格納することもできる。

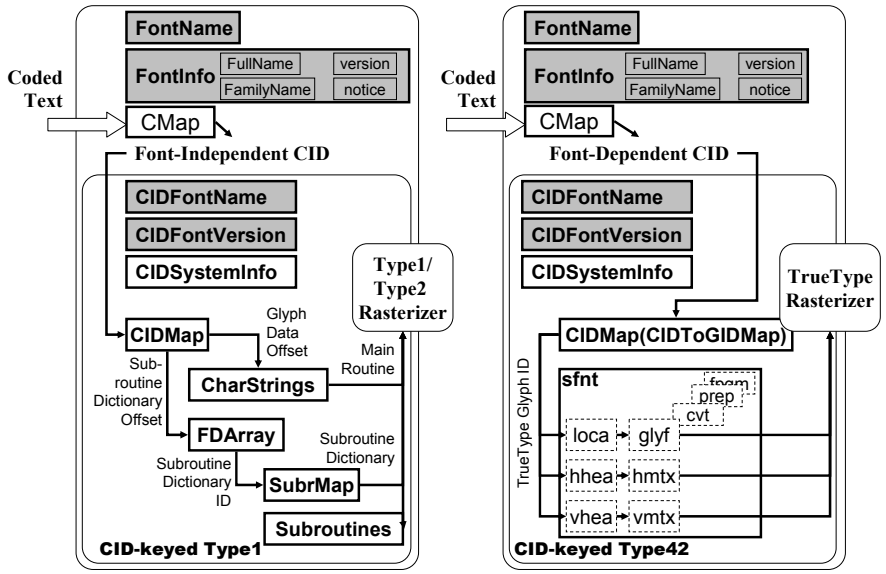


図 6: PostScript/PDF 中のフォントオブジェクトの構造とデータフロー
PDF 中で必須でないものはグレーで示した。

スタック型言語である PostScript に対し、PDF の全体構造はベクタグラフィクスやビットマップデータ、テキスト、フォント、ページレイアウト、色管理データなど特定用途のデータをそれぞれ個別にカプセル化したオブジェクトとして扱い、文書は様々なオブジェクトを参照する辞書として構成する(ベクタグラフィクスやテキストの描画処理は PostScript に近いが、各オブジェクトで使用可能な命令群が制限されており、たとえばテキスト描画オブジェクトの中でベクタグラフィックのパス構築命令を使うようなことはできない)。PDF のオブジェクトはオブジェクト番号およびリビジョン番号によって管理される。PostScript では資源を名前で管理していたが、PDF では名前を使わないため、PostScript では必須であったフォント名などは PDF のフォント

オブジェクトでは必須でなくなっている(厳密には PDF 1。0 では必須であったが、それ以降のバージョンでは省略可能)。

従って、PostScript/PDF の双方を対象とした場合、PostScript フォント名によってフォントを特定することはできず、描画データをもとに特定せざるを得ない。さらに、標準的な文字符号とグリフ ID を変換する CMap が用いられている場合は、CID-keyed Type42 フォントでも cmap 相当の情報を復元することができるが、近年の多くの PDF 生成系では PDF に格納する符号化文字列を TrueType フォントのグリフ ID 列に置きかえてしまうものが多いことにも注意する必要がある。

2.2 既存手法と問題点

フォントの特定としてもっとも単純な方法はラスタしたビットマップデータを画像として比較する手法であろう。しかし、PDF に埋め込まれた TrueType フォントは cmap が削除されているため、埋め込みフォント内のどのグリフが、どの文字符号に対応するか直接には判別できない。従って、仮に使用頻度の高い文字を指標文字として検査しようとしても、文字符号情報が削除されているため、全てのグリフに対して比較しなければならない。さらに、本稿で扱う問題では「処理系が持っているフォントのうち、どれか」だけではなく、「処理系が持っていないフォントの何であるか」を特定できることが重要である。PDF 中でわずか数個のグリフにしか使われていない場合でも特定を目指そうとすれば、ビットマップデータ比較手法では多くのフォントのラスタ結果をデータベースとして持つ必要があり、一般的なフォントのライセンスに抵触する恐れがある。

これらの問題を回避できる手法としては、TrueType フォントのテーブルのうち、cvt、fpgm、 prep テーブルはサブセット化に際して TrueType 描画命令を詳細に検査しない限り改変できないことに注目し(アーカイブライブラリを静的リンクする場合、リンカはアーカイブ内の不要なオブジェクトを削除することができるが、同一オブジェクト内部の不要な部分まで検査して削除することは難しい)、これらのテーブルのハッシュ値を用いて特定する手法がある。この方法は TrueType フォントのヘッダに含まれるチェックサムをハッシュ値と見なしても有効であることが示されており、理想的な場合はまったくハッシュ値を計算せずに、Type42 フォントオブジェクトのうち、sfnt オブジェクトの先頭 100 バイト程度の読み出しによってフォントが特定できるので、フォント単位の特定には有効である。しかし、フォントによっては cvt、 fpgm、 prep を持たない場合があり(この手法は本来は漢字フォントなど複合グリフを含むフォントの特定のためのものである)、さらに、フォントの派生バージョンなど、大半は同じだが一部のグリフのみ修正したようなフォント群を識別することは難しいという問題もある。

以上より、従来手法の問題点は以下の 2 点に整理できる。

- フォント単位での特定では詳細なバージョンの特定ができない
- ビットマップデータを全てのグリフデータに対し持つことは一般的なライセンス上困難

この解決方法として、本稿では TrueType フォントのグリフごとの描画命令に対してハッシュ値を計算し、各フォント・各グリフのハッシュ値データベースと照合することでグリフ単位でのフォント特定を考える。

3. 提案手法とその評価

3.1 評価実装の設計および実験環境

評価用のソフトウェアはオープンソースの PDF レンダリングライブラリである poppler-0。18 に対して、テキスト処理ルーチンにフォント抽出、グリフ描画命令抽出、ハッシュ値計算、データベース照合の機能を追加することで実装した。poppler のテキスト処理ルーチンは、PDF の埋め込みフォントは全く抽出せず、カプセル側の符号情報のみ処理する。そのため、フォントキャッシュの仕組みがないので、本稿での評価実験では 1 文字ごとにフォントを抽出した。

データベースは、Windows XP SP2、 Microsoft Office 2003 およびいくつかの商用フォントパッケージ にバンドルされるフォントをもとに、SQLite3 で下記のような構成のデータベースを作成した。

- フォントファイル名、フォントフェイス名、フォントバージョンの組に対し通し番号(フェイス ID)を与えるテーブル
- グリフ名に対し通し番号(グリフ名 ID)を与えるテーブル
  - ただし、u4E00 。。。 u9FFF などが個別のグリフ名とならないよう、いくつかの文字符号またはフォント内グリフ番号に基づくフォーマットはフォーマットに対して通し番号を与えた
- グリフ描画命令長(上位 32 ビット) + グリフ描画命令の crc32(下位 32 ビット)からなる 64 ビット整数に対し、MD5 ハッシュ値の上位 64 ビット、MD5 ハッシュ値下位 64 ビット、SHA1 ハッシュ値第 0-63 ビット、SHA1 ハッシュ値第 64-127 ビット、SHA1 ハッシュ値第 128-159 ビット、フェイス ID、グリフ名 ID を与えるテーブル(SQLite3 では整数は 64 ビット整数までしか扱えないので、MD5 や SHA1 のような 64 ビット以上のハッシュを分解して格納した)

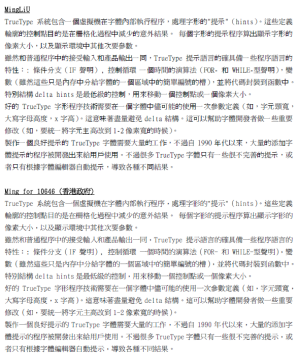
総フォント数 399 フェイス、総グリフ 2242037 個を含むデータベースを作成すると、容量 245MB となった。ここで、単純に 64 ビットハッシュ値とフェイス ID の対応を SQL 非対応の連想排列である Berkeley DB に格納した場合、そのファイルサイズは 45MB である。他のハッシュ値エントリが 6 個あることを考えると、ほぼエントリ数に比例した容量となっていると言える。

実験は Intel Atom 330、 RAM 2GB、 SATA 接続 HDD (2TB)上の GNU/Linux (x86-64)

で行った。Intel Atom 330 の設計最高クロックは 1.6GHz であるが、本実験では短時間の処理でのプロファイルデータを取るため、600MHz で動作させた。この結果、HDD は 1.6GHz では 100MByte/sec の読み取りが可能であるが、36MByte/sec 程度の速度となっている。

3.2 実験結果

評価は MingLiU および MingLiU_HKSCS の 2 書体を含む PDF(1 ページ、1031 文字)を用い、フォント抽出までで終了した場合、描画命令の抽出までで終了した場合、ハッシュ値計算およびデータベース照合まで行なった時間を計測した。ただし、照合キーとしては最も高速なハッシュとして CRC32 と描画命令長の組み合わせにより行った。


MingLiU
TrueType 系統包含一個虛擬機在字體內部執行程序，處理字形的"提示"(hints)。這些定義輪廓的控制點目的是在柵格化過程中減少的意外結果。 每個字形的提示程序算出顯示字形的像素大小，以及顯示環境中其他次要參數。
雖然和普通程序中的接受輸入和產品輸出一同，TrueType 提示語言的確具備一些程序語言的特性：：操作分支（IF 聲明），控制循環，一個時間的演算法（FOR, 和 WHILE-型聲明），變數（雖然這些只是內存中分給字體的一個區域中的簡單編號的標），並將代碼封裝到函數中，特別是精 delta hints 是最低級的控制，用來移動一個控制點或一個像素大小。
好的 TrueType 字形程序技術需要在一個字體中盡可能的使用一次參數定義（如，字元寬度，大寫字母高度，x 字高），這意味著盡量避免 delta 結構，這可以幫助字體開發者做一些重要修改（如，要統一將字元主高改到 1-2 像素寬的時候）。
製作一個良好提示的 TrueType 字體需要大量的工作，不過自 1990 年代以來，大量的添加字體提示的程序被開發出來給用戶使用，不過很多 TrueType 字體只有一些很不完善的提示，或者只有根據字體編輯器自動提示，導致各種不同結果。

Ming for 10646 (香港政府)
TrueType 系統包含一個虛擬機在字體內部執行程序，處理字形的"提示"(hints)。這些定義輪廓的控制點目的是在柵格化過程中減少的意外結果。 每個字形的提示程序算出顯示字形的像素大小，以及顯示環境中其他次要參數。
雖然和普通程序中的接受輸入和產品輸出一同，TrueType 提示語言的確具備一些程序語言的特性：：操作分支（IF 聲明），控制循環，一個時間的演算法（FOR, 和 WHILE-型聲明），變數（雖然這些只是內存中分給字體的一個區域中的簡單編號的標），並將代碼封裝到函數中，特別是精 delta hints 是最低級的控制，用來移動一個控制點或一個像素大小。
好的 TrueType 字形程序技術需要在一個字體中盡可能的使用一次參數定義（如，字元寬度，大寫字母高度，x 字高），這意味著盡量避免 delta 結構，這可以幫助字體開發者做一些重要修改（如，要統一將字元主高改到 1-2 像素寬的時候）。
製作一個良好提示的 TrueType 字體需要大量的工作，不過自 1990 年代以來，大量的添加字體提示的程序被開發出來給用戶使用，不過很多 TrueType 字體只有一些很不完善的提示，或者只有根據字體編輯器自動提示，導致各種不同結果。

図 7:処理時間評価に用いた PDF

計測結果を表 1 に示す。抽出したフォントデータによるフェイス単位での特定が従来手法の所要時間にあたる。また、参考までにテキスト抽出のみ、フォントキャッシュがある場合のラスタ時間も計測した。所要時間計測は gcc によってプロファイラを組み込んだ実行形式を作成し、実行後に gprof で計測した。それぞれ 100 回試行した平均時間である。

処理内容	処理時間(sec)	フォント抽出回数
フォント抽出まで(従来手法)	35.4873	1031
描画命令抽出まで	47.6259	1031
データベース照合まで(提案手法)	51.1539	1031
テキスト抽出のみ	0.0658	0
ラスタ処理(72dpi)	1.0925	2
PostScript 生成	6.4489	2

表 1: poppler-0。18 により計測した従来手法と提案手法の処理時間

4. まとめ、今後の課題

実験結果から、フォント抽出に要する時間約 35 秒に対し、グリフ描画命令抽出を加えた時間(約 48 秒)は 34.2%の処理時間増加、さらにハッシュ計算と SQLite3 によるデータベース照合を加えた時間(約 51 秒)は 44。1%の処理時間増加である。この結果から、フォント抽出に要する時間が最も長く、次いでグリフ描画命令抽出に要する時間､ハッシュ計算とデータベース照合による遅延は最も影響が小さいことがわかった。フォント抽出に要する時間は 1 回あたり(35.4873-0.0658)/1031 = 0.034 秒と見積もられる。フォントキャッシュの実装により、フォント抽出回数を 2 回までに削減することができれば､データベース照合まで処理する時間を 15 秒程度まで短縮できる可能性がある。ただし、この処理時間はラスタ処理時間や他の PDL への変換時間の数倍以上であり、単にグリフ特定を行うだけものとしては実用的な速度と言い難い。データベース照合をより高速なものに置き換えるなどの検討が必要である。

謝辞

本研究は科学研究費補助金　若手研究 B　課題番号 21700113 の補助をうけた。

参考文献

[1] Apple Computer, “TrueType Reference Manual”, http://developer.apple.com/fonts/TTRefMan/, 2011-09-25 閲覧
[2] Adobe Systems Incorporated, “PostScript Language Reference Manual”, Addison-Wesley, 1988.
[3] ISO/TC171/SC2, “ISO 32000-1, Document management – Portable document format – Part 1: PDF 1.7”, 2008.
[4] Hewlett Packard, “PCL 5 Printer Language Technical Reference Manual (Part I & II)”, Hewlett Packard, 1992.
[5] ISO/IEC JTC1/SC34/WG4 ISO/IEC 29500-1:2011 “Information technology -- Document description and processing languages -- Office Open XML File Formats -- Part 1: Fundamentals and Markup Language Reference”, 2011.
[6] ECMA International, “Standard ECMA-388: Open XML Paper Specification”, http://www.ecma-international.org/publications/standards/Ecma-388.htm
[7] International Digital Publishing Forum, “EPUB 3”, http://idpf.org/epub/30, 2011-09-08 閲覧.
[8] Jonathan Kew, Tal Leming, Erik van Blokland, “W3C Candidate Recommendation: WOFF File Format 1.0”, http://dev.w3.org/webfonts/WOFF/spec/, 2011-08-04 閲覧.
[9] ISO/IEC JTC1/SC29/WG11, ISO/IEC 14496-22:2009 “Information Technology – Coding of Audio-Visual Objects – Part 22: Open Font Format”, 2009.
[10] 鈴木俊哉, “電子文書中のフォントの特定とヒント制御”, 画像電子学会第 258 回研究会, 2011.