Vol.2010-DD-76 No.5
2010/7/23

TrueTypeフォントのUVS拡張を用いたAdobe CID字形集合の代替処理の高速化

鈴　木　俊　哉[†1]

第67回デジタルドキュメント研究会にて、ページ記述言語PostScriptにおける標準的な字形指定番号であるAdobe CIDから、TrueTypeフォントのグリフへのマッピング方法について発表した。アドホックなマッピング情報を持たないよう、PostScript資源として提供されている情報のみでマッピングを決定しようとすれば、テーブル構築の際に0.5〜3秒程度の遅延が発生することを示した。この負荷はラスタ処理が数十秒から数分におよび高解像度の印刷の場合には無視できるが、モニタ表示の観点では問題となる。この負荷はPostScript資源をフォントごとに読み込んでマッピングを構築するためと考えられるので、PostScript資源ではなく、近年普及が進みつつあるTrueTypeフォントのUVSサポートを用いて、このマッピング処理を高速化する方法について検討する。

Fast mapping from Adobe CID space to TrueType GID by UVS info in TrueType font

SUZUKI TOSHIYA[†1]

In SIGDD67, the method to minimize the missing glyph in the translation of CJK TrueType font to CIDFontType2 object for Adobe CID glyph space was presented. The method improves the missing glyph issue of Ghostscript-8, the latency caused by using 3 tables (horizontal CMap, vertical CMap, and ToUnicode or ToCode mapping table) is not negligible for document browsing on the computer displays. In this report, a method to improve the latency by using OpenType cmap table format 14 (UVS table) which is introduced for Variation Selector in TrueType font. The parser of UVS table was implemented by PostScript for Ghostscript, and VM usage and time to parse UVS table were measured. From the experimental results, the parsing latency for UVS table is shorter than that for Unicode cmap table, and the VM usage is about 50% of previous method.

†1 〒739-8511 東広島市鏡山1-4-2　広島大学大学院総合科学研究科
Faculty of Integrated Arts and Science, Hiroshima Univ., Kagamiyama 1-4-2, Higashi-Hiroshima-shi, 739–8511 Japan

1. 背　　景

情報交換用に設計された符号化文字集合における、いわゆる「例示字形」と、印刷・印字処理系に設計された字形集合は、厳密に区別されなければならない。前者においては、文字符号が第一義的なものであり、例示字形は符号化文字集合を為す閉集合の中の要素を特定するために示される二義的なものだからである。これに対し、後者は、特定の処理系を用いてある字形を出力することが念頭に置かれており、必ずしも様々な処理系の間で情報を交換するために設計されてはいない。このため、前者はプレインテキストのような、文字符号のみによる利用が広く行なわれているが、後者は特定の印刷用データ体系の中での字形指定データとして利用され、字形指定データだけがプレインテキストのように交換されることは稀であった。

さて、ISO/IEC 10646は符号化文字集合であり、厳密な字形の規定を含まない。しかし、字形の指定をするための付加的な情報を符号化文字列の中に埋め込むための符号(Variation Sequence、以下ではVSと略す)がいくつか定義されている。ISO/IEC 10646の中で字形が規定されているのは数学記号やモンゴル文字の数文字など非常に限定された字形だけであり、それ以外の文字に対してVSを使用した場合、表示系は単にこれを無視すると定められている。これに対し、ISO/IEC 10646を利用した符号化方式の規格であるUnicodeでは、上記の他に日中韓統合漢字に対するVS(Ideographic Variant Sequences、以下ではIVSと略す)を登録制によって管理し、さらに未定義のVSの使用を禁止している。2007年には最初の申請としてAdobeによるAdobe-Japan1字形集合が登録され、現在は日本から汎用電子政府用文字集合の最初のグループが提案されている。

これまで、Adobe CID字形番号を直接に指定して文字符号よりもさらに詳細な字形を指定できるものは、高品位な印刷用データを生成するアプリケーションに限られていた。たとえば、事務文書を作成するほとんどのソフトウェアではJIS90字形とJIS2004字形を切り替えるUIを持たない。しかし、今後は安価なアプリケーションからでも間接的にAdobe CIDを指定することができ、また、Adobe CID字形番号を指定できないアプリケーションからの出力にも、他のアプリケーションによって挿入されたIVSが透過的に含まれることで、表示の際にAdobe CIDによる字形管理が必要となる可能性がある。

2. PostScript/PDF内部の符号化文字列処理

本節では、PostScript/PDF内部の符号化文字列処理について整理する。

2.1 Adobe CID字形集合

Adobe CID字形集合は、PostScriptおよびPDFで用いるために策定された字形集合である[1)]。各字形には16ビットのCID番号がわりあてられている。伝統的な利用方法とし

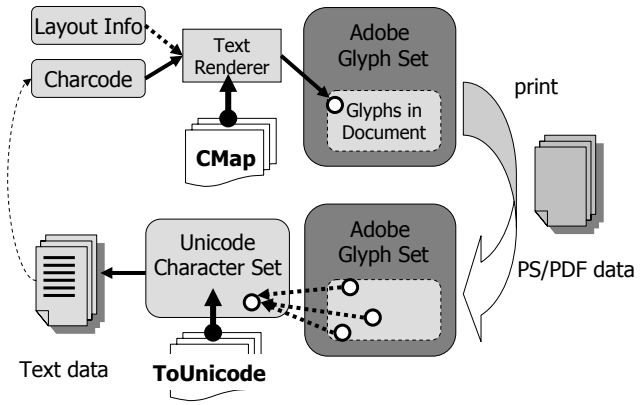


図 1 CMap と ToUnicode テーブルの役割

ては、各処理系の文字符号位置と CID 番号の対応関係を定義する符号表 (CMap) と組み合わせて用いていたが、文字符号と独立に CID 番号を直接指定して字形を特定することも可能である。この方式は、日本の漢字処理系のように、情報交換用文字集合における字形規定が緩やかで、様々な実装字形がありえるが、実際の利用においては細かな字形の差異が問題視される状況を考慮し、情報交換用文字集合では包摂して区別しないような差異をも区別するために導入された。当初、Adobe CID はほとんどの漢字が重複しているが符号位置が異なる文字符号をできるだけ小さいフォント資源で提供するための方式として導入された。ISO-2022-JP, Shift-JIS, EUC-JP など大半の符号位置を数学的演算で関係づけられる場合には問題ないが、ここに Unicode などを含めようとした場合、資源の流用が難しい。そこで、Adobe CID 字形集合を利用するために、文字符号に対し Adobe CID 字形番号をわりあてる対応表として CMap テーブルが定められ、これによって特定の文字符号で記述されたテキストを適切な字形で表示することが可能となる。その後、PDF に Adobe CID 字形集合が採用されるにあたり、文書の検索やプレインテキスト抽出などを可能とするため、CID で指定される字形に文字符号の属性をわりあてることが期待され、CMap とは別に ToUnicode テーブルとして配布されている。これらの関係を図 1 に示した。

CMap テーブルは特定のビットパターンに対して別のビットパターンをわりあてる対応表で、一般には文字符号ビットパターンに対して 16 ビットの CID 番号を対応づけている。基本的には最短一致のエントリが利用されるので、ブラフミ系文字のような符号化順序と印字順序を交換するような処理は実現できず、Adobe CID が定義されているのは符号化方式が比較的簡単であるが多数の字形が必要な漢字類に限られている。

Adobe-Japan1 に準拠する字形をデザインする際に、規格書印字に用いられた小塚明朝字形を参照するべきなのか、原規格となった JIS X 0208-1990 や JIS X 0213:2004 などを参照するべきなのか、Adobe-Japan1 規格書が原規格との対応づけを含まないためはっきりしなかったが、Adobe-Japan1 IVS 登録以後に更新された最新の Adobe-Japan1 規格書には原規格との対応表が含まれ、原規格を参照するべきことが明示された。

2.2 PostScript, PDF における VS 対応の可能性

PostScript, PDF においては、文字符号とグリフ指定は直結しておらず、その対応表を Adobe CMap 形式のデータで文書内に含めることができる。CMap は Adobe 製品に含まれるものを参照するだけでなく、ユーザが完全に独自に生成したものを含めることができる。これにより、PostScript PDF は、文書の中で複数のフォントや対応表を組み合わせ、独自の文字符号を表示できるフォントを処理系中に動的に生成することができる。本節では、PostScript, PDF の処理系の中に VS 対応のフォントを生成する可能性を考える。

VS を含む符号化文字列は以下のように処理しなければならない。

(1) UTF-8, UTF-16, UTF-32 などで符号化されたバイト列から、符号位置 1 個分のデータを読み込む。
(2) 読み込んだデータが VS でなければ、直前のデータをグリフ ID へ変換する。
(3) 読み込んだデータが VS であれば、直前のデータと共にグリフ ID へ変換する。

ここで、「VS なしの文字符号 1 個」または「VS ありの文字符号 1 個」をグリフ ID へ変換するのは既存のフォントで処理できる。問題となるのは、この 2 つを切り換えるため、直前の文字符号を記憶しておく処理である。

符号化文字列から、PostScript または PDF 処理系が CMap によって文字符号 1 個分のバイト列を切り出す処理は最短マッチであり、検索するバイトパターンに重複があってはならない。従って、1 個の CMap に、VS なしの符号位置-グリフ ID と、VS で修飾した符号位置-グリフ ID の両方を書き込むことができない。

2.3 OpenType における VS 対応

TrueType フォントについては、OpenType 規格 1.5 で cmap テーブル (TrueType フォントにおける符号位置-グリフ番号の対応表) が拡張され、VS に正式に対応した。その概略を図 3 に示す。規格書では ISO/IEC 10646 の VS ではなく、Unicode の VS が用いられることが定められている。OpenType における字形切り換えは cmap テーブルによって得たグリフ番号に対する置換として実装されているが、UVS への対応は cmap テーブルのみで実装されているので、いわゆる OpenType レイアウト機能への指定を与えなくとも利用できることが特徴である。

この拡張では、ある符号位置に対して

- 既定字形 (VS が付加されていない場合、または、フォントが対応していない VS が付加された場合に代替として表示するための字形)
- 非既定字形群 (特定の VS に対して用いる字形の集合)

の 2 種類の対応表が格納されている。既定字形は必ずしも非既定字形の 1 つである必要はない。また、現状の IVS 規格は完全に同一の字形に対して複数の VS を登録することを許

```
3 begincodespacerange    % 符号空間を 3 領域に分割することを宣言する
  <0000>      <D7FF>     % BMP の前半
  <D800DC00> <DBFFDFFF> % サロゲート領域
  <E000>      <FFFF>     % BMP の後半
endcodespacerange

1 beginnotdefrange       % 先に宣言した符号空間の一部の文字割当を禁止する
  <0000> <001f> 1        % ASCII の C0 領域は表示できないようにする
endnotdefrange

100 begincidchar         % 100 個単位で文字符号-CID 値の対応を列挙する
  <005c> 97
  <007c> 99
  <007d> 94
  ...
```

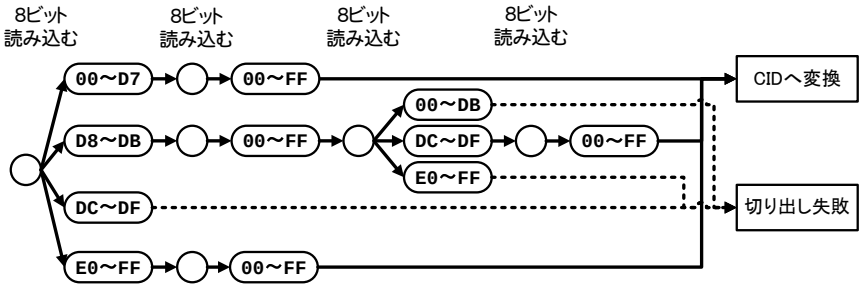
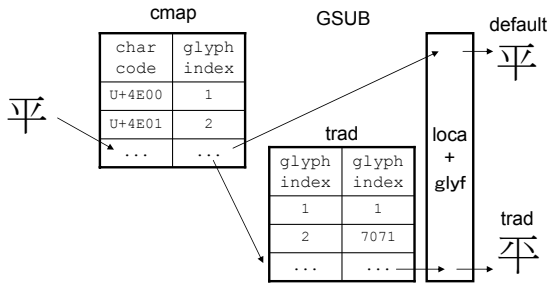


図 2 UTF-16 符号化の場合の CMap と、この CMap を用いた文字符号の切り出し手順

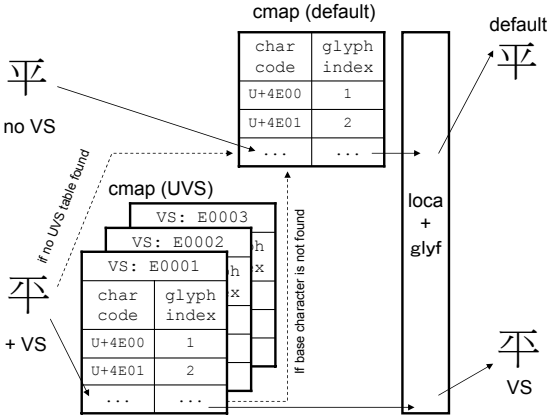
しており、この拡張でも非既定字形群の差異の大小を評価しない。そのため、ある非既定字形がフォントに含まれない場合、仮にその字形に近い非既定字形があっても既定字形で代替表示される。

3. Adobe CID 字形空間の CJK TrueType フォントによる代替表示

CFF OpenType 形式で Adobe-Japan1 字形集合を既に揃えている場合には IVS と CID の対応表を作成して cmap テーブルに追加するだけで良いが、CJK TrueType フォントは OpenType 拡張を含めても字形集合が少なく、また、符号位置も JIS 漢字といわゆる Microsoft CodePage 932 しか考慮されていない場合が多い。これらのフォントでは cmap



(a) Conventional glyph substitution in OpenType.



(b) UVS cmap since OpenType version 1.5.

図 3 従来の OpenType による異体字指定方法 (a) と、UVS cmap による異体字指定方法 (b)。

を追加しただけでは Adobe-Japan1 IVS の基底符号位置を全て表示できない可能性がある。

PostScript および PDF は従来の PostScript フォントや CFF OpenType 形式に相当する CIDFontType0 形式の他に、従来の TrueType フォントに相当する CIDFontType2 形式が利用可能である[2)]。CIDFontType2 形式のフォントは、一般に Adobe CID 空間を無視して設計されている TrueType フォントを描画形式の再変換をせずにプリンタに送信するか、あるいは文書中に埋め込むためのもので、その CID 字形番号は Adobe CID とは互換性がない (一般には TrueType フォントのグリフ番号がそのまま使用される)。しかし、CIDFontType2 オブジェクトはオブジェクト外に公開する CID 番号と内部の TrueType フォントのグリフ番号の対応づけを定義する CIDMap テーブルを持つことができるので、この

ToUnicode table	CMap table	$N_{\mathrm{CID}}^{\mathrm{CID}}$	$N_{\mathrm{CID}}^{\mathrm{CID}} - N_{\mathrm{UCS}}^{\mathrm{CID}}$
Adobe-CNS1-UCS2	UniCNS-UCS2-H	18698	425
Adobe-CNS1-UCS2	UniCNS-UTF16-H	18698	271
Adobe-GB1-UCS2	UniGB-UCS2-H	29143	518
Adobe-GB1-UCS2	UniGB-UTF16-H	29143	485
Adobe-Japan1-UCS2	UniJIS-UCS2-H	23057	13567
Adobe-Japan1-UCS2	UniJIS-UTF16-H	23057	7676
Adobe-Korea1-UCS2	UniKS-UCS2-H	18075	1019
Adobe-Korea1-UCS2	UniKS-UTF16-H	18075	1016

表 1 ToUnicode テーブルにより、PostScript および PDF 中での利用が予想されている CID 数 ($N_{\mathrm{CID}}^{\mathrm{CID}}$) と、CMap テーブルによって文字符号位置と対応づけられる CID 数 ($N_{\mathrm{UCS}}^{\mathrm{CID}}$) の対比。2 つの CID 数の差が、PostScript および PDF 中での利用の可能性があるが、Reversal CMap アルゴリズムではグリフを対応づけられない CID 数である。

対応表を適切に調整することで、Adobe CID と互換性のある CID を持った CIDFontType2 オブジェクトを生成することができる。

著者らは、Adobe-Japan1 IVS の登録以前に、Adobe が公開している字形・文字符号関連のデータベースにより Adobe 外の規格解釈を含めずに日中台韓の Adobe CID 字形空間を埋めるアルゴリズムについて検討を行ない、また、オープンソースソフトウェアで用いられている CJK TrueType フォントを用いてアルゴリズムの評価を行なった[3]。Adobe の提供する CMap テーブルは、文字符号位置から Adobe CID 字形番号への対応表であり、IVS の枠組では既定字形群に相当する。同様に、ToUnicode テーブルは、IVS の枠組では非既定字形群に相当する (ただし、ToUnicode テーブルは IVS 規格化以前に設計されたものであり、統合規準を越えた関係づけも含まれる)。CMap のみを使用して対応づけを行なう手法 (以下、Reversal CMap と呼ぶ) では、表 1 に示すように多数の字形が表示できないことがわかった。

著者らは ToUnicode テーブルを用いることでこの欠落する文字数を大幅に改善できることを明らかにしているが[4]、これによってフォント読み込み時に Reversal CMap に比べて 0.5〜2 秒の遅延が発生するため、高速な処理が求められる低解像度モニタ表示の場合には問題となり得る課題があった。

これは ToUnicode テーブルと TrueType 由来の cmap テーブルを結合して Adobe CID と Unicode 符号位置の対応データベースを作成することによる負荷と考えられる。そこで、より高速なデータベースとして、OpenType cmap table format 14 (以下、UVS テーブルと呼ぶ) を用いることを考える。ToUnicode テーブルを用いた方式の場合、PostScript/PDF の言語特有のデータと、TrueType フォント個別のデータを組み合わせる必要があるので、TrueType フォント 1 個ごとに処理が必要であるのに対し、UVS テーブルの場合は、標準化された符号化文字列と TrueType フォント内部のデータの 2 つしかないので、実装のカプセル化が容易であり、様々な PostScript 言語上のオブジェクトを参照しなくても良いため処理が高速化できる可能性がある。

本稿では、まず PostScript 言語によって UVS テーブルのパースを行ない、Adobe CID から Variation Selector を付加した既定文字を得る辞書オブジェクトを生成する時間と VM 消費量を計測した。

4. PostScript による UVS テーブルの処理負荷

4.1 実験環境およびフォント

計測は Centrino Duo 1GHz, RAM 2GB の GNU/Linux (kernel 2.6.30.4, GNU libc 2.10.2) 上で、Ghostscript 7.07 により行なった。

実験に用いたフォントは改変可能なライセンスで公開されている TrueType フォントを元に生成した。

WenQuanYi 文泉驛プロジェクト[5] によって作成されているフォント。基本的には中文用であるが、カナ、ハングル等を含めることによって CJKV 空間をできる限り埋めることを目指している。収録グリフ数 44960 個。

UMing Arne Göetje が台湾の Arphic がかつて公開した宋体・楷体フォントをもとに、CJKV 空間をできる限り埋めることを目指して作成していたフォント[6]。従って、基本的には繁体字用であるが、中華人民共和国用・香港用・台湾用に細かなグリフの差異を反映したデザインを目指している。収録グリフ数 27123 個。

sazanami 狩野宏樹氏がいわゆる和田研フォントをもとに作成したフォント。これについては著者が手作業で Adobe-Japan1-6 近似を行なっている[7]。JIS X 0208:1997 を基本とし、収録グリフ数 13584 個。

IPAex 情報処理推進機構 (IPA) が公開しているフォントで、JIS X 0213:2004 の文字集合をサポートしている[8]。IPAex ファミリは、JIS X 0213:2000 字形と JIS X 0213:2004 字形をサポートするため、OpenType の layout feature タグを持ち、さらにグリフ名を Adobe-Japan1 CID に似せてある。収録グリフ数 12218 個。

これらを、fontforge[9] によって Adobe-Japan1-6 CID の CFF OpenType に変換する。fontforge が TrueType フォントを CFF OpenType に変換する際、hwid(半角化)、fwid(全角化)、vert, vrt2(縦書き化) などの OpenType feature タグについて反映するが、trad, smpl, jp78, jp83, jp04, hojo, nlck などの OpenType feature タグは反映されない。また、Adobe-Japan1 IVS に関するデータベースは持っていない。従って、IPAex ファミリなど、異体字切り換えに用いる適切なグリフを持っていても、fontforge がそれを認識して UVS を追加するわけではない。そこで、fontforge の内部データ形式である SFD[10] に対して直接 UVS を挿入した上で、これを CFF OpenType に変換することによって、UVS cmap を含む CFF OpenType を生成した。さらに、この段階でグリフが欠落している CID に対して、ToUnicode テーブルによって代替表示の候補となりうるグリフを複製してわりあて、グリフを追加したものも作成した。

fontforge が CFF OpenType 用に生成する cmap は、グリフが実際にはフォントに含まれていない場合でも cmap テーブルに Unicode 符号位置と CID の対応を書き込み、グリフ

が存在しない問題に関しては CFF フォントレベルでのフォールバックを期待する。そのため、fontforge で生成した CFF OpenType は対象の CID 空間だけで cmap テーブルのサイズが決定され、読み込み時間はグリフ数によらない。

UVS テーブルは、Variation Selector が付加された文字符号からグリフ ID を決定するためのテーブルであるが、符号化文字列の中で全ての文字がグリフ ID を付加されているとは限らず、また、フォントが全ての Variation Selector に対して個別のグリフを提供するとも限らない。Variation Selector が付加された文字に対しても特別なグリフを割りあてず、規定のグリフで表示する場合が少なくないことを勘案し、UVS テーブルは

Default map cmap format 14 内部ではグリフ ID を記述せずに) 単に Unicode cmap を参照することで、規定の文字を表示させる表

Non-Default map cmap format 14 内部で基底文字符号位置と Variation Selector からグリフ ID への対応を明示的に書く表

の 2 つを持つ。Variation Selector 非対応の処理系が用いる Unicode cmap と共通な対応については、できる限り Default map を使うことで、UVS cmap のデータ量を削減することができる。しかし、これは単にフォントファイルのデータ量を削減しているだけであって、UVS つき Unicode 符号位置と Adobe-Japan1 CID のように連続してマップすることが難しい関係の場合、データ処理量が削減されるとは限らない。

4.2 実験結果

まず、VM 消費量の結果を表 2 に示す。上で述べたように、元の TrueType フォントから

- 単に fontforge によって変換したもの
- ToUnicode テーブルによって代替表示の可能性がある CID を埋めたもの

2 つの CFF OpenType を生成しているが、たとえば ipagex.ttf から生成したものは前者が IPAexGothic-uvs.otf、後者が IPAexGothic-uvs2.otf である。ただし、sazanami フォントの OTF 版については同一であるため、後者だけを用いた。

	OTF cmap size		Final VM usage		Δ VM usage	
Font	UVS	Unicode	global	local	global	local
IPAexGothic-uvs.otf	915	522	2201900	809331	+378516	+312855
IPAexGothic-uvs2.otf	5736	522	2475948	997879	+652564	+501403
UMingHK-uvs.otf	6737	522	2644032	1218987	+820648	+722511
WenQuanYiZenHei-uvs.otf	6740	522	2715008	1284139	+891624	+787663
UMingHK-uvs2.otf	11279	522	2794420	1314828	+971036	+818352
WenQuanYiZenHei-uvs2.otf	11585	522	2901344	1408608	+1077960	+912132
sazanami-uvs.otf	12816	522	2554564	1040962	+731180	+544486

表 2 実装した UVS テーブルのパーサの VM 消費量。処理開始の初期状態での VM 消費量は、global VM が 1823384 バイト、local VM が 496476 バイトである。

UVS のデータサイズが大きいほど VM 消費量も増えるが、400kB〜1MB 程度である。ToUnicode テーブルを用いた場合の VM 消費量は 1〜3MB であったことを考えると、VM 消費量が半分以下に抑えることができている。

次に、パースに要した時間を表 3 に示す。上で述べたように、UVS テーブルは単体で機能するわけではなく、他の Unicode cmap をフォールバックに用いるようにデザインされている。そこで、Adobe CID から Unicode 符号位置の対応表を作成する際に、UVS テーブルのみをパースした時間 (UVS) と、フォールバック用の Unicode cmap もパースした時間 (UVS + UCS) を計測した。また、比較のため、もとの TrueType フォントを ToUnicode テーブルを用いた手法で CID 化するのに要する時間 (UCS + ToUnicode) も示した。

	OTF cmap size		Parsing time		
Font	UVS	Unicode	UVS	UVS + UCS	UCS + ToUnicode
IPAexGothic-uvs.otf	915	522	0.143	0.647	1.655
IPAexGothic-uvs2.otf	5736	522	0.179	0.727	1.655
UMingHK-uvs.otf	6737	522	0.413	1.149	5.178
WenQuanYiZenHei-uvs.otf	6740	522	0.305	1.078	2.842
UMingHK-uvs2.otf	11279	522	0.446	1.187	5.178
WenQuanYiZenHei-uvs2.otf	11585	522	0.357	1.131	2.842
sazanami-uvs.otf	12816	522	0.292	0.838	1.085

表 3 実装した UVS テーブルのパーサの処理時間。UVS テーブルのみ読み込む時間 (UVS)、UVS テーブルとフォールバック用の Unicode cmap を読み込む時間 (UVS + Unicode)、同規模の TrueType フォントを Unicode cmap と ToUnicode CMap によって CID 化する時間 (UCS + ToUnicode)。

計測結果から、まず、UVS テーブル単体のパース時間は 0.2〜0.5 秒で、ToUnicode テーブルを用いた手法で問題となっていた 0.5〜2 秒の遅延を改善できることがいえる。UVS テーブルはデータ量は Unicode cmap よりも非常に大きくなる (10〜20 倍)、パースする時間は Unicode cmap よりも短い (半分以下) であることがわかる。

5. まとめ・今後の課題

本稿では、PostScript/PDF での文字列符号処理の中では直接 Variation Selector を付加した文字符号を処理できないことを問題とし、UVS テーブルを含むフォントから Adobe CID フォントを合成する方法と、その際の負荷 (VM 消費量、処理遅延) を調査した。また、これを ToUnicode テーブルを用いた Adobe CID フォント合成手法を比較し、負荷の改善の可能性を示した。

今後の課題として、本稿では UVS テーブルのパーサの実装と CFF OpenType による負荷の評価だけを行なったが、sfnt OpenType へ UVS を付加した場合の検討が必要である。特に、Adobe CID のような PostScript/PDF と強い結び付きがない Hanyo-Denshi IVS での負荷を予想するには sfnt OpenType での評価が適切と考えられる。

参 考 文 献

1) Adobe Systems Inc.: *Adobe Technote 5094: Adobe CJKV Character Collections and CMaps for CID-Keyed Fonts*, Adobe Systems Inc., San Jose (1998). http://partners.adobe.com/public/developer/en/font/ /5094.CJK_CID.pdf.
2) Adobe Systems Inc.: *Adobe Technote 5012: The Type42 Font Format Specification*, Adobe Systems Inc., San Jose (1998). http://partners.adobe.com/public/developer/en/font/ /5012.Type42_Spec.pdf.
3) suzuki toshiya, Yamada, T., Yamato, M. and Suzuki, H.: Emulation of Adobe CID Resources by CJK TrueType Fonts, *Document Numerique*, Vol.9, No.3-4, pp.17–43 (2006).
4) 鈴木俊哉：Adobe CID 字形集合の TrueType 代替可能範囲および処理負荷の評価，信学技報， Vol.108, No.156, pp.35–40 (2008).
5) Fang, Q.: 文泉驛. http://wenq.org/.
6) Götje, A.: *CJKUnifonts*. http://www.freedesktop.org/wiki/Software/CJKUnifonts.
7) 鈴木俊哉：字形の機械的な修正. https://www.codeblog.org/blog/mpsuzuki/200605.html.
8) IPA: IPA フォント. http://sourceforge.jp/projects/ipafonts/.
9) Williams, G.: *FontForge*. http://fontforge.sourceforge.net/.
10) Williams, G.: *Spline Font Database*. http://fontforge.sourceforge.net/sfdformat.html.