

電子文書中の TrueType グリフ照合とその高速化手法の検討

鈴木 俊哉

広島大学 総合科学研究科

〒739-8511 広島県東広島市鏡山 1-4-2 広島大学 情報メディア教育研究センタ

mpsuzuki@hiroshima-u.ac.jp

概要: PDF 中の TrueType フォントのグリフ単位での照合手法について、著作権・利用許諾に抵触しないデータベースを構築する手法を情報処理学会デジタルドキュメント研究会第 83 回研究会にて報告した。その際、PDF 処理の中でフォント抽出、グリフ描画命令抽出、SQL クエリの遅延によって、単純なラスタ処理の数十倍の処理時間となることを報告した。この遅延を削減するため、フォントキャッシュの導入とその効果について検討する。

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。字形情報交換の規格として、ISO 10036 が存在するが、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。

電子文書の一例として PDF[1]に埋め込まれたフォントは、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。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。

維蓮禮為雨讀請
維蓮禮為雨讀請

図 1: MingLiU の字形変更例

図 1に示すように変更前後では字形が大きく異なるが、ファミリー名は同一なため、識別が判断することが困難である。また、JIS2004 改訂のように一部の字形が変更されることもあり、フェイス単位での特定[3]では不十分な場合がある。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 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

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、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 PostScript[2]や PDF は特異的な仕様である。言い換えれば、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。

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 2に示す。PDF 中で必須でないものはグレーで示した。どちらも CID をインタフェースとする描画命令群オブジェクト(CIDFont)に文字符号との対応オブジェクト(CMap)を結合させたものである。

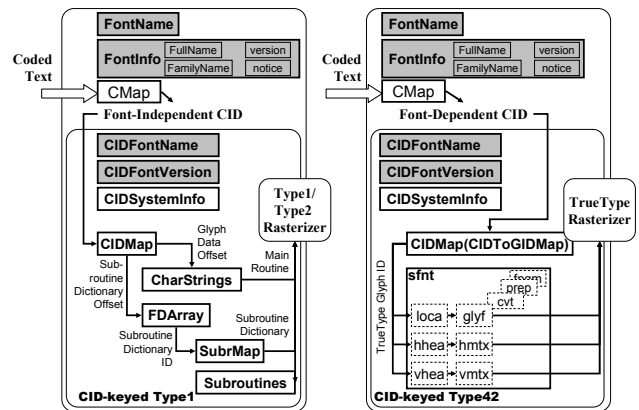


図 2: PostScript/PDF 中のフォントオブジェクト (Type1 型(右), TrueType 型(右))の構造

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、フォント名が欠けている場合があり、描画データをもとに特定せざるを得ない。

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 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。

この手法を単純に実装した場合、単純なラスタ処理の 50 倍以上の時間がかかり、実用に耐えない[4]。フォント抽出、描画命令抽出、DB 照合までの所要時間を比較した結果、増加した処理時間の 70%近くがフォント抽出に費やされていることがわかり、フォントキャッシュを実装して抽出回数を削減すればラスタ処理の 15 倍程度まで短縮できることが予想された。本稿ではフォントキャッシュを実装し、その効果を確認する。

3. 提案手法とその評価

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(図 1 の CMap のみ処理される)、描画データに関するオブジェクトは生成されない。本稿では、カプセル側オブジェクトの生成の際、同時に描画命令群オブジェクトも生成し、これをカプセル側オブジェクトの破壊まで維持することでキャッシュとして用いた。

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 でフォント名、バージョン、グリフ ID をキーとして描画命令ハッシュ値を値として持つようなデータベースを作成した。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。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、400MHz 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。

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。

処理内容	フォント キャッシュなし 処理時間(sec)	フォント キャッシュあり 処理時間(sec)
描画命令抽出まで	47.6259	0.8461
DB 照合まで	51.1539	7.5199
テキスト抽出のみ		0.0658
ラスタ処理(72dpi)		1.0925
PostScript 生成		6.4489

表 1: フォントキャッシュによる処理時間の短縮効果

計測結果を表 1 に示す。それぞれ 100 回試行した平均時間である。

4. まとめ、今後の課題

実験結果から、予想されていた目標値(ラスタ処理の 10 倍程度)よりも短縮効果が高く、8 倍程度まで処理時間を短縮でき、他 PDL への変換と同程度の処理時間となった。

プロファイルを分析した結果、描画命令抽出で打ち切った場合と DB 照合まで行った場合の処理時間の大半は SQLite3 のライブラリ中で消費されていたため、これ以上の短縮は DB 照合の回数自体を削減することを検討しなければならない。ただし、描画命令抽出までの時間はラスタ処理よりも短くでき、DB 照合を事後にバッチ的に行うためのインデキシングとしては有効であると言える。また、現時点では TrueType 描画命令のみ抽出しているが、フォント構造がより複雑な Type1 フォントを扱った場合の評価も必要であろう。

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] ISO/TC171/SC2, “ISO 32000-1, Document management – Portable document format – Part 1: PDF 1.7”, 2008.
- [2] Adobe Systems Incorporated, “PostScript Language Reference Manual”, Addison-Wesley, 1988.
- [3] 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.
- [4] 鈴木俊哉, “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”, 情処研報, SIGDD83-2, 2011.