# A Guide to gs-cjk Project

中日韓のための、あるオープンソースプロジェクトへの誘い

山田泰司 <taiji@aihara.co.jp>

2003 年 6 月 19 日

**概要**

本稿では、Open Source プロジェクトの一つである「gs-cjk project」へみなさまを招待する。本プロジェクトは、PostScript および PDF 処理系のひとつとして世界で広く使われている Ghostscript(gs) において、中国語 (繁体字、簡体字)、日本語、韓国語のフォントを「清く正しく」扱うための方法を開発するプロジェクトである。Ghostscript は Artifex Software Inc., artofcode LLC, Aladdin Enterprises が、ソースコードを公開しながら世界のボランティアのプログラマとともに保守・開発しているソフトウェアである。我々 gs-cjk project チームから見れば、そのオリジナルの Ghostscript は「本家」ということになる。そして、本家 Ghostscript には、ライセンス形態の異なる 2 系統の著作物が存在し、ひとつは Aladdin Free Public License の最新版「AFPL Ghostscript」であり、ひとつは GNU General Public License(GPL) の最新版ではない「GPL Ghostscript」である。

近年、gs-cjk project は、GPL 系統の Ghostscript において、めでたく本家統合を達成した。これに続いて、GNU/Linux, BSD ディストリビュータ, Windows 等で gs のメンテナ達が追従し、それにより我々の成果が一般のユーザにも広く使われているようである。

しかし、2003 年現在、いくつかの成果について、最新の AFPL 系統の Ghostscript への本家統合は、未だ達成されていない。本家統合は本プロジェクトの目標のひとつであるが、ここでは、初心に返る意味でも一度立ち止まって Ghostscript における CJK 化の歴史を振り返りつつ、gs-cjk 技術を深く活用する方法、PDF 作成術、本家統合の戦略、プロジェクト参加などの諸情報を広く紹介する。

## 目次

# 1 あらすじ

Ghostscript は Open Source プロジェクトの Postscript 処理系の一つであり、世界中で広く使われている。もしあなたが、Microsoft Windows や Apple Mac OS で TeX を使っているならば、TeX からコンパイルした文書の印刷や PDF 化において、そのバックエンドとして Ghostscript は使われているであろうし、もしあなたが Unix 系 OS を使っているならば、TeX はもとより各種文書のプリンタへの印刷において、やはりバックエンドで Ghostscript は使われているはずである。

PostScript は Adobe Systems Incorporated による proprietary な「ページ記述言語」であるが、その仕様の詳細が従来より公開されており、かつ、それを利用したソフトウェアの開発や利用が許されているので、Ghostscript, gs-cjk プロジェクトのような活動が可能になっている。無論、Adobe からも PostScript 技術をソフトウェア上で利用する SDK(Software Development Kit) も配布・販売されている。Adobe 製品として、PDF 文書を閲覧する Acrobat Reader(無償)、PDF 文書の校正や PostScript から PDF へ変換する Acrobat と付属の Distiller、ディジタル画像編集 Photoshop、ベクトルグラフィックス編集 Illustrator 等が有名であり、これらの技術基盤として、PostScript はかねてより重要な位置付けにある。

PostScript はスタック型のプログラミング言語としても興味対象とされ、それを扱えるプログラマは今でも少なくない。TeX を扱う人であれば、PSTricks というグラフ等を描くパッケージを使えば、TeX ソース中で PostScript を直接扱うことも出来るので柔軟なグラフを文書内に埋め込ませることも可能である。本稿では、簡単な PostScript プログラムを組み、TeX や Illustrator、Word 等に取り込み可能な EPSF(encapsulated PostScript file) 形式のファイルを作成する方法、PSTricks で PostScript プログラムを組む方法を紹介する。ここで、PostScript の基本的な考え方を理解することで、あとにつづく Ghostscript, gs-cjk プロジェクトにおいて開発者がどういったものに取り組んでいるかが想像できよう。

さて、PostScript はページ記述言語であるので、それを最終的に可視化する形態 (コンピュー

タのディスプレイやプリンタ、組版のためのフィルムなど) へ「ラスタライズ」するための処理系が必要となる。Adobe 公式の PostScript 処理系は有償でプリンタベンダや商用 OS 等にライセンスされており、一般に手軽に直接扱う対象ではない。このような事情もあって、クローンである Ghostscript のような無償の PostScript 処理系が広く使われている。

PostScript はプログラミング言語ゆえに、その表現能力の高さから (特に Mac や UNIX で) 広く使われてきた一方で、その柔軟さゆえにポータビリティが失われることが多々あり、また、最終的に生成された PostScript ファイルのソフトウェア上の再編集も困難である。その反省から Adobe により生み出された形式が PDF(Portable Document Format) である。PDF 形式は、PostScript にあるような制御構文を排除し、文書の各々のオブジェクトに効率良くランダムアクセスでき、また、文書の表現力を高めるためのさまざまなフォントを埋め込むことができるので、たいへん可搬性が高い。現在 PDF は、ディスプレイ上の文書閲覧・校正から紙面への印刷までをカバーし、かつ、さまざまなプラットフォームで使用されている。

PDF 形式の文書を直接作成するアプリケーションも多々あるが、それでも取り込まれている EPSF 等の過去のソフトウェア資産を生かすには、PostScript から PDF への変換が可能であれば良い。そのような用途の PostScript 処理系が Adobe より提供されており、有償の Acrobat 付属の Distiller(蒸留器) がそれである。「蒸留」とはよくいったもので、世にある複雑怪奇な PostScript コードを「ろ過」して、すっきりとした PDF にするわけであろう。

PostScript をラスタライズする処理系として Ghostscript を説明したが、実際に Ghostscript は各種プリンタ用のドライバを備えており、多様なデバイスに出力することが可能になっている。PDF 形式も、対応している入出力デバイスの一つであり、Distiller のような用途に用いることができる。おそらく、現在は Ghostscript をこのような用途に使いたいユーザが多いのではないかと考えており、筆者も実際にそのように使っている。

さて、Ghostscript は、ライセンス形態により大きく分けて 2 系統存在する。ひとつは、Ghostscript を開発・保守している Artifex Software Inc., artofcode LLC, Aladdin Enterprises が定める AFPL(Aladdin Free Public License) により保護される gs8.x(2003/6 現在) と、GPL(GNU General Public License) により保護される gs7.0x(2003/6 現在) がある。AFPL gs がリリースされて一定の期間が経つと、よりアドバンテージのある最新バージョンを AFPL とするかわりに、古いバージョンの AFPL gs に若干の修正してそれが GPL gs としてリリースされる。ここでは、開発中の最新バージョン、つまり次期 AFPL gs 候補を「HEAD」と呼ぶ。ちなみに、HEAD のソースコードツリーも無論公開されており、誰でもアクセスすることが出来る。

筆者が関わっている gs-cjk プロジェクトは、本家 Ghostscript へ、中国語 (繁体字、簡体字)、日本語、韓国語の各種フォントを「清く正しく」扱える方法を模索し、それを具現化したコードを開発、そして統合することを目的としている。そして、近年、gs-cjk project は、GPL 系統の Ghostscript において、めでたく「CJK ユーザに支持されてきた技術」の本家統合を達成した。しかし、2003 年現在、AFPL 系統、つまり最新の Ghostscript へはすべての技術は統合されていない。これは、最新バージョンには、gs-cjk の技術と競合する (とされる) コードがあとから導入されてしまっていることが主たる理由である。

Ghostscript への日本語化については、歴史的にみて、これまでさまざまな有志達が実現し、実際に日本のユーザに使用されてきたが、本家統合は今だ達成されていない。これまで、gs-cjk は、日本語だけを考慮したコードを完全に排除し、中国語 (繁体字、簡体字)、日本語、韓国語に分け隔てなくコーディングするというポリシーで取り組んできた結果、日本だけでなく、中国、韓国のユーザにも支持され、GPL 系統の本家統合を達成したのだが、色々な事情で、HEAD には未達成なのである。よって、HEAD 統合の戦略を考えなければならない局面に立たされている上に、不幸なことに開発メンバがこのプロジェクトに時間を割けなくなって

いる。

本稿では、初心に返る意味で、Ghostscript における CJK 化の歴史を振り返りつつ、gs-cjk 技術を深く活用する方法、本家統合の戦略、プロジェクト参加などの諸情報を広くわかりやすく紹介したい。これにより、「本家に gs-cjk 技術導入をリクエストしてくれる方」、「gs の CJK まわりのバグ出しをしてくれる方」、「バグを直してくれる方」、「本家を巧みに説得してくれる方」が一人でも多く我々の前に現れてくれることを期待している。

## 2 ページ記述言語 PostScript と Ghostscript

「ページ記述言語 PostScript」をプログラミング言語からみた場合、その際だった特徴としてあげられるものは、ひとつは「スタック型言語」であるということと、もうひとつは「辞書」と呼ばれる名前空間を備えるということであろう。その仕様自体はとてもシンプルなもので、これらを理解することで、柔軟にプログラミングを行なえるだけでなく、基本的な算術演算も用意されているので柔軟なコンピュータグラフィックスを描かせることができる。

この言語を「ページ記述言語」としてみた場合、そのもうひとつの特徴は「デバイス非依存」であるということであろう。それゆえ、PostScript で描いたグラフィックスは、コンピュータのディスプレイ上、各種プリンタ、様々なファイル形式へ描くことができる。

ここではまず、簡単ではあるが具体的なプログラミングをすることで、PostScript とはいったいどういったものであるか理解してみよう。

## 2.1 PostScript プログラミング

**◆(例題 1) 算術演算および辞書** PostScript の算術演算オペレータを用いて、以下のような差分方程式 (Logistic 写像):

$$x_{t+1} = ax_t(1 - x_t)$$

の $a = 4$, $x_0 = 0.1$, $t = 1, \ldots, 10$ のときの値を出力せよ。

**◇(解答例)** PostScript インタープリタのコンソール上で、Logistic 写像をひとつの辞書 (<< キー 値 …>>) で以下のように定義し、

```
/Logistic <<
  /a 4
  /x .1
  /map {
    /x a x mul 1 x sub mul def
  }
>> def
```

以下のように繰り返し構文 (repeat) と辞書スタックオペレータ (begin,end) を使って、写像を次々と呼び出して、値を出力 (=) させればよい。

```
10 {
  Logistic begin
    map x =
  end
} repeat
```

すべてのコードはオペレランドスタック上で操作されるわけだが、手続き map のみに注目してこのスタック操作をみてみよう。以下は左欄がそのコードが一つずつ連なる様子、右欄がスタックの状態であり、その行末がスタックトップを表している。

```
/x                          % /x
/x a                        % /x  4
/x a x                      % /x  4  x
/x a x mul                  % /x  4x
/x a x mul 1                % /x  4x  1
/x a x mul 1 x              % /x  4x  1  x
/x a x mul 1 x sub          % /x  4x  (1 - x)
/x a x mul 1 x sub mul      % /x  4x(1 - x)
/x a x mul 1 x sub mul def  % x ⇐ 4x(1 - x)
```

ご覧の通り mul,sub は算術演算オペレータであり、スタックトップから 2 つのオブジェクトを取り出して演算を行ない、その結果をスタックトップに置いている。def は、スタックトップから二つのオブジェクトを「キー」と「値」としてカレント辞書に定義するオペレータである。

ちなみに、出力例は以下のようになる。

```
0.36
0.9216
0.289014
0.82194
0.585419
0.970814
0.113336
0.401964
0.961556
0.147865
```

**◆(例題 2) グラフィックスと座標系**　図1のようなロゴをページ出力する PostScript ファイルを作成せよ。

図 1: 例題 2 のロゴ

**◇(解答例)**　`x.ps` というファイル名で以下のようなテキストファイルを作成する。

```
%!
100 100 translate 10 10 scale
{
  newpath
  0 32 moveto
  13 15 lineto 0 0 lineto 2 0 lineto 18 19 lineto 8 32 lineto
  closepath fill
}
dup exec
32 32 translate 180 rotate
exec
showpage
```

そして、PostScript インタープリタやプリンタへこのファイルを指定し実行すればよい。

オペレータ `dup` はスタックトップの一つのオブジェクトを複製する。PostScript のような言語を知らない人は、上記のコードで何が複製されているのかわかるだろうか。

先に、`/map {...}`や `10 {...} repeat` のような形式について (直観的な理解を期待し) 特に説明しなかったが、`{...}`は「実行可能配列」オブジェクトである。ここではそれを複製し、複製されたひとつを実行 (`exec`) し、座標系を変更したのち、スタックトップに置いておいたもうひとつのそれを実行 (`exec`) している。つまり、例題の図は点対称な形の組合せなので、手続きを再利用しているわけである。手続きに名前を付けて、以下のようにしても同じ結果を得る。

```
%!
100 100 translate 10 10 scale
/X1 {
  newpath
  0 32 moveto
  13 15 lineto 0 0 lineto 2 0 lineto 18 19 lineto 8 32 lineto
  closepath fill
} def
X1 32 32 translate 180 rotate X1
showpage
```

さて、グラフィック命令についてであるが、ここでの基本はパス構築オペレータ (`newpath`,`moveto`,`lineto`,`closepath`)、ペイントオペレータ (`fill`)、ユーザ座標系変換オペレータ (`translate`,`rotate`) である。PostScript のユーザ座標系の既定値は、左下を原点とし、1/72 インチを 1 単位とする実数座標系である。つまりこの例では、左上の座標 $(0, 32)$ へカレントポイントを移動し、$(13, 15)$-$(0, 0)$-$(2, 0)$-$(18, 19)$-$(8, 32)$ の順にパスを構築、カレントパスをカレント色で塗りつぶしている。そして、原点を $(32, 32)$ へ移動し 180° 回転した座標系で同じ描画を行なっている。

**◆(例題 3) グラフィックス状態の保存と回復**　図2のようなロゴをページ出力する PostScript ファイルを作成せよ。

図 2: 例題 3 のロゴ

**◇(解答例)**　例えば、sun.ps というファイル名で以下のようなテキストファイルを作成する。

```
%!
100 100 translate 10 10 scale
{
  {
    gsave
    {
      newpath
      1 15 moveto 1 4 lineto 4 4 3 180 0 arc
      7 15 lineto 5 15 lineto 5 4 lineto 4 4 1 0 180 arcn
      3 15 lineto 1 15 lineto
      closepath fill
    }
    dup exec
    16 16 translate 180 rotate
    exec
    grestore
  }
  4 { 0 32 translate -90 rotate dup exec } repeat pop
}
[
  1 2 sqrt div 45 cos mul       1 2 sqrt div 45 sin mul
  -1 2 sqrt div 45 sin mul      1 2 sqrt div 45 cos mul
  16                            0
] concat
exec
showpage
```

そして、PostScript インタープリタやプリンタへこのファイルを指定し実行すればよい。

グラフィック状態オペレータ gsave,grestore に囲われた部分のコードは、前例と基本的に同様であり、「u」の形をふたつ組み合わせた「un」の形を描いている。さらにそれを 4 つ組み合わせた形を描く上で、「n」を描くために座標変換したものを、保存 (gsave) しておいた変換前の元の座標系に回復 (grestore) させている。座標系のカレント変換行列 (CTM: Current Transformation Matrix) の他にも様々な状態が保存および回復される。

ちなみに concat は、ユーザ座標系変換オペレータ (translate,rotate,scale) の一般形で、

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

の変換行列の 3 列目を除いた $[a\ b\ c\ d\ t_x\ t_y]$ と CTM の積にて座標形を変更するオペレータである。

**◆(例題 4) CJK テキスト出力と縦書き**　図3は、それぞれ左から中国語 (繁体字)、中国語 (簡体字)、日本語、韓国語、英語のテキスト出力である。これらのようなページ出力するそれぞれの PostScript ファイルを作成せよ。

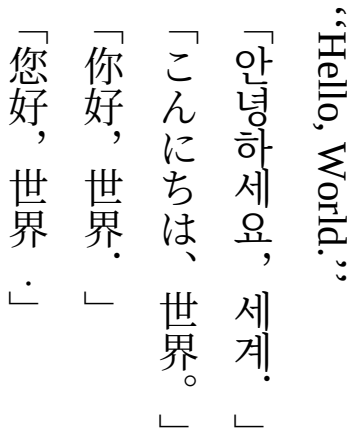



図 3: 例題 4 の出力例

**◇(解答例)**　左から順に、hello-t.ps, hello-c.ps, hello-j.ps, hello-k.ps, hello.ps として各国エンコーディングでテキストファイルを作成する。

```
%!
/MSung-Light--B5-V findfont 20 scalefont setfont
116 590 moveto（ここに台湾の big5 エンコーディングでテキストを書く） show
showpage
```

```
%!
/STSong-Light--GB-EUC-V findfont 20 scalefont setfont
116 590 moveto（ここに中国本土の euc-china エンコーディングでテキストを書く） show
showpage
```

```
%!
/HeiseiMin-W3-RKSJ-V findfont 20 scalefont setfont
116 590 moveto（ここに日本の shift_jis エンコーディングでテキストを書く） show
showpage
```

```
%!
/HYSMyeongJo-Medium--KSC-EUC-V findfont 20 scalefont setfont
116 590 moveto（ここに韓国の euc-korea エンコーディングでテキストを書く） show
showpage
```

```
%!
/Times-Roman findfont [
  0 -1
  1 0
  -.25 0
] makefont 20 scalefont setfont
116 590 moveto（ここに ASCII エンコーディングでテキストを書く） show
showpage
```

そして、PostScript インタープリタやプリンタへこれらのファイルを指定し実行すればよい。

往々にしてマルチバイトエンコーディングにて文字が指定される CJK フォントは、シングルバイトエンコーディングにて文字が指定されるラテンフォントとは、そのフォントの指定方法は本質的に異なる。

ラテンフォントでは**/フォント名**を `findfont` の引数にすれば良いのに対して、CJK フォントでは**/フォント名--エンコーディング名**を `findfont` の引数にする。「エンコーディング名」とは実際には、Adobe から配布されている **CMap リソース名**である。そして「フォント名」とは正確には **CID-keyed フォント名**である。

但し、歴史的には、日本語に関しては**/フォント名-エンコーディング名**全体で **OCF フォント**を指す場合もある。よって、下位互換性の為に、**/フォント名-エンコーディング名**を `findfont` の引数にしてもよい。

「エンコーディング名」の末尾が「`-V`」であるものは縦書きで用いられ、実際に縦に文字が並び、その上、いくつかのグリフは縦書き用グリフに置き換えられる。一方、通常の横書きでは末尾が「`-H`」となっている。

**◆(例題 5) カラーデバイスとシェーディング**　図4のような画像をページ出力する PostScript ファイルを utf-8 エンコーディングで作成 (しなくてもよいが、解答例をみてなんとなく理解) せよ。

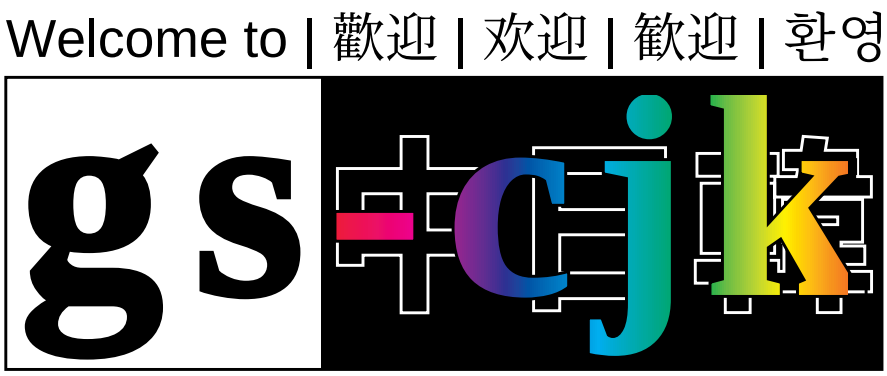



図 4: 例題 5 の画像

**◇(解答例)**　例えば、`welcome-to-gs-cjk.ps` というファイル名で以下のようなテキストファイルを作成する。但し解答例では、`(\ooo) show` のように 8 進数で文字列が指定されているが、そのまま 8 進数でも構わないし、utf-8 エンコーディングのままでも構わない。

```
%!
110 582 moveto
/Helvetica findfont 18.3 scalefont setfont
(Welcome to | ) show
/MSung-Light--UniCNS-UTF8-H findfont 18.3 scalefont setfont
(\346\255\241\350\277\216) show
/Helvetica findfont 18.3 scalefont setfont
( | ) show
/STSong-Light--UniGB-UTF8-H findfont 18.3 scalefont setfont
(\346\254\242\350\277\216) show
/Helvetica findfont 18.3 scalefont setfont
( | ) show
/HeiseiMin-W3--UniJIS-UTF8-H findfont 18.3 scalefont setfont
(\346\255\223\350\277\216) show
/Helvetica findfont 18.3 scalefont setfont
( | ) show
/HYSMyeongJo-Medium--UniKS-UTF8-H findfont 18.3 scalefont setfont
(\355\231\230\354\230\201) show
110 475 300 100 rectstroke
218 475 192 100 rectfill
/HeiseiKakuGo-W5--UniJIS-UTF8-H findfont 63 scalefont setfont
1 setgray
218 500 moveto
(\344\270\255\346\227\245\351\237\223) false charpath stroke
0 setgray
/Bookman-Demi findfont 98 scalefont setfont
110 500 moveto (gs) show
gsave
110 500 moveto (gs-cjk) false charpath clip
newpath
<<
  /ShadingType 2
  /ColorSpace /DeviceCMYK
  /Coords [218 475 409 475 4 2 roll]
  /Function <<
    /FunctionType 3
    /Functions [
      <<
        /FunctionType 2 /Domain [0 1] /N 1
        /C0 [0 1 1 0] /C1 [0 0 1 0] % red ... yellow
      >>
      <<
        /FunctionType 2 /Domain [0 1] /N 1
        /C0 [0 0 1 0] /C1 [1 0 1 0] % yellow ... green
      >>
      <<
        /FunctionType 2 /Domain [0 1] /N 1
        /C0 [1 0 1 0] /C1 [1 0 0 0] % green ... cyan
      >>
      <<
        /FunctionType 2 /Domain [0 1] /N 1
        /C0 [1 0 0 0] /C1 [1 1 0 0] % cyan ... blue
      >>
      <<
        /FunctionType 2 /Domain [0 1] /N 1
        /C0 [1 1 0 0] /C1 [0 1 0 0] % blue ... magenta
      >>
      <<
        /FunctionType 2 /Domain [0 1] /N 1
        /C0 [0 1 0 0] /C1 [0 1 1 0] % magenta ... red
      >>
    ]
    /Domain [0 1]
    /Bounds [60 120 180 240 300 5{360 div 5 1 roll}repeat]
    /Encode [0 1 0 1 0 1 0 1 0 1 0 1]
  >>
>> shfill
grestore
(-cjk) false charpath stroke
showpage
```

そして、PostScript インタープリタやプリンタへこのファイルを指定し実行すればよい。

ここで特に高度なのは、`shfill` の引数になっているカラーシェーディング辞書である。これを理解するには PostScript における「色空間」の扱いやそれに対する知識が必要になるので、特に詳しくここでは説明しない。参考文献 [8] を読めばこういった高度な色表現も可能であると言うことを示すにとどめる。

### 2.1.1 【まとめ】 オペランドスタック操作・制御オペレータ

PostScript により慣れ親しむには「オペランドスタック操作オペレータ」に慣れることである。このようなスタック型言語に不慣れなプログラマの為にスタック操作オペレータ、配列構築オペレータ、辞書構築オペレータをわかりやすく図5にまとめておこう。

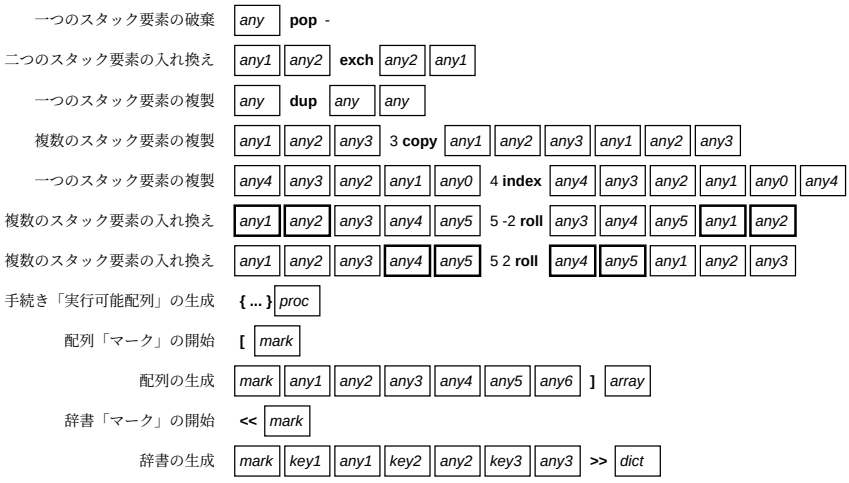



図 5: 代表的な、オペランドスタック操作オペレータ、配列構築オペレータ、辞書構築オペレータ

PostScript をプログラミング言語として扱うには「制御オペレータ」を理解することである。ここで制御オペレータをわかりやすく図6にまとめておこう。

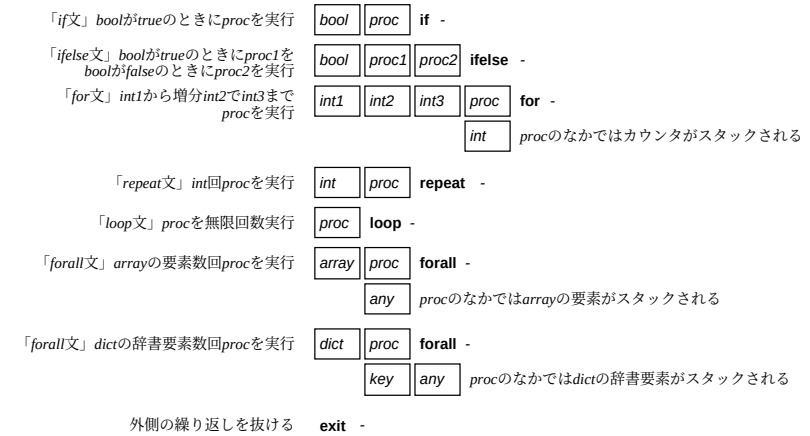



図 6: 代表的な制御オペレータ

## 2.2 EPS ファイルを組んでみよう

さて、PostScript で描いた一ページの図は、EPSF(encapsulated PostScript file) 形式に適合するように書くことで、Word や TeX 等の文書へ取り込むことができる。といっても何にも難しいことはない。基本的には描画領域 (バウンディングボックス) を整数値で書いておくだけである。

バウンディングボックス指定は、

`%%BoundingBox:` $ll_x\ ll_y\ ur_x\ ur_y$

のような形式である (*ll*: Lower Left, *ur*: Upper Right)。

一例だけ示そう。例題 2 の解答例を EPSF にしたい場合は以下のようにすればよい。

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 32 32
%%EndComments
{
  newpath
  0 32 moveto
  13 15 lineto 0 0 lineto 2 0 lineto 18 19 lineto 8 32 lineto
  closepath fill
}
dup exec
32 32 translate 180 rotate
exec
showpage
%%EOF
```

このように、「`%%`」という形式のコメントを用いて PostScript ファイルに情報を付加する作法を DSC(Document Structuring Conventions) といい、DSC 適合文書には、適合している DSC のバージョンに応じてファイルの先頭に「`%!PS-Adobe-3.0`」のようにバージョンを指定する。これは PostScript の言語レベル (最新はレベル 3) とは無関係であるので勘違いしないこと。ちなみに、現在参照が容易な DSC は参考文献 [7] のバージョン 3.0 である。

## 2.3 TeX のなかで PS プログラミングしてみよう

PSTricks という TeX のパッケージをご存知だろうか。これは、TeX もしくは LaTeX から PostScript を簡単に扱う為のマクロ集である。PostScript をよく知らなくても TeX が扱えるなら様々な図形が描けるようになっている。

PSTricks の基本的な使い方は、さまざまなウェブサイトで紹介されているのでここでは説明しない。むしろ筆者は、PostScript を直接 TeX のなかで扱いたいが為に PSTricks を愛用している。そういった応用例を紹介しよう。

**◆(例題 6) PSTricks のグラフと PostScript 算術演算** 図7のように、例題 1 で用いた Logistic 写像の時系列を PSTricks の`\psplot` マクロでグラフ描画せよ。

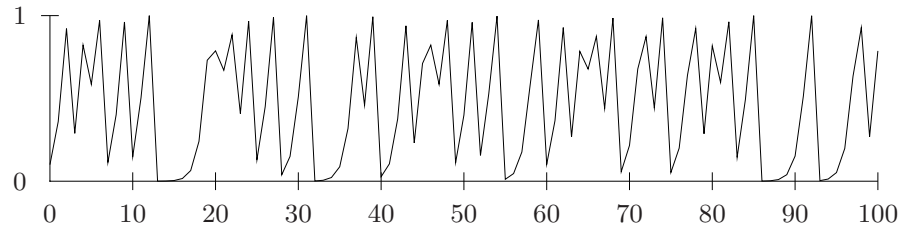



図 7: 例題 6 のグラフ

**◇(解答例)** PostScript による算術演算コードを`\code` マクロ内で定義することによって、図7のような時系列データを描くことが出来る。

```
\psset{xunit=.1cm}\psset{yunit=2cm}\psset{linewidth=.1pt}
\begin{pspicture}(0,0)(100,1)
  \psaxes[Dx=10,Dy=1](0,0)(100,1)
  \pscustom{%
    \code{%
      /Logistic <<
        /a 4
        /x .1
        /map {
          /x a x mul 1 x sub mul def
        } bind
      >> def
    }
    \psplot[plotpoints=101]{0}{100}{Logistic begin x map end}
  }
\end{pspicture}
```

**◆(例題 7) PSTricks と PostScript テキスト** TeX で円記号「\」を印字せよ。

**◇(解答例)** 文献 [33] で紹介されている TeX における円記号の印字方法:

```
\DeclareRobustCommand{\YEN}{{\tt\ooalign{Y\crcr\hss=\hss}}}
```

は、残念ながら円記号ではなく「`Y`」と「`=`」との重ね書きで実現されている。最終的な紙面への出力ではそれでも構わないが、例えば、TeX から PDF へ変換した段階で、PDF 上で「`¥`」を選択したときに「`Y`」と「`=`」になってしまう。

正しい円記号「\」を印字したければ、以下のようなマクロを用意し、

```
\newcommand{\putcidAJ}[1]{%
  \pscustom{%
    \dim{1zh}
    \code{%
      /KozMin-Regular /CIDFont findresource exch scalefont setfont
      0 0 moveto #1 glyphshow
    }
  }%
}
\newcommand{\pscharAJsysyen}{%
  \putcidAJ{291}\makebox[.5zw]{}}
```

`\pscharAJsysyen` とすればよい。

これと似たような問題を孕む文字は他にもたくさんあり、参考文献 [27] に詳しい。

## 2.4 オリジナルのフォントを作成してみよう

PostScript で基本となるフォントは Type1 フォントであるが、ここでは PostScript におけるフォントを理解する為に、大変簡単に作成することができる Type3 フォントを実際に作成してみる。

**◆(例題 8) Type3 フォント** 例題 2、3 のロゴを Type3 フォントにせよ。

**◇(解答例)** 例えば、`MyFont.pfa` というファイル名で次頁のようなテキストファイルを作成する。

解答例は、例題 2,3 の解答例をおよそ 1000 倍した座標系で描いているだけで描画コードはほぼ同一である。これは PostScript フォントとしてデフォルトのフォント変換行列 FontMatrix を `[ .001 0 0 .001 0 0 ]` としておく慣わしがあるからである。

フォント辞書に適切な値を指定したあと、**`/フォント名 フォント辞書 definefont`** で PostScript インタープリタににフォントを登録する。`definefont` はスタックにあるフォント辞書を吸い込まないので、この場合 `pop` してスタックに余計なオブジェクトを置かないようにすればよい。

このフォントを使用する場合、

```
%!
/MyFont findfont 100 scalefont setfont
100 600 moveto (Xs) show
showpage
```

などとすれば、図8のように出力される。

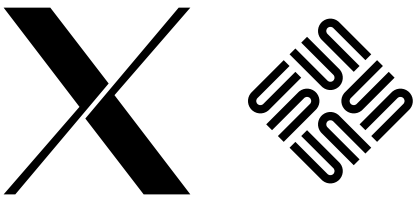


図 8: 例題 8 の使用例

```
%!PS-AdobeFont-1.0: MyFont
%%EndComments
/MyFont <<
  /FontType 3
  /FontMatrix [ .001 0 0 .001 0 0 ]
  /FontName (MyFont)
  /FontBBox { 0 0 800 800 }
  /Encoding 256 array
  0 1 255 {                              % i
    1 index exch /.notdef put            % (arr i /.notdef put)
  } for
  dup (X) 0 get /X put                   % (arr ((X) 0 get)charcode /X put)
  dup (s) 0 get /s put                   % (arr ((s) 0 get)charcode /s put)
  /CharProcs <<
    /.notdef {}
    /X {
      {
        newpath
        0 800 moveto
        325 375 lineto 0 0 lineto 50 0 lineto 450 475 lineto 200 800 lineto
        closepath fill
      }
      dup exec
      800 800 translate 180 rotate
      exec
    } bind
    /s {
      {
        {
          matrix currentmatrix
          {
            newpath
            25 375 moveto 25 100 lineto 100 100 75 180 0 arc
            175 375 lineto 125 375 lineto 125 100 lineto 100 100 25 0 180 arcn
            75 375 lineto 25 375 lineto
            closepath fill
          } bind
          dup exec
          400 400 translate 180 rotate
          exec
          setmatrix
        } bind
        4 {
          0 800 translate -90 rotate
          dup exec
        } repeat pop
      } bind
      [
        1 2 sqrt div 45 cos mul          1 2 sqrt div 45 sin mul
        -1 2 sqrt div 45 sin mul         1 2 sqrt div 45 cos mul
        400                              0
      ] concat
      exec
    } bind
  >>
  /BuildGlyph { % font charname
    1000 0                               % dx dy(width)
    0 0 800 800 setcachedevice           % llx lly urx ury(BBox) setcachedevice
    exch /CharProcs get exch             % CharProcs charname
    get exec                             % /X exec
  } bind
  /BuildChar { % font charcode
    1 index /Encoding get exch get       % font charname
    1 index /BuildGlyph get exec         % font charname BuildGlyph
  } bind
>>
definefont pop
%%EOF
```

## 2.5 ホストコンピュータと PostScript/PDF 処理系の関係

PostScript をどう利用するかは人それぞれであり、Windows で TeX や Adobe Distiller もしくは Ghostscript を扱っているユーザ、Mac や Unix で PostScript プリンタを所有しているユーザ、Unix で PostScript プリンタを所有しておらずプリンタドライバとして Ghostscript を使用しているユーザ、単に PDF を作成したいだけのユーザ、などさまざまなパターンが存在し、それらのすべてのパターンに適合した解説を行なうのは正直難しい。いや、むしろ、これらのユーザに当てはまらない、PostScript と関わりを持ったことのないユーザにそれを説明するのはかなり困難である。

まず筆者が思いつくこととして結論から述べると、Ghostscript を用いることで

- *Display PostScript* を搭載したオペレーティング環境ではなくても PS ファイルのディスプレイ上のプレビューが可能になる (但しここでは、*Display Ghostscript* には言及しない)。
- PostScript プリンタではなくても、PS ファイルをプリンタに印刷することができるようになる (高価な純正 PostScript プリンタほどプリンタ性能を出し切らないまでも取り敢えず安価なプリンタが使える)。
- PS ファイルから PDF ファイルを作成することができるようになる (有償の Adobe Distiller の取り敢えずの代用になる)。

といったメリットがあるということをまず述べておこう。その上、CJK-ready な Ghostscript であればそれらのメリットが日中韓で享受できるということも加えておきたい。

そこで、パソコンのようなホストコンピュータと、PostScript インタプリタのさまざまな関係を例示することで、それぞれの応用可能性を見出して頂ければ幸甚である。

- ホストコンピュータ ── 制御コード ──→ プリンタ

一般的な、ホストコンピュータとプリンタの関係である。ホストコンピュータはオペレーティング環境もしくはアプリケーションの機能を使ってプリンタに制御コードを送り込み、プリンタ用紙に印字する。稀にユーザが直接プリンタに制御コードを送り込む場合もあるだろうが、通常それは、プリンタドライバプログラマの仕事である。また、制御コードはプリンタによってまちまちである。

- ホストコンピュータ ─ ページ記述言語 → PDL コントローラ プリンタ

プリンタの制御コードのなかでは、それが「ページ記述言語 (PDL: Page Description Language)」に分類されるものもある。代表的な PDL に、HP の HP-PCL、Canon の LIPS、Epson の ESC/Page、それともちろん Adobe の PostScript があげられる。

- ホストコンピュータ ── PostScript ──→ PostScript プリンタ

そのような PDL コントローラを有するプリンタには高度な演算処理能力が要求され、価格も高価である。最近は、本格的な DTP の現場を除いて PDL の高性能化は要求されておらず、ホストコンピュータ側にほとんどの演算処理を任せてプリンタの低コスト化を実現している「ホストベースプリンタ」が一般的である。

- ホストコンピュータ ラスタ処理 ── ビットマップ ──→ ホストベースプリンタ

ホストベースプリンタにおいて、ホストコンピュータ側で処理すべき作業は、ベクトルグラフィックスやフォントのラスタライズ (RIP: Raster Image Processing) 等であり、プリンタにはその演算処理結果であるビットマップ情報が送られることになる。

- 


本格的な DTP では、PostScript がデファクトスタンダードとなっている。これは、(少なくとも従来までは) この業界のプラットフォームとして Mac、アプリケーションには Adobe の製品が主に使われているからであろう。

- 


また、Unix とユーザ層がかなり一致すると思われる組版システム「TeX」も PostScript と縁を切れない関係になっている。それも原因の一つとなっているであろうが、Unix でのプリンティングは PostScript がデファクトスタンダードとなっている。一方で、そのような状況を憂慮するグループが PostScript や Ghostscript に依存しない Unix のプリンティング環境を整備しつつあるようである。

- 


プリンタベンダが Adobe 純正の CPSI(Configurable PostScript Interpreter) や PostScript 互換 RIP を用いて、特定のオペレーティング環境向けへ「ソフトウェア RIP」を販売する場合もある。プリンタ性能を十二分に引き出しつつ PostScript という業界標準の PDL が使えるので、DTP の現場では重要な選択肢であろう。

- 


Ghostscript はフリーのソフトウェア RIP として使うことが可能である。しかも、gs の出力デバイスとしてさまざまなプリンタがサポートされていたり、プリンタベンダから gs 用のデバイスドライバが (GNU/Linux 用として) 提供されていたりする。新しい動きとしては、gs に標準搭載された HP IJS(Ink Jet Server) 向けドライバや、gs 用の Omni グループが提唱する PDC(Printer Driver Communication) 向けドライバがある [30]。

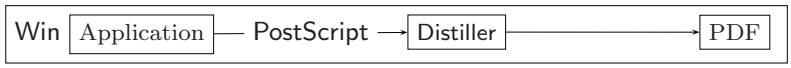
- 


- 


Mac では QuickDraw (GX)、Windows では GDI という、オペレーティング環境に備わっているグラフィック描画インターフェースをアプリケーションが使用しているので、プリンタベンダ純正もしくはオペレーティング環境標準搭載のプリンタドライバで対応したプリンタへ簡単に、画面で見ているものそのままを印字することができる。Unix では (ベンダ Unix では Display PostScript というのもあるが)、今後は Gnome-print、PDC、Xprint 等が整備されていくのだろう。当然のことながら、これに関しては Ghostscript への依存度をまったく無しにする方向性が正しいのだろうと筆者も思う。

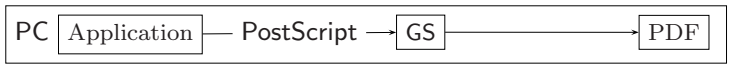
- 


Mac OS X では、QuickDraw GX の考え方を更に押し進めて、かつ、標準フォーマットを PDF とした「Quartz」と呼ばれる API がオペレーティング環境に備わっている。これは Mac が DTP 分野で根強く使用されていることから考えると至極自然な選択肢と思われる。なぜなら一般には文書の標準フォーマットである PDF は、印刷業界では RIP への中間フォーマットとしての地位を確立しているからである。PDF を採用するメリットのひとつとして、ホストコンピュータに搭載されているディスプレイ用フォントとしても印刷用フォントとしても使用できる高品位な OpenType フォントを PDF へ埋め込めることがあげられるだろう。これにより、指定されている日本語フォントを所有していない場合でも正しく閲覧・印刷できるようになる。

- 

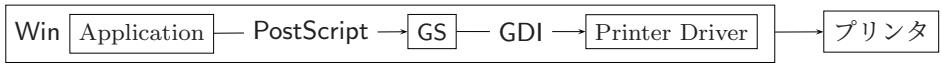
- 


PDF は、Adobe から無償で配布されている Reader によりクロスプラットフォームで閲覧することができ、また、Adobe 製品の Acrobat で文書の校正等も可能である。さらに、Acrobat に同梱されている Distiller は PostScript を PDF へ変換する一種の PostScript インタープリタであり、PDFWriter はオペレーティング環境に備わっているグラフィック描画インターフェース経由であたかもプリンタに印刷するかのように PDF を作成する仮想プリンタである。

- 

- 


Ghostscript を有償の Acrobat 同梱の Distiller のフリーの代替品として用いることで、PS ファイルから PDF ファイルを作成することができる。その場合、gs の出力デバイスは `pdfwrite` となる。また、Windows 上で Ghostscript を有償の Acrobat 同梱の PDFWriter のフリーの代替品としても用いることで、あたかもプリンタに印刷するかのように PDF を作成することができる。これについては次々節で詳しく述べよう。

- 


Windows 上で Ghostscript を使って仮想的なネットワーク PostScript プリンタを構成することができる。それには redmon という、やはり Ghostscript 開発関係者から配布されているフリーソフトを用いる。これについては参考文献 [31] に詳しいが、次々節で簡単に説明しよう。

## 2.6 Unix 系 OS で PS ファイルから PDF ファイルを作成してみよう

Unix への gs のインストールについてはここでは説明しない。なぜなら、ウェブを探せば誰かが書いてくれている内容だし、Unix といってもいろいろ環境が異なるからである。また、gs-cjk の要である CIDFnmap の設定方法も特に説明しないが、一般的なヒントとして「4.8.1 **【コラム】** gs-cjk CIDFnmap を最大限活用するためには」を参考にして欲しい。最低限の設定方法であれば参考文献 [30, 31] やウェブを探せば誰かが書いてくれているはずである。Unix 使いであればそれぐらい自分でできるはずだ。

さて、Unix で CJK-ready な gs がインストールされているものとしよう。`filename.ps` を PDF ファイルにするには以下のようにすれば良い。

```
% ps2pdf filename.ps
```

但しこれだと PDF バージョンが 1.2 になってしまうので、1.3 や 1.4 にしたければ、

```
% ps2pdf13 filename.ps
```

```
% ps2pdf14 filename.ps
```

のようにする。

## 2.7 Windows で PDF ファイルを作成してみよう

Windows への gs のインストールについてはここでは説明しない。なぜなら、ウェブを探せば誰かがバイナリを配布してくれているからである。また、gs-cjk の要である CIDFnmap の設定方法も特に説明しないが、一般的なヒントとして「4.8.1 **【コラム】**gs-cjk CIDFnmap を最大限活用するためには」を参考にして欲しい。最低限の設定方法であれば参考文献 [31] をお勧めする。但し、Windows 使いには CIDFnmap の記述すら敷居が高いかも知れないので、希望者には Windows XP 用の CJK-ready gs インストールキットを配布しよう。

さて、Windows XP で CJK-ready な gs がインストールされているものとしよう。

まず、「リダイレクトポートモニタ redmon」をインストールする。

① ウェブで `redmon17.zip` を探して、
② 内容物の `setup.exe` を実行する。

これで完了である。が、ここからがオペレーティング環境と redmon を結びつける作業である。

③［コントロールパネル］→［プリンタと FAX］→［プリンタのインストール］を選ぶ。
④［プリンタの追加ウィザード］で［次へ］。
⑤［このコンピュータに接続されているローカルプリンタ］のみをチェックして［次へ］。
⑥［新しいポートの作成］をチェックして、［Redirect Port］を選んで［次へ］。
⑦［リダイレクトポートの追加］ダイアログで［RPT1:］のまま［OK］。
⑧［プリンタの追加ウィザード］で PostScript プリンタドライバを選ぶ。例えば、製造元:［Fuji Xerox］、プリンタ:［FX DocuPrint C1250 PS-J5 v1.3］や製造元:［Apple］、プリンタ:［Color LaserWriter 12/600J］などでよい。
⑨ もしかすると［既存のドライバを使う］という確認がされるので (推奨) されている方にチェックして［次へ］。
⑩［プリンタ名］はのちのちの為、短い名前にしておく。例えばこの場合［ps2pdf］として［次へ］。
⑪［プリンタ共有］はこの場合、［しない］のまま［次へ］。
⑫ まだ設定は終ってないので［テストページを印刷しますか］は［いいえ］で［次へ］。

次からが、redmon と Ghostscript を結びつける作業である。

⑬［コントロールパネル］→［プリンタと FAX］で先ほど作成した［ps2pdf］プリンタのところで右クリックして［プロパティ］を選ぶ。
⑭［ps2pdf のプロパティ］ダイアログにて［ポート］タグを開く。
⑮［RPT1:Redirect Port］が選択されているので、そこで［ポートの構成］ボタンを押す。
⑯ このポートのリダイレクトプログラムとして［`c:\gs\gs7.06\bin\gswin32c.exe`］を指定 (パスは環境に応じて書き換えること)。
⑰ プログラムの引数として［`-Ic:\gs\gs7.06\lib;c:\gs\fonts\;c:\Windows\fonts -q -dNOPAUSE -dBATCH -sDEVICE=pdfwrite -sOutputFile="%1" -dCompatibilityLevel=1.4 -dSAFER -sPAPERSIZE=a4 -c .setpdfwrite -f -`］を指定 (フォントパスは環境に応じて書き換えること)。
⑱ 出力の形態はこの場合［ファイル名を尋ねる］を選択。
⑲ そして［OK］。

これで、Word 等 Windows のアプリケーションからこのプリンタ「ps2pdf」へ印刷すると、［ファイル名を指定して保存］ダイアログが開かれるので、保存するフォルダやファイル名を指定して保存する。これにより作成された PDF ファイルは、GDI とベンダの PostScript プリンタドライバにより作成された PostScript コードを、redmon が Ghostscript へポートリ

ダイレクトし、Ghostscript の pdfwrite ドライバが redmon から指定されたファイルへ PDF として出力していることになる。

ちなみに、「リダイレクトポートモニタ redmon」をアンインストールするのは、多少困惑するかも知れないが、リダイレクトポートモニタを使用しているすべてのプリンタと作成したリダイレクトポートを削除すれば、redmon をアンインストールすることができる。

## 2.8 Windows をネットワーク PostScript プリンタに見立ててしまおう

ここでの解説は参考文献 [31] で紹介されているものとほとんど同じである。但し、その文献では Windows に lpd サービスを導入する方式を紹介されているので、こちらでは smb プリンタ共有で使う方式を紹介するに留める。

さて、Windows XP で CJK-ready な gs がインストールされているものとしよう。また、先の ps2pdf プリンタを設定したものとほぼ重なるが、今一度説明しよう。

まず、「リダイレクトポートモニタ redmon」をインストールする。

① ウェブで `redmon17.zip` を探して、

② 内容物の `setup.exe` を実行する。既にインストールされていればその旨が通知される。

これで完了である。が、ここからがオペレーティング環境と redmon を結びつける作業である。

③［コントロールパネル］→［プリンタと FAX］→［プリンタのインストール］を選ぶ。

④［プリンタの追加ウィザード］で［次へ］。

⑤［このコンピュータに接続されているローカルプリンタ］のみをチェックして［次へ］。

⑥［新しいポートの作成］をチェックして、［Redirect Port］を選んで［次へ］。

⑦［リダイレクトポートの追加］ダイアログで［RPT2:］などのまま［OK］。

⑧［プリンタの追加ウィザード］で PostScript プリンタドライバを選ぶ。例えば、製造元:［Fuji Xerox］、プリンタ:［FX DocuPrint C1250 PS-J5 v1.3］や製造元:［Apple］、プリンタ:［Color LaserWriter 12/600J］などでよい。

⑨ もしかすると［既存のドライバを使う］という確認がされるので (推奨) されている方にチェックして［次へ］。

⑩［プリンタ名］はのちのちの為、短い名前にしておく。例えばこの場合［ps2win］として［次へ］。

⑪［プリンタ共有］はとりあえず、［しない］のまま［次へ］。

⑫ まだ設定は終ってないので［テストページを印刷しますか］は［いいえ］で［次へ］。

次からが、redmon と Ghostscript とプリンタを結びつける作業である。

⑬［コントロールパネル］→［プリンタと FAX］で先ほど作成した［ps2win］プリンタのところで右クリックして［プロパティ］を選ぶ。

⑭［ps2win のプロパティ］ダイアログにて［ポート］タグを開く。

⑮［RPT2:Redirect Port］等が選択されているので、そこで［ポートの構成］ボタンを押す。

⑯ このポートのリダイレクトプログラムとして［`c:\gs\gs7.06\bin\gswin32c.exe`］を指定 (パスは環境に応じて書き換えること)。

⑰ プログラムの引数として［`-Ic:\gs\gs7.06\lib;c:\gs\fonts\;c:\Windows\fonts -q -dNOPAUSE -dBATCH -sDEVICE=mswinpr2 -sOutputFile="\\spool\cc600px" -dSAFER -dNoCancel -sPAPERSIZE=a4 -`］を指定 (`cc600px` はパソコンに実際に繋がっているプリンタ名に書き換えること、フォントパスは環境に応じて書き換えること)。

⑱ 出力の形態はこの場合［プログラムに任せる］を選択。

⑲ そして［OK］。

これで、Word 等 Windows のアプリケーションからこのプリンタ「ps2win」へ印刷する

と、最終的にはプリンタ「cc600px」へ印刷される。これにより印刷されるものは、GDI とベンダの PostScript プリンタドライバにより作成された PostScript コードを、redmon が Ghostscript へポートリダイレクトし、Ghostscript の mswinpr2 ドライバが再び GDI 経由で`\\spool\phaser750` のようなプリンタスプーラへ出力していることになる。お気づきだろうが、このような使い方はまず嬉しくない。このプリンタ「ps2win」の使い道はこれをネットワーク PostScript プリンタとして利用してはじめてその価値がある。そこで、［プリンタ共有］を［する］にし、Unix からのこのプリンタへの出力する方法を簡単に紹介しよう。

「ps2win」プリンタを導入した Windows マシンの IP アドレスを仮に「192.168.0.1」としよう。Unix マシンから解決できる名前が既にあるなら名前でもよい。そして、もっとも簡単な方法は、Samba クライアントを使って、

```
% echo 'print filename.ps' | smbclient \\\\192.168.0.1\\ps2win -P
```

とすることである。

この方法を基本として、printcap, lp, LPRng, CUPS 等、使用している Unix 系 OS に備わっている印刷環境に設定を追加するのは容易であるので割愛する。また、Samba を使わずに Windows 側に lpd サービスをエントリする方法であれば、参考文献 [31] に詳しい。

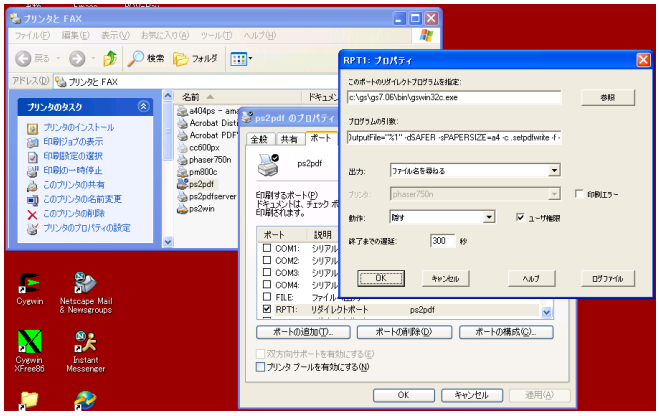


図 9: redmon, gs による ps2pdf プリンタの作成

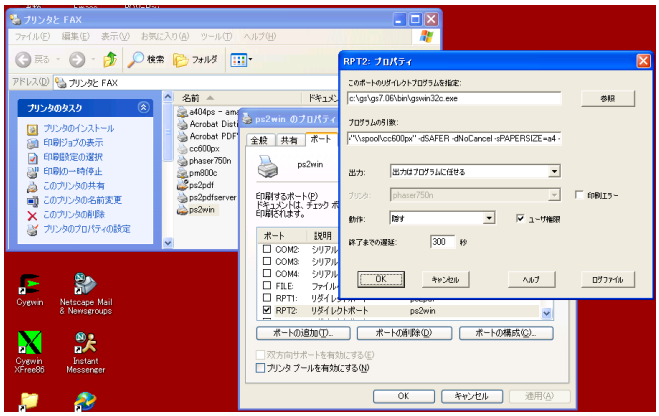


図 10: redmon, gs による ps2win プリンタの作成

# 3 Ghostscript における CJK 化の歴史

本節では、gs-cjk へ至る Ghostscript における CJK 化の歴史について、筆者の知る範囲で解説する。

Ghostscript 日本語化の先駆けは片山紀生氏による OCF ベースの日本語化パッチである。これは gs-cjk 以前まで、いや、今でもそれを基本とした日本語化パッチが使われている。日本語 OCF フォントは CID フォントに取って代わられる、既に廃れている種類のフォントであるのだが、日本語化に関しては CID ベースの日本語化が gs-cjk 以前はなかった。一方で、韓国語化に関しては CID ベースのパッチが gs-cjk 以前にも存在した。但し内部で、さまざまなフォント形式に対応する為の FreeType2 と文字コード変換ライブラリ iconv を用いていた為、これ自体を中国語・日本語化するには多少問題がある。なぜなら、文字コードとグリフコードは根本的に相互変換不可能であり、特に、比較的混沌としている日本語の漢字や記号類の文字コードとグリフを対応づけるには細かな多くの障壁があるからである。

OCF ベースの日本語化パッチはその後、淺山氏、松田氏、田中氏、鈴木氏等により発展し、近年では「VFlib による日本語化パッチ」と呼ばれ、さまざまなフォントを扱う為に VFlib2 やさらには FreeType1 を内部で用いている。これが本家統合を目標にした史実があるのか否か筆者は知らないが、この延長上に中国語・韓国語化を軌道に載せられない理由は、「OCF ベースであるから」である。

Adobe 公式に定義される OCF フォントは、日本語のみである。しかし市場には中国語 OCF フォント・韓国語 OCF フォントは存在するようであるが、フォントの実装は各フォントベンダ依存で、もし多言語を扱える VFlib3 等に対応させたとしても、「OCF のコンポジット構造」を実フォントに併せて再現するのは多大なコストが掛かる。

しかも、PostScript では既に OCF フォントは廃れ CID フォントに移行済みなのであるのでそれをやる動機も薄いのは明らかである。一方で、日本語 PDF では CID フォントがコア技術として必要な為、VFlib による日本語化パッチでは、なんと「OCF による CID エミュレーション」なるものが開発されていた。「大は小を兼ねる」というが、この場合、小から大をエミュレーションしようとしているので所詮無理な話である。しかし、CID フォントに含まれる多彩なグリフのなかでも使用頻度の高いグリフは OCF フォント内にも含まれる為、特に日本では、根本的な CID 化に対する要求は薄かったように思われる。

これでわかるように、普段日常で使っている日本語の文字については OCF 技術ベースでもグリフとしてはとりあえず十分なのである。gs-cjk のサイト [6] にも書かれているように、「Ghostscript に対する VFlib パッチと gs-cjk パッチは独立であり、gs-cjk が VFlib に取って代わるものではない」というのは、こういった事情もあるのであるが、特に Unix 系ディストリビューションでは VFlib パッチを捨てる方向に動いてしまった。これ自体は gs-cjk 開発陣はそれほど好ましいとは考えていなかったが、gs-cjk 普及にひと役かったことも確かである。その先駆けとして、Debian JP の武氏の貢献度が高い。氏は、gs-cjk 前身である筆者の gs6.01 用 CJK パッチの頃から Debian への導入準備を開始し、gs-cjk 普及へ大きく貢献した。

同時期に特に中国の GNU/Linux ディストリビュータも、筆者の gs6.01 用 CJK パッチ導入に精力的だった。筆者の知る限り、中国独自の Ghostscript 中国語化パッチは存在しないようなので、中国での GNU/Linux 普及のタイミングと Ghostscript CJK 化への要望が時期的にマッチしていたのではないかと思う。

さて、筆者の gs6.01 用 CJK パッチとは正確には CJK TrueType フォントを CID フォントのように見せかけて使えるようにする技術である。この前身は、鈴木秀幸氏の日本語

TrueType を CID フォントのように見せかけて使えるようにするパッチである。筆者はこれを CJK 化し、加えて、「縦書き用グリフ」を TrueType フォントから拾って CID グリフ空間へ割り当てるアルゴリズムを開発した。これには CJK すべてに一つのアルゴリズムを適用することを課し、TrueType グリフから CID グリフ変換アルゴリズムと同様、膨大な検証作業が必要であった。現在、gs-cjk で使用されているアルゴリズムも実装は違えども基本的に筆者によるアルゴリズムを基盤としている。

一部で「Ghostscript でははじめから CID フォントを扱えるようになっている。gs-cjk 化は TrueType を CID フォントのように見せかける技術と CID フォントを簡便に定義できる技術 (CIDFnmap) の部分」などということを触れ回っている方々がいるが、それは多少事実と異なる。筆者を除く gs-cjk 開発メンバの貢献には、CID フォントは高価である故、TrueType からフェイクした CID フォントを使った、Ghostscript で安定した CID フォントを扱う為の多数の修正も含まれている (それは大変苦痛を伴う、込み入った作業であったようだ)。今、そのほとんどが Ghostscript の HEAD にも取り込まれている。特に、大和氏、鈴木俊哉氏、斉藤氏、狩野氏等の貢献度が高い。つまり、OCF ベースの VFlib による日本語化パッチは長らく使われてきたものであるので、少なくとも安定度は高かったが、全面的に CID へ移行した gs-cjk では、CMap 処理・縦書きを含む動作の安定度が大変低かったのである。ここで著名な影響力のある方によって流布されている誤解を訂正させて頂く。

さて、先ほど日本語 OCF ベースのフォント定義技術が、gs-cjk により捨てられつつあることを述べたが、OCF と CID は独立したものであり、日本語 OCF フォントは将来の Ghostscript においても使用可能な状態を保たなければならない。OCF は Original Composite Font という名が示す通り、使用されるエンコーディングに応じたフォント固有の構成方法が採られており、CJK において一元的な開発は出来ない。一方で、日本語 OCF フォントを所有する PostScript プリンタのように Ghostscript を扱わなければならない場面も稀ではあるが存在する。これにより開発されたのが、筆者による「CID による日本語 OCF エミュレーション」である。これは、Ghostscript とは独立した成果とした。驚いたことにこの期に及んで『あるタイプの OCF では Ghostscript のバグが露呈する』ということが判明し、これは鈴木俊哉氏により修正された。

gs-cjk の本家統合の視点で見た場合、最も重要なテーマは「一国の都合で開発したものでは統合はあり得ない」ということであろう。その点については gs-cjk は当初から技術的にはクリアしており、「一国の都合の匂い」がする提案が成された場合には、少なくとも筆者は不快感を隠さなかった。加えて、ウェブサイト、ドキュメント類、サンプル類の CJK 化も必要であり、本質的ではないこのような作業を筆者は特に神経質に扱った。しかし、現在では既にアジア各国で十分認知されているので、現在、特にこの部分の作業については筆者は何も活動していない。また、本家統合の視点でのさらに重要なテーマとして「本家との交渉」であろう。筆者ははずかしながらこの分野においてはまったく貢献しておらず、鈴木俊哉氏がほぼ一手に引き受けた。現在、gs-cjk 化された Ghostscript をそのまま、パッチなどあてずにユーザが使えるのは最大の功労者、鈴木俊哉氏のおかげである。

図11に大まかな Ghostscript CJK 化年表を記した。

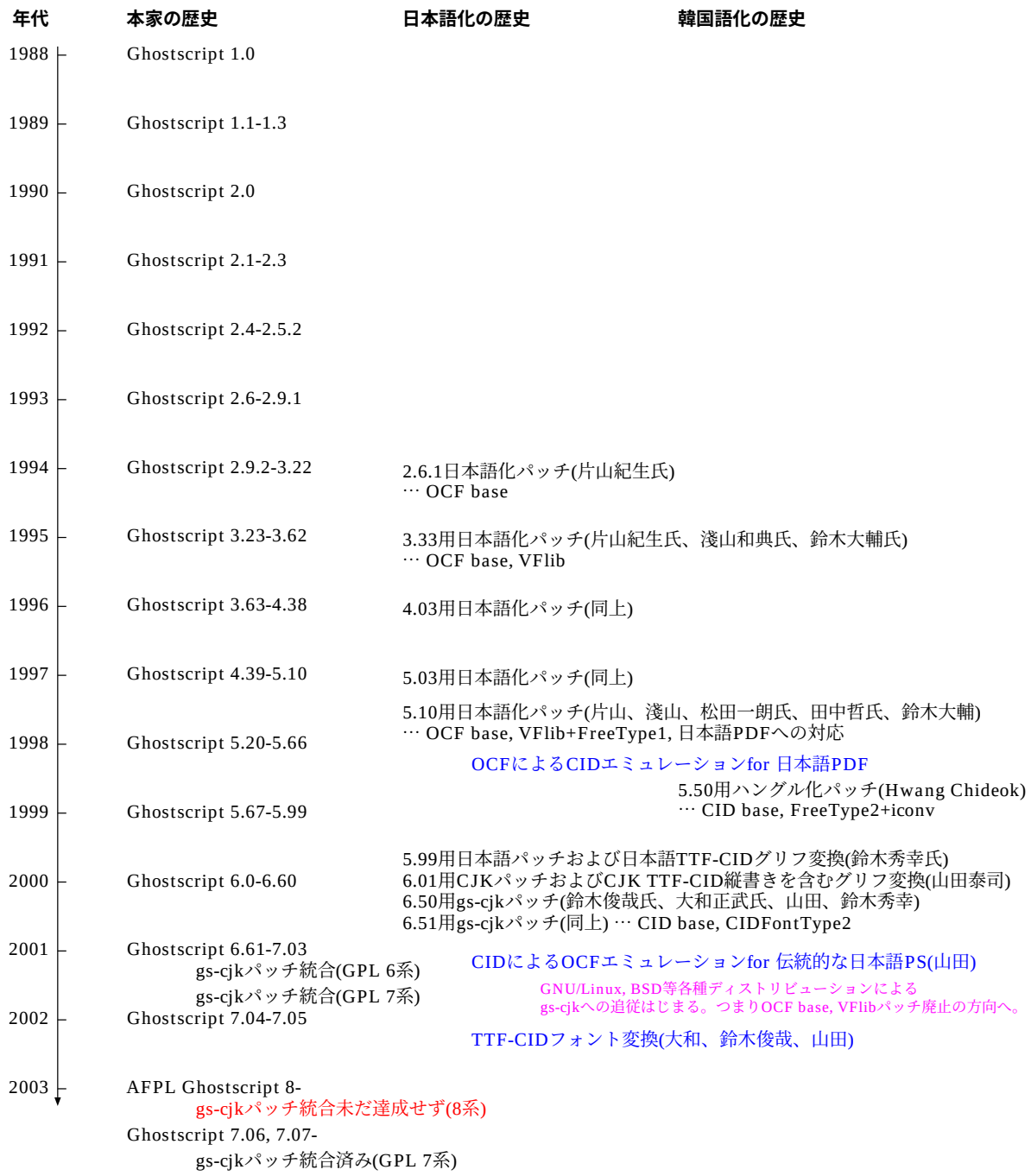



図 11: gs-cjk へ至る Ghostscript における CJK 化年表

# 4 CJK リソースと gs-cjk

## 4.1 CJK フォント

ここでは、これまでにここで登場している各種フォント形式について、特に CJK に関するアウトラインフォントについて、簡単ではあるが、まとめておく。

OCF かつてのマルチバイト用 PostScript プリンタフォント。正確には、最大 256 グリフをもつ PostScript Type1 フォント群を、使用されるエンコーディング方式に応じて Type0 フォントとして混成したフォントを指す。印字するときに文字列に応じて自動的に構成されているフォントへディセンド (下降) していくため、シフト JIS、日本語 EUC のようなダブルバイト、マルチバイトエンコーディングにて 256 を越えるグリフが扱えるようになる。標準装備として、PostScript Level2 プリンタの `Ryumin-Light`, `GothicBBB-Medium`(モリサワ) が有名。後述の日本語の CID グリフセットは、これらをもとに `Adobe-Japan1`, `Adobe-Japan2` が定められた。

TrueType Font Apple と Microsoft が策定したディスプレイ用アウトラインフォント。低解像度向けのビットマップフォントを内蔵することが出来、Windows において CJK フォントに関しては未だに主流である。プリンタ用フォントと比べ安価で、オペレーティング環境に同梱されているので、印刷にも使用したいという要望は根強くある。標準装備として、Windows の `MingLiU`(繁体字), `SimHei`, `SimSun`, `SimSun-18030`(簡体字), `MS-Mincho`, `MS-Gothic` (日本語) `Batang`, `Gungsuh`(韓国語) が有名。グリフセットはベンダ依存で、基本的には情報交換用文字コード Big5, Johab, PRC(People Public of China), ShiftJIS, Unicode, Wansung でグリフにアクセスする。Unicode 以外の TrueType フォントは現在入手困難。

CIDFont OCF フォントに代わる、マルチバイト用 PostScript プリンタフォント。65536 ものグリフを扱うことが出来、テキストのエンコーディングに応じて Adobe CMap と組み合わせて使用される。CIDFont と言った場合には CIDFontType0 のことを指すことが多く、別に、TrueType フォントを PostScript 上で CIDFont として扱えるように辞書としてくるんだ形式を CIDFontType2 と呼ぶ。同様に、TrueType フォントを非 CID フォントとして扱えるように辞書としてくるんだ形式を Type42 フォントと呼ぶ。Type42 も CIDFontType2 も流通はしておらず、PostScript の内部的な形式と言える。CIDFontType0 について、標準装備として Acrobat Distiller の `MSung-Light`(繁体字), `STSong-Light`(簡体字), `HeiseiKakuGo-W5`, `HeiseiMin-W3`(日本語), `MHei-Medium`, `HYGoThic-Medium`, `HYSMyeongJo-Medium`(韓国語) が有名。CID グリフセットとして、`Adobe-CNS1-4` (繁体字), `Adobe-GB1-4`(簡体字), `Adobe-Korea1-2`(韓国語), そして後述のヒラギノの Apple Publishing Glyph Set とすりあわせを行なった `Adobe-Japan1-5`(日本語) が最新。

OpenType Font Adobe, Microsoft が策定した TrueType Font, CIDFont に代わるディスプレイ用にも印刷用にも使用に耐え得るホストベースプリンタ用フォント。ゆえに、フォントはオペレーティング環境側にインストールされることを想定している。標準装備として Adobe 製 `KozMinPro-Regular` (日本語), Acrobat 5 の `HYSMyeongJoStd-Medium`(韓国語), `MSungStd-Light`(繁体字), `STSongStd-Light`(簡体字)、Mac OS X の `HiraginoKaku`, `HiraginoMin`(大日本スクリーン) が有名。このヒラギノの CID グリフセットは `Adobe-Japan1-4` のスーパーセットである Apple

Publishing Glyph Set に基づいており、その後策定された `Adobe-Japan1-5` とは 28 グリフ以外では互換性がある。Ghostscript では扱うことは出来ない (gs-cjk 開発関係者によると、Ghostscript に手直しすれば扱えることが確認されているが、筆者は未確認)。現在、印刷業界にてもっとも注目されているフォント形式。Mac OS X, Windows XP では扱うことが可能。

## 4.2 CJK 文字コードからグリフセットへの変換表

CID フォントは、ラテンフォントとは異なり、Adobe CMap と組み合わせて使用される。漢字文化圏では、グリフがもともと多様であることによりコード空間を広げた結果、本来漢字とは関係のない記号類もグリフとしてシステムフォントに歴史的に組み入れられてきたこともあって、印刷業界等を問わず、大量のグリフが要求されてきた。無論、多種多様な漢字も増え続けている。Adobe が各プラットフォーム向けにそれらのグリフを簡単に印字できるよう、オペレーティング環境非依存の PostScript の枠組で開発・保守しているのがキャラクター写像テーブル辞書、CMap である。

実際のところ、Adobe が配布している CMap は大きく分けて三つある。ひとつは、文字コードや文字集合から Adobe CID への写像を定義する *ToCID CMap*。もうひとつは、Adobe CID から文字コードや文字集合への写像を定義する *FromCID CMap* である。両者は逆写像の関係にあると思われるだろうが、実はそうではない。ToCID CMap は、想定しているプラットフォームである文字列から期待されるグリフへの写像を定義しているが、FromCID CMap は、あるプラットフォームで CID へ写像されたグリフを、あるプラットフォームで文字コードへ逆変換する目的に使用される。つまり、グリフから情報交換用文字コードへの便宜的な変換テーブルであり、Acrobat (Reader) のようなアプリケーション中でグリフをオペレーティング環境に即した文字列と認識させる場合等に用いられる。残りのひとつは、CID に直接は関係の無い *nonCID CMap*。これは文字コードから文字コードへの変換テーブルである。これもやはり、Acrobat (Reader) のようなアプリケーション中で使用されるのであろう。

まずは、最新の各国 ToCID CMap について概観しよう。ちなみに、以下、CMap `*-H` に対応している CMap `*-V` は、縦書き用の CMap であり、横書き用 CMap を継承する形で、書字方向によってデザインに変化があるグリフ (例えば、句読点など多数) が再定義されている。

**◆台湾 … Adobe-CNS1 用 ToCID CMap**

- Adobe-CNS1-0: 14099 CID までの恒等写像
- Adobe-CNS1-1: 17408 CID までの恒等写像
- Adobe-CNS1-2: 17601 CID までの恒等写像
- Adobe-CNS1-3: 18846 CID までの恒等写像
- Adobe-CNS1-4: 18965 CID までの恒等写像
- B5-H, B5-V: chinese-big5 エンコーディング (半角固定ピッチラテン)
- B5pc-H, B5pc-V: chinese-big5 エンコーディング (可変ピッチラテン)
- ETen-B5-H, ETen-B5-V: big5+ETen 拡張エンコーディング
- CNS1-H, CNS1-V: CNS11643-1992-p1 文字集合 (ISO-2022 準拠コードポイント)
- CNS2-H, CNS2-V: CNS11643-1992-p2 文字集合 (ISO-2022 準拠コードポイント)
- CNS-EUC-H, CNS-EUC-V: CNS11643-1992-p1&2、台湾 EUC
- UniCNS-UCS2-H, UniCNS-UCS2-V: Unicode UCS-2
- UniCNS-UTF8-H, UniCNS-UTF8-V: Unicode 3.1 UTF-8 エンコーディング
- UniCNS-UTF16-H, UniCNS-UTF16-V: Unicode 3.1 UTF-16 エンコーディング

- UniCNS-UTF32-H, UniCNS-UTF32-V: Unicode 3.1 UTF-32 エンコーディング
- ETHK-B5-H, ETHK-B5-V: big5+ETen+香港 SCS 拡張エンコーディング
- HKdla-B5-H, HKdla-B5-V: big5+ダイナラボ香港拡張 A エンコーディング
- HKdlb-B5-H, HKdlb-B5-V: big5+ダイナラボ香港拡張 B エンコーディング
- HKgccs-B5-H, HKgccs-B5-V: big5+香港 GCCS 拡張エンコーディング
- HKm314-B5-H, HKm314-B5-V: big5+モノタイプ香港拡張 314 エンコーディング
- HKm471-B5-H, HKm471-B5-V: big5+モノタイプ香港拡張 471 エンコーディング
- HKscs-B5-H, HKscs-B5-V: big5+香港 SCS 拡張エンコーディング

**◆中国 … Adobe-GB1 用 ToCID CMap**

- Adobe-GB1-0: 7717 CID までの恒等写像
- Adobe-GB1-1: 9897 CID までの恒等写像
- Adobe-GB1-2: 22127 CID までの恒等写像
- Adobe-GB1-3: 22353 CID までの恒等写像
- Adobe-GB1-4: 29064 CID までの恒等写像
- GB-H, GB-V: GB2312-1980 文字集合 (ISO-2022 準拠コードポイント)
- GB-EUC-H, GB-EUC-V: euc-china エンコーディング (半角固定ピッチラテン)
- GBpc-EUC-H, GBpc-EUC-V: euc-china エンコーディング (可変ピッチラテン)
- GBT-H, GBT-V: GB/T12345-1990 文字集合 (ISO-2022 準拠コードポイント)
- GBT-EUC-H, GBT-EUC-V: GB/T12345-1990 文字集合、euc-china エンコーディング (半角固定ピッチラテン)
- GBTpc-EUC-H, GBTpc-EUC-V: GB/T12345-1990 文字集合、euc-china エンコーディング (可変ピッチラテン)
- GBK-EUC-H, GBK-EUC-V: MS PRC Chinese エンコーディング (半角固定ピッチラテン)
- GBKp-EUC-H, GBKp-EUC-V: MS PRC Chinese エンコーディング (可変ピッチラテン)
- GBK2K-H, GBK2K-V: GB18030-2000 文字集合、GBK ベースのマルチバイトエンコーディング
- UniGB-UCS2-H, UniGB-UCS2-V: Unicode UCS-2
- UniGB-UTF8-H, UniGB-UTF8-V: Unicode 3.2 UTF-8 エンコーディング
- UniGB-UTF16-H, UniGB-UTF16-V: Unicode 3.2 UTF-16 エンコーディング
- UniGB-UTF32-H, UniGB-UTF32-V: Unicode 3.2 UTF-32 エンコーディング

**◆日本 … Adobe-Japan1 用 ToCID CMap**

- Adobe-Japan1-0: 8284 CID までの恒等写像
- Adobe-Japan1-1: 8359 CID までの恒等写像
- Adobe-Japan1-2: 8720 CID までの恒等写像
- Adobe-Japan1-3: 9354 CID までの恒等写像
- Adobe-Japan1-4: 15444 CID までの恒等写像
- Adobe-Japan1-5: 20317 CID までの恒等写像
- H, V: JISX0208-1997 文字集合 (ISO-2022 準拠コードポイント)
- RKSJ-H, RKSJ-V: JISX0208-1997 文字集合、Shift-JIS エンコーディング
- EUC-H, EUC-V: JISX0208-1997 文字集合、euc-japan エンコーディング
- 83pv-RKSJ-H: KanjiTalk6 文字集合、Shift-JIS エンコーディング (可変ピッチラテン)
- Add-H, Add-V: 富士通文字集合 (ISO-2022 準拠コードポイント)
- Add-RKSJ-H, Add-RKSJ-V: 富士通文字集合、Shift-JIS エンコーディング
- Ext-H, Ext-V: NEC 文字集合 (ISO-2022 準拠コードポイント)

- Ext-RKSJ-H, Ext-RKSJ-V: NEC 文字集合、Shift-JIS エンコーディング
- NWP-H, NWP-V: NEC 文豪文字集合 (ISO-2022 準拠コードポイント)
- 90pv-RKSJ-H, 90pv-RKSJ-V: KanjiTalk7 文字集合、Shift-JIS エンコーディング (可変ピッチラテン)
- 90ms-RKSJ-H, 90ms-RKSJ-V: MS Kanji 文字集合、Shift-JIS エンコーディング (半角固定ピッチラテン)
- 90msp-RKSJ-H, 90msp-RKSJ-V: MS Kanji 文字集合、Shift-JIS エンコーディング (可変ピッチラテン)
- 78-H, 78-V: JISC6226-1978 文字集合 (ISO-2022 準拠コードポイント)
- 78-RKSJ-H, 78-RKSJ-V: JISC6226-1978 文字集合、Shift-JIS エンコーディング
- 78ms-RKSJ-H, 78ms-RKSJ-V: MS Kanji+JISC6226-1978 文字集合、Shift-JIS エンコーディング
- 78-EUC-H, 78-EUC-V: JISC6226-1978 文字集合、euc-japan エンコーディング
- UniJIS-UCS2-H, UniJIS-UCS2-V: Unicode UCS-2(可変ピッチラテン)
- UniJIS-UCS2-HW-H, UniJIS-UCS2-HW-V: Unicode UCS-2(半角固定ピッチラテン)
- UniJISPro-UCS2-V: UniJIS-UCS2-V と 9 コードポイント (元号、クオート等) 以外は同じ
- UniJISPro-UCS2-HW-V: UniJIS-UCS2-HW-V と 9 コードポイント (元号、クオート等) 以外は同じ
- UniJISPro-UTF8-V: UniJIS-UTF8-V と 9 コードポイント (元号、クオート等) 以外は同じ
- UniJIS-UTF8-H, UniJIS-UTF8-V: Unicode 3.2 UTF-8 エンコーディング
- UniJIS-UTF16-H, UniJIS-UTF16-V: Unicode 3.2 UTF-16 エンコーディング
- UniJIS-UTF32-H, UniJIS-UTF32-V: Unicode 3.2 UTF-32 エンコーディング
- UniJISX0213-UTF32-H, UniJISX0213-UTF32-V: UniJISX0213-UTF32-H は UniJIS-UTF32-H と 65 コードポイント [17] 以外は同じ、UniJISX0213-UTF32-V は UniJIS-UTF32-V と 9 コードポイント (元号、クオート等)[17] 以外は同じ
- Hiragana: ひらがな文字集合、シングルバイト
- Katakana: カタカナ文字集合、シングルバイト
- Hankaku: ラテン+カタカナ文字集合、シングルバイト
- Roman: ラテン文字集合、シングルバイト
- WP-Symbol: ワープロ用記号文字集合、シングルバイト

**◆日本 (補助漢字) … Adobe-Japan2 用 ToCID CMap**

- Adobe-Japan2-0: 6068 CID までの恒等写像
- Hojo-H, Hojo-V: JISX0212-1990 文字集合 (ISO-2022 準拠コードポイント)
- Hojo-EUC-H, Hojo-EUC-V: JISX0212-1990 文字集合についての euc-japan エンコーディング
- UniHojo-UCS2-H, UniHojo-UCS2-V: Unicode UCS-2
- UniHojo-UTF8-H, UniHojo-UTF8-V: Unicode 3.1 UTF-8 エンコーディング
- UniHojo-UTF16-H, UniHojo-UTF16-V: Unicode 3.1 UTF-16 エンコーディング
- UniHojo-UTF32-H, UniHojo-UTF32-V: Unicode 3.1 UTF-32 エンコーディング

**◆韓国 … Adobe-Korea1 用 ToCID CMap**

- Adobe-Korea1-0: 9333 CID までの恒等写像
- Adobe-Korea1-1: 18155 CID までの恒等写像
- Adobe-Korea1-2: 18352 CID までの恒等写像

- KSC-H, KSC-V:　KSX1001-1992 文字集合 (ISO-2022 準拠コードポイント)
- KSC-EUC-H, KSC-EUC-V:　KSX1001-1992 文字集合、euc-korea エンコーディング (半角固定ピッチラテン)
- KSCpc-EUC-H, KSCpc-EUC-V:　MacOS-KH 文字集合、euc-korea エンコーディング (可変ピッチラテン)
- KSCms-UHC-H, KSCms-UHC-V:　MS Unified Hangul Code エンコーディング (可変ピッチラテン)
- KSCms-UHC-HW-H, KSCms-UHC-HW-V:　MS Unified Hangul Code エンコーディング (半角固定ピッチラテン)
- KSC-Johab-H, KSC-Johab-V:　Johab エンコーディング
- UniKS-UCS2-H, UniKS-UCS2-V:　Unicode UCS-2
- UniKS-UTF8-H, UniKS-UTF8-V:　Unicode 3.1 UTF-8 エンコーディング
- UniKS-UTF16-H, UniKS-UTF16-V:　Unicode 3.1 UTF-16 エンコーディング
- UniKS-UTF32-H, UniKS-UTF32-V:　Unicode 3.1 UTF-32 エンコーディング

恒等写像用の CMap について若干補足しておこう。本稿 (例題 7) にて CID フォントと `glyphshow` オペレータの使用例を示したが、今一度、PostScript で示す。以下は Adobe-Japan1 で定義される 2 種類の半角円記号「\」「\」を印字する例である。

```
%!
/KozMin-Regular /CIDFont findresource 10 scalefont setfont
100 100 moveto 61 glyphshow 291 glyphshow
showpage
```

このように、CID フォントを `glyphshow` で印字する場合、`findfont` は使わずに `/CIDFont findresource` を用いて、直接 CID にアクセスする。しかし、`glyphshow` の引数は整数ひとつ (非 CID フォントの場合、キャラクタ名) なので、複数のグリフを連続して印字する場合には多少不便である。よって以下のように 16 進数表記の文字列とともに恒等写像用の CMap を使うことが出来る。

```
%!
/KozMin-Regular--Adobe-Japan1-0 findfont 100 scalefont setfont
100 100 moveto <003d0123> show
showpage
```

もしくは、以下のように 8 進数表記の文字列とともに恒等写像用の CMap を使ってもよいだろう。

```
%!
/KozMin-Regular--Adobe-Japan1-0 findfont 100 scalefont setfont
100 100 moveto (\000\075\001\043) show
showpage
```

実は恒等写像用 CMap は他にも `Identity-H` と `Identity-V` がある。これを用いる場合、

```
%!
/KozMin-Regular--Identity-H /Identity-H [/KozMin-Regular] composefont pop
/KozMin-Regular--Identity-H findfont 100 scalefont setfont
100 100 moveto <003d0123> show
showpage
```

などのように使う。Adobe Distiller の場合は一行目は不要だが、Ghostscript の場合は必要となってしまっている (これは gs-cjk では対応していたが、本家に却下されたもののひとつである)。

ここで注目したいのは、現在在る恒等写像 CMap が CID 最大値 65535 であろうことが推測される点である。一方で、規格書によると CID の最大値は実装依存であるとされている。Ghostscript の CID 実装限界も 65535 である (実は、各所でその半分であるコードもある…)。もし将来、CID 最大値が 16 ビットの 65535 を越えた場合 (これは、例えば、Unicode 3.2 をフル実装するグリフ集をフォントに収める場合等、容易に考えられる)、恒等写像 CMap を新たに追加するだけで容易に対応できるようにすることを Adobe は考えているのだろう。仮に、もし将来の Adobe-GB1-5 が 65536 個を越えるグリフ集であるなら、恒等写像 CMap の `Adobe-GB1-5` については 32 ビット対応され、`Identity32-H`, `Identity32-V` のような新たな 32 ビット対応恒等写像用 CMap も用意されるのではないかと筆者は推測している。

ちなみに、「ISO-2022 準拠コードポイント」と説明を付記した CMap については、Adobe や Ken Lunde 氏の文書 [3] では「ISO-2022 エンコーディング」と説明されている。これは日本人以外の読者には大変誤解を生みやすい単語である。なぜなら、これらの CMap で定義されているのは ISO-2022 エンコーディングに使われる $94^n$ コードポイントにすぎず、エンコーディング方式を実現したものではないからである。

ところで、ISO-2022-JP エンコーディングは日本では電子メール等で使用されており、慣れ親しんだエンコーディング方式であるが、ベースとなる ISO-2022 は大変優れたエンコーディング方式にもかかわらず、アジアの一部を除いてはあまり縁の無いものになってしまっている [1, 4]。ISO-2022 実装のエディタとして `Emacs/Mule`、ビューアとして `lv` があげられるが、プレーンテキスト印刷フィルタとして、拙作の `tops`[18] をあげておきたい。これは PostScript の機能のみを用いた各種エンコーディング用テキスト印刷フィルタであり、Adobe CMap の利点を最大限に使用することで、テキストで意図しているグリフがそのまま印字されるよう、ホストアプリケーションでエンコーディング処理をまったく行なわずに実装されている。ちなみに、ISO-2022 エンコーディングについては PostScript の Type0 のフォント切替え機構を用いている。出力例が Ghostscript のサンプル `examples/cjk/iso2022.ps` として提供されており、スクリーンショットが gs-cjk のサイトの「Ghostscript CJK supplement information」[23] で閲覧できる。

## 4.3 CJK グリフセットから文字コードへの変換表

次に、最新の各国 FromCID CMap について概観しよう。

**◆台湾 … Adobe-CNS1 用 FromCID CMap**

- Adobe-CNS1-B5pc: Adobe-CNS1 グリフセットから chinese-big5 エンコーディングへの写像
- Adobe-CNS1-ETen-B5: Adobe-CNS1 グリフセットから big5+ETen 拡張エンコーディングへの写像
- Adobe-CNS1-UCS2: Adobe-CNS1 グリフセットから Unicode UCS-2 への写像
- Adobe-CNS1-H-CID: 用途不明。おそらく、ラテン+繁体字フォントへ CID フォントを代替させるための恒等写像 CMap
- Adobe-CNS1-H-Host: 用途不明。おそらく、ラテン+繁体字 Windows フォントへ CID フォントを代替させるための CMap
- Adobe-CNS1-H-Mac: 用途不明。おそらく、ラテン+繁体字 Mac フォントへ CID フォントを代替させるための CMap

**◆中国 … Adobe-GB1 用 FromCID CMap**

- Adobe-GB1-GBK-EUC: Adobe-GB1 グリフセットから MS PRC Chinese エンコーディングへの写像
- Adobe-GB1-GBpc-EUC: Adobe-GB1 グリフセットから euc-china エンコーディングへの写像
- Adobe-GB1-UCS2: Adobe-GB1 グリフセットから Unicode UCS-2 への写像
- Adobe-GB1-H-CID: 用途不明。おそらく、ラテン+簡体字フォントへ CID フォントを代替させるための恒等写像 CMap
- Adobe-GB1-H-Host: 用途不明。おそらく、ラテン+簡体字 Windows フォントへ CID フォントを代替させるための CMap
- Adobe-GB1-H-Mac: 用途不明。おそらく、ラテン+簡体字 Mac フォントへ CID フォントを代替させるための CMap

**◆日本 … Adobe-Japan1 用 FromCID CMap**

- Adobe-Japan1-90ms-RKSJ: Adobe-Japan1 グリフセットから MS Kanji 文字集合、Shift-JIS エンコーディングへの写像
- Adobe-Japan1-90pv-RKSJ: Adobe-Japan1 グリフセットから KanjiTalk7 文字集合、Shift-JIS エンコーディングへの写像
- Adobe-Japan1-UCS2: Adobe-Japan1 グリフセットから Unicode UCS-2 への写像
- Adobe-Japan1-H-CID: 用途不明。おそらく、ラテン+日本語フォントへ CID フォントを代替させるための恒等写像 CMap
- Adobe-Japan1-H-Host: 用途不明。おそらく、ラテン+日本語 Windows フォントへ CID フォントを代替させるための CMap
- Adobe-Japan1-H-Mac: 用途不明。おそらく、ラテン+日本語 Mac フォントへ CID フォントを代替させるための CMap
- Adobe-Japan1-PS-H: 用途不明。おそらく、ラテン+日本語 PostScript フォントへ CID フォントを代替させるための CMap
- Adobe-Japan1-PS-V: 用途不明。おそらく、ラテン+日本語 PostScript フォントへ CID

フォントを代替させるための CMap

**◆韓国 … Adobe-Korea1 用 FromCID CMap**

- Adobe-Korea1-KSCms-UHC: Adobe-Korea1 グリフセットから MS Unified Hangul Code エンコーディングへの写像
- Adobe-Korea1-KSCpc-EUC: Adobe-Korea1 グリフセットから MacOS-KH 文字集合、euc-korea エンコーディングへの写像
- Adobe-Korea1-UCS2: Adobe-Korea1 グリフセットから Unicode UCS-2 への写像
- Adobe-Korea1-H-CID: 用途不明。おそらく、ラテン+韓国語フォントへ CID フォントを代替させるための恒等写像 CMap
- Adobe-Korea1-H-Host: 用途不明。おそらく、ラテン+韓国語 Windows フォントへ CID フォントを代替させるための CMap
- Adobe-Korea1-H-Mac: 用途不明。おそらく、ラテン+韓国語 Mac フォントへ CID フォントを代替させるための CMap

## 4.4 文字コードから文字コードへの変換表

次に、最新の各国 nonCID CMap について概観しよう。

**◆台湾 … nonCID CMap**

- UCS2-B5pc: Unicode UCS-2 から chinese-big5 エンコーディングへの写像
- UCS2-ETen-B5: Unicode UCS-2 から big5+ETen 拡張エンコーディングへの写像
- B5pc-UCS2C, B5pc-UCS2: chinese-big5 エンコーディングから Unicode UCS-2 への写像 (B5pc-UCS2 ではひとつのコードポイントから「コード列」への写像を定義)
- ETen-B5-UCS2: big5+ETen 拡張エンコーディングから Unicode UCS-2 への写像

**◆中国 … nonCID CMap**

- UCS2-GBK-EUC: Unicode UCS-2 から MS PRC Chinese エンコーディングへの写像
- UCS2-GBpc-EUC: Unicode UCS-2 から euc-china エンコーディングへの写像
- GBK-EUC-UCS2: MS PRC Chinese エンコーディングから Unicode UCS-2 への写像
- GBpc-EUC-UCS2C, GBpc-EUC-UCS2: euc-china エンコーディングから Unicode UCS-2 への写像 (GBpc-EUC-UCS2 ではひとつのコードポイントから「コード列」への写像を定義)

**◆日本 … nonCID CMap**

- UCS2-90ms-RKSJ: Unicode UCS-2 から MS Kanji 文字集合、Shift-JIS エンコーディングへの写像
- UCS2-90pv-RKSJ: Unicode UCS-2 から KanjiTalk7 文字集合、Shift-JIS エンコーディングへの写像
- 90ms-RKSJ-UCS2: MS Kanji 文字集合、Shift-JIS エンコーディング Unicode UCS-2 への写像
- 90pv-RKSJ-UCS2C, 90pv-RKSJ-UCS2: KanjiTalk7 文字集合、Shift-JIS エンコーディング Unicode UCS-2 への写像 (90pv-RKSJ-UCS2 ではひとつのコードポイントから「コード列」への写像を定義)

**◆韓国 … nonCID CMap**

- UCS2-KSCms-UHC: Unicode UCS-2 から MS Unified Hangul Code エンコーディング

への写像

- UCS2-KSCpc-EUC: Unicode UCS-2 から MacOS-KH 文字集合、euc-korea エンコーディングへの写像
- KSCms-UHC-UCS2: MS Unified Hangul Code エンコーディングから Unicode UCS-2 への写像
- KSCpc-EUC-UCS2C, KSCpc-EUC-UCS2: MacOS-KH 文字集合、euc-korea エンコーディングから Unicode UCS-2 への写像 (KSCpc-EUC-UCS2 ではひとつのコードポイントから「コード列」への写像を定義)

## 4.5 情報交換用文字コードと印刷用グリフセット

情報交換用文字コードと印刷用グリフセットの関係は、非常に複雑である。それは Adobe CMap を眺めてみるだけでも容易に想像できる。同時に、Adobe CMap により、印刷業界で第一線を行く Adobe の取り組み方が垣間見え、アプリケーションユーザからは見えない底辺の箇所で大変な労力が割かれていることが判る。

CMap は Adobe 及びÓReilly により配布されている誰でも手に入れることができるファイルであり、もちろん改変は許されていないが、自由に使用することが出来る。gs-cjk では既にその使用に頼っているが、Ghostscript に限らず PostScript とは直接関係ないアプリケーションでも利用が可能になれば、CJK 周辺のコンピューティングに価値ある恩恵が得られると考えている。

## 4.6 CJK TTF から CID へのグリフ変換アルゴリズム

さて、gs-cjk の成果のひとつとして一般に広く受け入れられているものが「TrueType フォントを CID フォントのように見せかけて使えるようにする技術」であるが、その現時点のアルゴリズムは参考文献 [25] に記されている。この文献に書いてあるように、CID フォントは TrueType フォントに比べてグリフが豊富にあるため、単一の TrueType フォントからグリフセットを満すこと自体には無理がある。但し、通常使われる文字に関しては特に目立つ問題はないため、この技術がユーザに自然に受け入れられているのだと思われる。つまり、Ghostscript のユーザ層には Adobe 製品のような緻密なコーディングは求められておらず、孕んでいる問題を重要視していないとも受け止められる。

では実際のその部分 `gs_ttf.ps` のコードを、特に使用されているリソースに注目して見てみよう。

```
/Adobe-CNS1 <<
  /Registry (Adobe) /Ordering (CNS1) /CIDCounts [14099 17408 17601 18846 18965]
  /Big5 { 0 {
    /Adobe-CNS1-ETen-B5         .applyCIDToCode
    /ETen-B5-V                  .applyvCMap
    /ETen-B5-H                  .applyhCMap
  } }
  /Unicode { 3 {
    /Adobe-CNS1-UCS2            .applyCIDToUnicode
    /UniCNS-UCS2-V              .applyvCMapUnicode
    /UniCNS-UCS2-H              .applyhCMap
  } }
>>
/Adobe-GB1 <<
  /Registry (Adobe) /Ordering (GB1) /CIDCounts [7717 9897 22127 22353 29064]
  /PRC { 2 {
    /Adobe-GB1-GBK-EUC          .applyCIDToCode
    /GBK-EUC-V                  .applyvCMap
    /GBK-EUC-H                  .applyhCMap
  } }
  /Unicode { 4 {
    /Adobe-GB1-UCS2             .applyCIDToUnicode
    /UniGB-UCS2-V               .applyvCMapUnicode
    /UniGB-UCS2-H               .applyhCMap
  } }
>>
/Adobe-Japan1 <<
  /Registry (Adobe) /Ordering (Japan1) /CIDCounts [8284 8359 8720 9354 15444]
  /ShiftJIS { 2 {
    /Adobe-Japan1-90ms-RKSJ     .applyCIDToCode
    /90ms-RKSJ-V                .applyvCMap
    /90ms-RKSJ-H                .applyhCMap
  } }
  /Unicode { 4 {
    /Adobe-Japan1-UCS2          .applyCIDToUnicode
    /UniJIS-UCS2-V              .applyvCMapUnicode
    /UniJIS-UCS2-H              .applyhCMap
  } }
>>
/Adobe-Japan2 <<
  /Registry (Adobe) /Ordering (Japan2) /CIDCounts [6068]
  /Unicode { 0  {
    /UniHojo-UCS2-V             .applyvCMapUnicode
    /UniHojo-UCS2-H             .applyhCMap
  } }
>>
/Adobe-Korea1 <<
  /Registry (Adobe) /Ordering (Korea1) /CIDCounts [9333 18155 18352]
  /Johab { 1 {
    /KSC-Johab-V                .applyvCMap
    /KSC-Johab-H                .applyhCMap
  } }
  /Unicode { 2 {
    /Adobe-Korea1-UCS2          .applyCIDToUnicode
    /UniKS-UCS2-V               .applyvCMapUnicode
    /UniKS-UCS2-H               .applyhCMap
  } }
  /Wansung { 1 {
    /Adobe-Korea1-KSCms-UHC     .applyCIDToCode
    /KSCms-UHC-V                .applyvCMap
    /KSCms-UHC-H                .applyhCMap
  } }
>>
/Identity <<
  /Registry (Unregistered) /Ordering (Identity) /CIDCounts [65535]
  /H { 0 { /Identity-H          .applyhCMap } }
  /V { 0 { /Identity-H          .applyvCMap } }
>>
```

ここで.apply*というのは独自の手続きであり Adobe CMap のデータを利用して、TrueType フォントの cmap テーブルを頼りに、グリフセットを Adobe CID に近くなるよう「割当て」を行うものである。Big5, Johab, PRC, ShiftJIS, Unicode, Wansung は TrueType の cmap テーブルの識別名、残りは、Adobe CMap に関連する識別名であり、「正しいグリフを割当ててくれる CMap を後から適用すること」によって、つまり割当てを上書きする形で並び替えを行っている。

かなり単純なことしかやっていないことがわかるだろう。詳しくは、参考文献 [25] に書いてあるのでそちらに譲るが、改善の余地はまだまだたくさんあるのである。例えば、`Uni*-UCS2-H,V` の代りに `Uni*-UTF16-H,V` を使うようにするのもちょっとした改善となる。

本格的な改善をするとなると、Ghostscript を広範囲に渡って直す必要が生じるかもしれない。例えば、`Adobe-*-H-Host` 等を使用して、CJK TrueType+ラテンフォントを混成して、ひとつの CID フォントの代替フォントとするのが理想的かもしれない。ただその方式に固定してしまうのもユーザの選択肢を奪ってしまうので考えものである。よって、Ghostscript のフォント定義のための機構である cidfmap(本家製)、CIDFnmap(gs-cjk 製)、Fontmap(非 CJK 用本家製) を全面的に書き換えた方が好ましいのかもしれないとも考えている。

### 4.6.1 【コラム】テキストフィルタ into PostScript — tops

本論で取り上げられている `tops` について紹介しよう。これは参考文献 [21] で紹介されているテキストフィルタを拡大解釈し、タブ、行送り、改頁処理、カラー制御、太字、下線等を ISO-6429 のエスケープシーケンスの枠組みで処理し、物理ページ上のマルチページ印刷に対応、そして、CJK 印刷、縦書き印刷に完全対応、非 CJK 印刷に関しては参考文献 [22] の 1.2 に則ったエンコーディングベクタ、CJK 印刷に関しては Adobe CMap に則ったエンコーディングに加えて、PostScript で実装した ISO-2022 エンコーディングのサブセットを実現、といった特徴をもつ。といいつつもその実体は、かつての gs-cjk project でのバグ洗い出しのために作成したアプリケーションの側面が色濃くあるので、筆者が考える限りに Ghostscript を「苛める」方向で実装されている。ゆえに、あるエンコーディング方式では PostScript 純正プリンタや Adobe Distiller では正しく動作するが、Ghostscript では未だに動作しないものもある。

基本的には Unix 系オペレーティング環境向けであるが、ほとんどの機能は PostScript で実現されているため、Windows や Mac への移植は容易であろう (但し、Unix 以外ではこのようなテキストフィルタの需要自体が無いと思う)。例えば、`filename.txt` が euc-jp2 エンコーディングであるなら、以下のように使う。

```
% tops -I euc-jp2 filename.txt | lp
```

多くは固定ピッチのフォントが規定値になっているが、可変ピッチのフォントも特に問題無く以下のように印刷することが出来る。例えば、`filename.txt` が日本語で utf8 エンコーディングであるなら、以下のようにプロポーショナルフォントを指定してもよいだろう。

```
% tops -I utf8-jp -F Latin=Times-Roman -F Latin_Bold=Times-Bold -F Japan1=MS-PM
incho -F Japan1_Bold=MS-PGothic filename.txt > filename.ps
% lpr filename.ps
```

種明しをすると、上例で生成される `filename.ps` は、インストールされている `/usr/local/share/tops/ishow-utf8-jp.ps` のファイル末尾に `fiename.txt` を結合しているだけである。サポートしているエンコーディング方式に応じて、`ishow-*.ps` がそれぞれ用意されている。中には「euc-jp と eucjp は同じじゃないか？」と思われるのもあるが、

フォントの構成方法が異っていたりする。では、主要なエンコーディング方式でのフォントの構成方法をまとめておこう。

○ b5-eten: 繁体字 CID フォント、ETen-B5 CMap による単一フォント
○ gbk2k: 簡体字 CID フォント、GBK2K CMap による単一フォント
○ hkscs: 繁体字 CID フォント、HKscs-B5 CMap による単一フォント
○ sjis: 日本語 CID フォント、RKSJ CMap による単一フォント
○ uhc: 韓国語 CID フォント、KSCms-UHC CMap による単一フォント
○ big5: 繁体字 CID+ラテンフォントの CMap による混成フォント
○ shift_jis: 日本語 CID+ラテンフォントの CMap による混成フォント
○ euc-china: 簡体字 CID/OCF+ラテンフォントの Type0/rearranged 混成フォント
○ euc-japan: 日本語 CID/OCF+ラテンフォントの Type0/rearranged 混成フォント
○ euc-korea: 韓国語 CID/OCF+ラテンフォントの Type0/rearranged 混成フォント
○ euc-cn: 簡体字 CID+ラテンフォントの CMap による混成フォント
○ euc-jp: 日本語 CID+ラテンフォントの CMap による混成フォント
○ euc-jp2: 補助漢字を含む日本語 CID+ラテンフォントの CMap による混成フォント
○ euc-kr: 韓国語 CID+ラテンフォントの CMap による混成フォント
○ euc-tw: 繁体字 CID+ラテンフォントの CMap による混成フォント
○ euccn: 簡体字 CID/OCF+ラテンフォントの Type3 による混成フォント
○ eucjp: 日本語 CID/OCF+ラテンフォントの Type3 による混成フォント
○ euckr: 韓国語 CID/OCF+ラテンフォントの Type3 による混成フォント
○ iso-2022-cjk: 繁体字+簡体字+補助漢字を含む日本語+韓国語 CID/OCF+ラテンフォントの Type0 混成フォント
○ iso-2022-cn: 簡体字 CID/OCF+ラテンフォントの Type0 混成フォント
○ iso-2022-jp: 日本語 CID/OCF+ラテンフォントの Type0 混成フォント
○ iso-2022-jp2: 補助漢字を含む日本語 CID/OCF+ラテンフォントの Type0 混成フォント
○ iso-2022-kr: 韓国語 CID/OCF+ラテンフォントの Type0 混成フォント
○ iso-2022-m17n: 繁体字+簡体字+補助漢字を含む日本語+韓国語 CID/OCF+ラテンフォント、ISOLatin1+2+5+Cyrillic エンコーディングベクタの Type0 混成フォント
○ iso-6429: ラテンフォント、Adobe 標準エンコーディングベクタでの単一フォント
○ iso-8859-1: ラテンフォント、ISO-8859-1 エンコーディングベクタでの単一フォント
○ iso-8859-2: ラテンフォント、ISO-8859-2 エンコーディングベクタでの単一フォント
○ iso-8859-3: ラテン 3 フォント、ISO-8859-3 エンコーディングベクタでの単一フォント
○ iso-8859-4: ラテン 4 フォント、ISO-8859-4 エンコーディングベクタでの単一フォント
○ iso-8859-5: Cyrillic フォント、ISO-8859-5 エンコーディングベクタでの単一フォント
○ iso-8859-7: Greek フォント、ISO-8859-7 エンコーディングベクタでの単一フォント
○ iso-8859-9: ラテンフォント、ISO-8859-9 エンコーディングベクタでの単一フォント
○ iso-8859-10: ラテン 6 フォント、ISO-8859-10 エンコーディングベクタでの単一フォント
○ iso-8859-13: ラテンフォント、ISO-8859-13 エンコーディングベクタでの単一フォント
○ iso-8859-14: ラテン 8 フォント、ISO-8859-14 エンコーディングベクタでの単一フォント
○ iso-8859-15: ラテンフォント、ISO-8859-15 エンコーディングベクタでの単一フォント
○ koi8-r: Cyrillic フォント、KOI8-R エンコーディングベクタでの単一フォント
○ utf8-cn: 簡体字 CID フォント+ラテンフォントの CMap による混成フォント
○ utf8-jp: 日本語 CID フォント+ラテンフォントの CMap による混成フォント
○ utf8-kr: 韓国語 CID フォント+ラテンフォントの CMap による混成フォント
○ utf8-tw: 繁体字 CID フォント+ラテンフォントの CMap による混成フォント
○ utf16-cn: 簡体字 CID フォント+ラテンフォントの CMap による混成フォント
○ utf16-jp: 日本語 CID フォント+ラテンフォントの CMap による混成フォント
○ utf16-kr: 韓国語 CID フォント+ラテンフォントの CMap による混成フォント
○ utf16-tw: 繁体字 CID フォント+ラテンフォントの CMap による混成フォント

特に、`iso-2022-*`では、JISX0208-1983 と JISX0208-1978(いわゆる 78JIS) の両方が印字可能となっているのは注目に値すると思う。

この他にも `ishow-*.ps` があるにはあるのだが、非 CID フォントに関して、書字方向が

Right-to-Left であるべきなのにそうなっていないもの・実質的に使えるフォントが存在しない類・複雑なリガチャ (合字) についてまったく検討していない類等々あり、CJK 以外ではかなり独り善がりな状態で開発が滞っている。また、Type3 による混成は仕方無いとしても、Type0 混成フォントが Ghostscript で PDF へ変換するとビットマップになってフォントとしての情報が失われてしまう (これは Ghostscript における課題であろう)。とはいえ、紙面への CJK 印刷には十分実用的なので、是非 Unix ユーザは御試しあれ。

### 4.6.2 【コラム】Adobe CMap を可視化する PostScript プログラム

筆者はもともと参考文献 [1] で取り上げられている「文字コード」に興味があり、同時に PostScript にある程度慣れていたこともあって、「文字コード」が情報交換の枠を出て、実際の形である「グリフ」へ達することが具体的に示されている Adobe CMap に大変興味をもった。それでたまたま gs-cjk project のなかでも文字コードと CMap に纏わる作業を担当したわけだが、正直なところ、自国のグリフに関しては当初はさほど興味はなかった。興味をもっている台湾、中国、韓国のグリフについては参考文献 [3] で得た知識などはみるみるうちに古くなっていき、日本のグリフについても JISX0213 や Apple Publishing Glyph Set に関連した新しい動きも目の当たりにした。結局、グリフに関する知識を得るために常に最新の Adobe CMap を可視化する必要性が生じた。そのために作成したのが、「CMap を可視化する PostScript プログラム」「2 つの CMap の差異のみを可視化する PostScript プログラム」[17] である。

Adobe CID フォントで定義されるグリフ集は参考文献 [11, 12, 13, 14, 15, 16] に (韓国語以外はすべて) 載っている。しかし、そのグリフがどういったエンコーディングの場合に要求されているか、なぜ似たようなグリフがたくさんあるのか、その文化的な背景はなんなのか、これらの文献を観るだけでなく、CMap まで辿ると理解できる場合が多い。加えて、これらをすべて合せると 92766 ものグリフがあり、とてもボランティアで面倒見切れる量ではない。しかし、CMap がバージョンアップした場合や、素性の似た CMap 同士の差異のみが可視化出来れば、能率良く理解を深めることが可能となる。

興味深い CMap については、サイト [17] にて既に PDF 形式で公開しているので、眺めてみるだけでも面白いだろう。

### 4.6.3 【コラム】hanzi, kanji, hanja, Unicode

欧米人からみて *Chinese Characters* といった場合、何を指すのだろうか。おそらく中国や日本、韓国で用いられている「漢字」のことを指すだろう。しかし、中国を発祥の地とする漢字は日本や韓国でも独自の発展をしており、同じ字形でも意味がまったく異なっていたり、意味はどうあれ一見同じに見えるものでも微妙に字形が異なっていたりする。前者について言えば、例えば、「机」という字形は日本では desk の意味に使われるが、中国本土ではこれは machine、「機」の簡体字として用いられる。台湾では繁体字が使われるので machine は「機」のまま、desk を表す漢字一字はない。後者については、情報交換用文字コードのひとつであり現在 Windows 等で用いられている Unicode では「CJK 統合漢字」と呼ばれ、同一コードに似たような字形が押し込められてしまっている。この問題に関しては専門家が兼ねてより議論を重ねている事柄であるようなので、筆者はこれに関して特に意義を唱えるつもりは毛頭ないのだが、Unicode の UCS2 にて同一のコードポイントに割り当てられている漢字 (中国では hanzi、日本では kanji、韓国では hanja) の一部を PostScript の機能を使って印字してみることによって、各国の字形がどのようになっているかを紹介しておこう。

| | | | | |
|---|---|---|---|---|
| U+4e0e | 与 | 与 | 与 | |
| U+9aa8 | 骨 | 骨 | 骨 | 骨 |
| U+5167 | 內 | 內 | 內 | 內 |
| U+5185 | | 内 | 内 | |
| U+6236 | 戶 | 戶 | 戶 | 戶 |
| U+6237 | 户 | 户 | | |
| U+6238 | | 戶 | 戶 | |
| U+5203 | 刃 | 刃 | 刃 | 刃 |
| U+7070 | 灰 | 灰 | 灰 | 灰 |
| U+76f4 | 直 | 直 | 直 | 直 |
| U+5c71 | 山 | 山 | 山 | 山 |
| U+98df | 食 | 食 | 食 | 食 |

表 1: UCS2, traditional hanzi(台湾), simplified hanzi(中国本土), kanji(日本), hanja(韓国)

まず、「与」の字形であるが、3 画めが突き出るものと出ないものが UCS-2 では統合されてしまっている。「これは仕方がないだろう」という見解が参考文献 [1] で示されているが、参考文献 [2] によると、日本の「当用漢字字体表」「常用漢字表」で例示されている字形であり、伝統的な楷書の形ではむしろ台湾の方が近い。

次に、「骨」の字形であるが、中国本土のみ画数が異なるのがわかるだろうか。「与」と同様にこのように字形が異なるものが一つのコードポイントに割り当てられているものは多い。

次に、「内」の字形であるが、部分が「入」か「人」かで別々のコードポイントに 2 種類割り当てられている。部分が「入」の方は日本の漢字辞典にも多大な影響を与えている中国の「康熙字典体」での字形であり、「人」の方は伝統的な楷書の形に近く、日本の「当用漢字字体表」でも例示されている字形である。

次に、「戸」の字形であるが、デザインの違いでなんと別々のコードポイントに 3 種類割り当てられている。U+6236 が「康熙字典体」での字形、U+6237 が「康熙字典体の序文」の楷書体で書かれている字形、U+6238 が康熙字典体に大いに影響を受けた「顔真卿」の字形に見受けられる。現在の明朝体は顔真卿の書きぶりである「顔法」の影響を受けている部分が多いということである。

次に、「刃」の字形であるが、これも「与」と同じく日本だけ少数派の字形となってしまっている。これに関するはっきりとした由来は筆者はわからないが、日本人には「与」と同じように突き出る書きぐせがあるのではないか、という見解が参考文献 [2] で示唆されている。

次に、「灰」の字形であるが、これも「与」と同じく日本だけ、1 画めと 2 画めが交差しない字形となっており、「当用漢字字体表」で例示されている字体である。

次に、「直」の字形であるが、興味深いことに中国の「康熙字典体」の影響を強く受けている字形は日本と韓国で見受けられ、台湾と中国の字形は伝統的な楷書の形を受け継いでいるように見える。

次に、「山」の字形であるが、これは細かなデザインに注目すると、中国と韓国では (書く為の字形ではない) 宋朝体・明朝体とは言え、正しい画数がわかるようになっていることがわかるだろう。つまり、日本の明朝体の「山」は 2 画めが突き出すぎているデザインなのである。

最後に、「食」の字形であるが、これも細かなデザインに注目すると、中国と台湾では明朝体とは言え、楷書体の「大きく払うのは一箇所のみで、もうひとつの小さい払いは止めて書く」という慣わしに則ってデザインされていることがわかる。

こうして、Unicode で同一のコードポイントに割り当てられている漢字を、各国の印刷の分野でデフォルトで使用される字体 (中国・台湾では宋朝体、日本・韓国では明朝体) でいちどきに印字してみて、違和感を覚える漢字の歴史を (参考文献 [2] を頼りに) 紐解いてみると、なるほど、時代時代の権力者によって定められる字形に影響を与えられている字形もあり、根強く人々の手によって書かれてきた字形を継承しているものもあり、特に台湾の字形は楷書体として美しい字形を教養・文化として大切にしていることが見受けられたり、日本のいわゆる「異体字」と呼ばれる字形が、漢字発達の一過程に過ぎない「篆書」をものさしとする流派である中国の「康熙字典体」にて「俗字」されるものに過ぎず、それを十分理解してフォントをデザインしているように見受けられる台湾・中国と、それと対照的に工業文化 (古くからの新聞の活版印刷業界) に多大な影響力を与えられている日本の漢字、日本と中国から商業的・文化的に大きな影響を与えられている韓国の hanja や台湾の漢字、などなど、いろいろと興味深い描像が得られる。

## 4.7 フリー・公有日本語フォントと gs-cjk

CJK-enable な Ghostscript で使用できるフリーもしくは公有な日本語フォントについてここで紹介する。日本語のような大量のグリフセットが要求されるアウトラインフォントが有志によって非営利目的で作成され、自由に使える時代を迎えたことに感動を覚えざるを得ない (6/19 追記、ここに紹介されているフリーフォントは「知的所有権を侵害していることが判明した渡辺ビットマップフォント」を参考にしており、これらは現在、公開を取り止めています)。

**◆東風明朝・ゴシックシリーズ** 古川氏による TrueType フォントとそれを起源とする狩野氏による CID 版、OpenType 版。詳細は`http://www.on.cs.keio.ac.jp/~yasu/jp_fonts.html`, `http://kappa.allnet.ne.jp/Kochi-CID/`。Ghostscript で使うには、gs の lib/CIDFnmap.Koc と参考文献 [30, 31] を参照のこと。特に CID 版は狩野氏本人が gs での用途に注意を払って下さっているので安定した利用が可能になっている。

**◆拡張 Watanabe 明朝フォント** 内田明氏が「JISX0213:2000 の実装水準 3 に適合する状態での利用を目的」として作成されている TrueType フォント。詳細は`http://www.asahi-net.or.jp/~sd5a-ucd/freefonts/`。Ghostscript で使うには、gs の lib/CIDFnmap に以下を加えるとよい。

```
/extended-watanabe-mincho       (extendedWatanabeMincho.ttf)    ;
/extended-watanabe-mincho-L     (extendedWatanabeMincho-L.ttf)  ;
```

但し、当然のことながら、いくつかのグリフが作者の意図するものと Adobe-Japan1 のものと一致しない。これについては、筆者が上述の `gs_ttf.ps` のコードを拡張して Adobe から将来配布される (かもしれない) 新しい CMap により対応する所存である。

**◆拡張ワタナベワダケン明朝フォント** 内田明氏が「JISX0213:2000 の実装水準 4 に適合する状態での利用を目的」として作成されている TrueType フォント。詳細は`http://www.asahi-net.or.jp/~sd5a-ucd/freefonts/`。Ghostscript で使うには、gs の lib/CIDFnmap に以下を加えるとよい。

```
/xWatanabe-WadaLab-mincho       (xWatanabeWadaLabSP.ttf)        ;
/xWatanabe-WadaLab-mincho-L     (xWatanabeWadaLabSP-L.ttf)      ;
```

但し、当然のことながら、いくつかのグリフが作者の意図するものと Adobe-Japan1 のものと一致しない。これについては、筆者が上述の `gs_ttf.ps` のコードを拡張して Adobe から将来配布される (かもしれない) 新しい CMap により対応する所存である。

**◆クワクチヤウワタナベ和田研明朝フォント** 内田明氏が「JISX0213:2000 の実装水準 4 に適合する (であらう) 状態での利用と旧字化の試みを目的」として作成されている TrueType フォント。詳細は`http://www.asahi-net.or.jp/~sd5a-ucd/freefonts/`。Ghostscript で使うには、gs の lib/CIDFnmap に以下を加えるとよい。

```
/qWatanabeWadaLab-mincho        (qWatanabeWadaLabSP.ttf)        ;
/qWatanabeWadaLab-mincho-L      (qWatanabeWadaLabSP-L.ttf)      ;
```

但し、いくつかのグリフが作者の意図するものと Adobe-Japan1 のものと一致しない。これについては、筆者が上述の `gs_ttf.ps` のコードを拡張して Adobe から将来配布される (であらう) 新しい CMap により対応する所存である。

**◆発羊明朝フォント** 内田明氏が「Adobe-Japan1-4 の追加漢字グリフの提供を目的」として作成されている TrueType 及び OpenType フォント。詳細は `http://www.asahi-net.or.jp/~sd5a-ucd/freefonts/`。Ghostscript で使うには、gs の lib/CIDFnmap に以下を加えるとよい。

```
/QuiMi-mincho                (QuiMi-mincho.ttf)      ;
%/QuiMi-mincho               (QuiMi-mincho.otf)      ;
```

## 4.8 ベンダの CJK フォントと gs-cjk

ここでは、CJK-enable な Ghostscript で技術的には使用できる (であろう) 商業ベースの CJK フォントについて筆者の知っている範囲で紹介する。但し、ユーザがそのフォントの使用許諾が得られているかを十分注意を払って使用すべきである。

**◆Adobe Acrobat バンドルフォント** Acrobat 4 に装備されているのは OpenType フォントで、Adobe-CNS1 では宋体 `MSung-Light-Acro`, 黒体 `MHei-Medium-Acro`、Adobe-GB1 では、宋体 `STSong-Light-Acro`、Adobe-Japan1 では、明朝体 `HeiseiMin-W3-Acro`, ゴシック体 `HeiseiKakuGo-W5-Acro`、Adobe-Korea1 では、明朝体 `HYSMyeongJo-Medium-Acro`, ゴシック体 `HYGoThic-Medium-Acro` である。

Acrobat 5 に装備されているのは OpenType フォントで、Adobe-CNS1 では宋体 `MSungStd-Light-Acro`、Adobe-GB1 では宋体 `STSongStd-Light-Acro`、Adobe-Japan1 では明朝体 (小塚明朝) `KozMinPro-Regular-Acro`、Adobe-Korea1 では明朝体 `HYSMyeongJoStd-Medium-Acro` である。Acrobat 6 については筆者はまだ購入していないので未確認。

無償で入手できる Reader も同様であるようだが、当然のことながら Reader 以外のアプリケーションでの使用は許可されていないので、Ghostscript で使用してはならない。日本語フォントの小塚シリーズは Adobe 製であることでも有名。ウェブの情報によると Reader 6 でのフォントのバンドル状況はこれまでとは異なるようである。

**◆Arphic(文鼎科技公司) のフリーフォント** ARPHIC による繁体字及び簡体字のフリー TrueType フォントは、Adobe-CNS1 として宋体 `ShanHeiSun-Light(bsmi00lp.ttf)`, 楷書体 `ZenKai-Medium(bkai00mp.ttf)`、Adobe-GB1 として宋体 `BousungEG-Light-GB(gbsn00lp.ttf)`, 楷書体 `GBZenKai-Medium(gkai00mp.ttf)` が、`ftp://ftp.gnu.org/pub/`から入手でき、Ghostscript で使うには、gs の lib/CIDFnmap.Arp もしくは参考文献 [31] を参照のこと。漢字のデザインが秀逸なので、以下のようにして、日本語としても使用したいところだが、いかんせん、ひらがなのデザインに異和感がある上、漢字も日本で見慣れている漢字とは異るので、このような使い方は単なる興味本位に過ぎない。

```
/ZenKai-Medium-Adobe-Japan1      (bkai00mp.ttf) /Adobe-Japan1 ; % not for use
/ShanHeiSun-Light-Adobe-Japan1   (bsmi00lp.ttf) /Adobe-Japan1 ; % not for use
/BousungEG-Light-GB-Adobe-Japan1 (gbsn00lp.ttf) /Adobe-Japan1 ; % not for use
/GBZenKai-Medium-Adobe-Japan1    (gkai00mp.ttf) /Adobe-Japan1 ; % not for use
```

余談だが、似たような方策を本家開発者が、実用に堪え得ると本気で考えて実装しているのを伝え聞いて、大変呆れた記憶がある。CJK フォントの入手がラテンフォントと比べて困難である故の苦肉の策なのかもしれないが、Unicode で同一のコードポイントに割当られた CJK 統合漢字が、印刷の分野で現時点で同一に使用できると考えているとしたら大いなる過ちである。

◆Baekmuk **フリーフォント**　Jeong-Hwan Kim, Font21 Inc. による韓国語 TrueType フォントは、Adobe-Korea1 として明朝体 `Baekmuk-Batang(batang.ttf)`, 丸ゴシック体 `Baekmuk-Gulim (gulim.ttf)`, ゴシック体 `Baekmuk-Dotum (dotum.ttf)`, 太ゴシック体 `Baekmuk-Headline (hline.ttf)` が、`ftp://ftp.mizi.com/pub/baekmuk/`から入手でき、Ghostscript で使うには、gs の lib/CIDFnmap.Bae もしくは参考文献 [31] を参照のこと。明朝体、丸ゴシック体についてはハングル (hangul) と漢字 (hanja) ともに充実している。TrueType フォントであることもあって、記号類の不足については致し方ないだろう。

◆ÓReilly **サンプルフォント**　Adobe が開発者向けに`ftp://ftp.oreilly.com/pub/`で配布しているサンプルのための CID フォント。これらが実用的であるかどうかは筆者にはわからないが、補助漢字以外はすべてかなりのサブセットであるし、日本語に関しては品質面で実用的ではないと思う。もし Ghostscript で使うには、gs の lib/CIDFnmap.Ore もしくは参考文献 [31] を参照のこと。

◆MS Windows XP, MS Office XP **バンドルフォント**　MS Windows XP に装備されている CJK フォントはさすがに充実している。MS Office XP をインストールするとさらに充実する。このあたりは他のオペレーティング環境は敵わないだろう。Ghostscript で使うには、gs の lib/CIDFnmap.Win もしくは参考文献 [31](Unix 向けの雑誌であるが Windows に限った設定例を示している) を参照のこと。但し、MingLiU(mingliu.ttc) については特許に纏わるヒンティング技術の関係で gs では事実上扱えない。また、MS Windows のフォントは他のオペレーティング環境での使用が許されていない (らしい) ので、Windows にインストールした Ghostscript でしかこれらを使用することが出来ないので注意すること (ちなみに筆者は Unix 使いなので、Cygwin を Windows にインストールすることによってこの問題を回避している)。つまり、単一のマシンに複数のオペレーティング環境をインストールした状態でも Windows のフォントディレクトリにアクセスしてはならないということだろうが、そんなことは使用許諾書を読んでもどこにも書いていないような気もする。バンドル日本語フォントはリコーによる OEM 供給。

◆Sun Solaris 8 **バンドルフォント**　おそらく、無償で入手が可能な、バンドル CJK フォントが充実している唯一のオペレーティング環境。Solaris 9 については未調査。Ghostscript で使うには、gs の lib/CIDFnmap.Sol を参照のこと。

◆Apple Mac OS X **バンドルフォント**　最新の日本語 OpenType フォント (ヒラギノ) がバンドルされていることで有名なオペレーティング環境。筆者は Mac OS X を使える環境にないので試したわけではないが、Ghostscript ではヒラギノは使えないらしい。直したら使えるようになった、という話しも伝え聞いているのだが、詳細は不明。他の CJK フォントについても筆者はまったくわからない。

◆Turbolinux **バンドルフォント**　MS 明朝準拠の `tlmincho.ttc`,MS ゴシック準拠の `tlgothic.ttc` がバンドルされている GNU/Linux。他にも丸ゴシック体 `tlmarugo.ttc`, 行書体 `tlgyosho.ttc`, 教科書体 `tlkyoksh.ttc` も提供されているらしい。筆者は所有していないので、TrueType フォント内で定義されている PostScript フォント名がわからず、CIDFnmap.TL を例示及び動作確認することが出来ずにいる。これらの日本語フォントはリコーによる OEM 供給。

他の、特に Unix 系オペレーティング環境についての CJK フォントバンドル状況については筆者はわからないが、公有の Debian GNU/Linux でも gs-cjk がうまく使われているらしい。

次に、TrueType, CID, OpenType フォントに絞って、フォントベンダの CJK フォント供給について、筆者の知る範囲でまとめておこう。ちなみに、DTP 業界では Type1 フォントを PostScript フォントと称したり、日本語 OCF フォントを和文 Type1 フォントと称したり、CID フォントを ATM フォントと称したりするので、購入する場合には注意が必要。そして、Mac OS 9 用として販売しているフォントは Ghostscript からアクセスするのは難しいとのこと。それに Ghostscript での使用はサポート対象外のはずであるので、フォントベンダに問い合わせても窓口を困らせるだけである。

**◇モリサワ** `www.morisawa.co.jp`
言わずと知れた最大手フォントベンダ。PostScript のデフォルトの日本語フォント名としても使用されるので、gs の lib/CIDFnmap.CJK では「モリサワフォントを何に代替させるか」が主題になってしまっているが、本物を持っているに越したことはない。

**◇大日本スクリーン** `www.screen.co.jp`
日本語のヒラギノ OpenType, CID で有名な「千都フォント」シリーズを販売。

◇Adobe Type Library `www.adobe.co.jp`
日本語の平成書体 (HeiseiMin,HeiseiKakuGo) シリーズの CID 版,OpenType 版、小塚明朝・ゴシックシリーズの CID 版,OpenType 版などを販売。余談だが、モリサワ、大日本スクリーン、Adobe と Apple が提携して、次世代 DTP システムを共同開発するとのこと。OpenType, Adobe-Japan1-5, PDF1.5, Mac OS X, Adobe 製アプリケーションで囲い込みの DTP 最強グループ。Unix 系オペレーティング環境で TeX をちまちまやっている筆者にはまさに別世界だが、実は大いに興味がある。

**◇タイプバンク** `www.typebank.co.jp`
明朝体、ゴシック体、カリグラゴシック体、テレビ明朝、新聞書体など高品位な日本語 CID, OpenType フォントを販売。

**◇ダイナコムウェア** `www.dynacw.co.jp`
中国語、日本語の漢字フォントの大手ベンダ。CID, OpenType 版の日本語 100 書体パッケージが有名。

**◇ニィス** `www.nisfont.co.jp`
日本語の NIS フォント、S 明朝、書体作家シリーズ等を CID, TrueType 版で販売。

**◇フォントワークス** `www.fontworks.com`
日本語のロダン，マティス，セザンヌ，スーラ，グレコなど CID, OpenType 版で販売。

**◇リョービ** `www.ryobi-group.co.jp`
日本語 TrueType, CID フォントパックの販売。

**◇イワタ** `www.iwatafont.co.jp`
日本語の新聞書体や学参書体が充実。待望の OpenType 版も発売された。

**◇モトヤフォント** `www.motoyafont.jp`
さまざまな日本語書体が充実。TrueType, CID 版で販売。

**◇日本リテラル**
日本語のセイビシリーズの TrueType 版の開発。サイト不明。

**◇トライオクス** `www.triox.co.jp`
日本語の武蔵野書体シリーズ TrueType フォントの販売。

**◇視覚デザイン研究所** `www.vdl.co.jp`
斬新なデザインの日本語 CID, OpenType フォントを販売。

**◇シーアンドジイ** `www.c-and-g.co.jp`
個性的な日本語 TrueType フォントを販売。

**◇日本活字工業** `www.sun-inet.or.jp/~ntftokyo`

活版活字の日活書体シリーズを日本語 TrueType フォントで販売。

◇**創英企画** www02.so-net.ne.jp/~souei100

キヤノンやリコーへ OEM 供給されている日本語の創英「字林」シリーズ書体を開発・販売。

◇Arphic, **文鼎科技公司** www.arphic.com.tw

中国語 TrueType フォントを軸に、日本語 TrueType フォントも販売する大手フォントベンダ。

◇Founder, **北京北大方正** www.founder.com.cn

Sun, Turbolinux, MS, Fuji Xerox, オムロン, IBM, Bitstream, Intel に CJK TrueType フォントを供給する大手フォントベンダ。

◇Zhong Yi Electronics www.zhongyicts.com.cn

Sun, MS, SGI 等に中国語フォントを供給する大手フォントベンダ。

◇Agfa Monotype www.agfamonotype.com

Adobe Acrobat 等にも Adobe-CNS1 OpenType フォントを供給している大手フォントベンダ。もともとは非 CJK フォントで有名だが、昨今は、GBK2K と Unicode をベースに CJK をカバーしているとのこと。Ghostscript の顧客らしいので、日本語フォントを中国語フォントで代替させようとしたのはここの意向であると推測している。パックスアメリカーナと中華思想があわさると大変脅威であるが、こういった所謂「国際化」の動きを受け止める心も必要だろう。

◇ChangZhou SinoType Technology

Adobe Acrobat 等にも繁体字 Adobe-GB1 OpenType フォントを供給している大手フォントベンダ。サイト不明。

◇**北京漢儀科印信息技術**

HANYI シリーズの中国語 TrueType フォントを開発。サイト不明。

◇Hunan Huatian Information Industry www.htit.com

HuaTian シリーズの中国語 TrueType フォントを開発・販売。

◇**ユーンデザイン研究所** www.yoonfont.co.kr

韓国語の Yoon シリーズを CID フォントで開発･販売､Adobe Acrobat 等にも Adobe-Korea1 CID を供給。

◇**ハーニャン** www.hanyang.co.kr

韓国語の Hanyang シリーズを CID フォントで開発・販売。

◇AVector, **向量** www.tops.com.hk/av-font

香港フォントの全真字庫シリーズを TrueType フォントで開発・販売。

◇**リコー** www.ricoh.co.jp

TrueType フォントを OEM 供給。CJK、タイフォントも開発とのこと。

これらのサイトをみてまわると、いくつかのベンダはサンプルフォントを無料ダウンロードできるようにしてくれていることがわかる。ここでは「モトヤフォント」の TrueType フォントの Ghostscript の使用例を示しておこう。

モトヤフォントのサイトで指示された手続きを経ると `MTXc1kp.ttc` というファイルが得られる。次に、gs の lib/CIDFnmap に以下のようなエントリを加えると、

```
/FMotoyaCedar-W1p       (MTXc1kp.ttc)          ;
/FMotoyaCedar-W1        (MTXc1kp.ttc)   2      ;
/FMotoyaCedar-W1kp      (MTXc1kp.ttc)   3      ;
```

TrueType コレクションフォントのそれぞれが `FMotoyaCedar-W1p`, `FMotoyaCedar-W1`, `FMotoyaCedar-W1kp` という CID フォント名で使用できるようになる。`FMotoyaCedar-W1` での印字例を示しておこう。

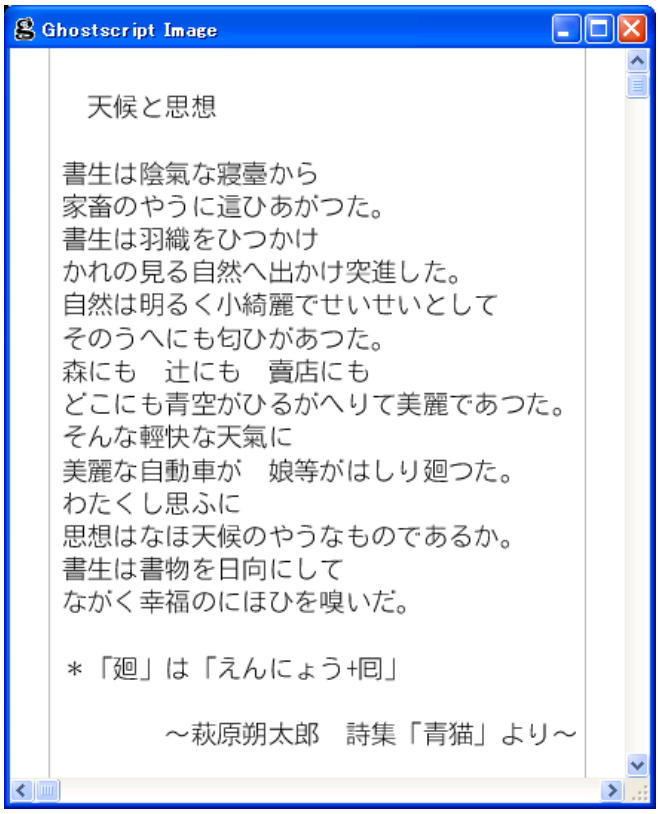



図 12: FMotoyaCedar-W1

ちなみに、近い将来、この「廻」の字を正しい字形で印字されるように JIS 第 4 水準までサポートする所存である。

### 4.8.1 【コラム】gs-cjk CIDFnmap を最大限活用するためには

gs-cjk がマージされている Ghostscript において、TrueType, CID フォント, OpenType を設定するファイルは、lib/CIDFnmap および、それからロードされる lib/CIDFnmap.*である。gs のデフォルトでは、ほとんどのエントリが「コメントアウト」されており、つまり、なんにも設定されていない。非 CJK フォントの設定ファイルである lib/Fontmap と異なり、Ghostscript で標準配布される CJK フォントも用意されていない。加えて、オペレーティング環境やユーザ環境によって所有しているであろう CJK フォントが異るので、CIDFnmap を適切に設定するための解説を書くのは困難を伴う。しかし、gs-cjk を最大限活用するためには CIDFnmap を書けないとはじまらないので、ここでは重要なポイントのみに絞って説明しよう。

```
%
% CIDFnmap
%
(CIDFnmap.Ore)  .runlibfile
(CIDFnmap.ARP)  .runlibfile
(CIDFnmap.Bae)  .runlibfile
(CIDFnmap.Koc)  .runlibfile
%(CIDFnmap.Sol) .runlibfile % for Solaris only
%(CIDFnmap.Win) .runlibfile % for Windows only

(CIDFnmap.CJK)  .runlibfile

(CIDFnmap.b5)   .runlibfile
(CIDFnmap.gb)   .runlibfile
(CIDFnmap.sj)   .runlibfile
(CIDFnmap.ksx)  .runlibfile
```

コメントアウトされていない `lib/CIDFnmap.*`は自由に手に入るフォントばかりなので、オペレーティング環境に依存しない設定は以上のようなものになる。重要なポイントは `CIDFnmap.CJK`, `CIDFnmap.b5`, `CIDFnmap.gb`, `CIDFnmap.sj`, `CIDFnmap.ksx` は必ず使うことである。次に `CIDFnmap.CJK` を編集する。編集のポイントは中身を見て、『コメントアウトされていない `lib/CIDFnmap.*`でエントリされているフォント名、つまり所有しているフォントの行のコメントを外す』ことである。こうすることで、目安としては、世の中で配布されているほとんどすべての PS/PDF ファイルが読めるようになる。欲張れば、より品質の高いフォントへ代替させることによって、かなりクオリティの高い CJK-ready な PostScript 処理系となろう。

重要なことを書き忘れたが、gs-cjk に関するどのサイト・文献にも書いてる基本事項として、筆者により動作確認済の Adobe CMap アーカイブ`acro5-cmaps-2001.tar.gz`もしくは`acro5-cmaps-2001.zip`と`adobe-cmaps-200211.tar.gz`もしくは`adobe-cmaps-200211.zip`を gs-cjk の ftp サイト [6] から取得し、gs の lib/gs_res.ps にて規定される `GenericResourceDir` フォルダに展開することを忘れないこと。通常は、lib/gs_res.ps を編集して、`GenericResourceDir` は gs をインストールしたフォルダ近辺に変更するので、そのとき、`FontResourceDir` も変更しておくとよい [30, 31]。CMap ファイルがないと CJK 処理は全滅となるので必須の作業である。

# 5 gs-cjk の現状と未来

## 5.1 トラブルシューティング「伝統的な日本語 OCF への対応」

CJK-ready な gs を用意しても、極めて稀に日本語 PS ファイルが処理できない場合がある。これは、その PostScript コードが昔の OCF フォントを前提に書かれていて、CID フォントへ移行した gs-cjk では不都合があるからである。その PostScript コードが CID と OCF の双方を考慮して `(begin|end)rearrangedfont` オペレータを使って書かれてあったとしても、現在の gs ではそのオペレータが実装されておらず、どちらの方策も処理できないという状況になっている。

問題を回避する方法を一つ用意したので紹介しよう。それは先にも紹介した「CID による日本語 OCF エミュレーション [19]」により、あたかも伝統的な日本語 OCF フォントがあるかのように見せかける方法である。

参考サイト [19] から `ocfcid-j-20020201.zip` もしくは `ocfcid-j-20020201.tar.gz` を取得して gs のフォントサーチパスが通っているフォルダに展開するだけである。展開されたファイル群はフェイクの OCF フォントである。

但し、これで問題が回避されたとしても、そのままにしておくのには大きな問題がある。先にも述べたように gs は OCF フォントを PDF へアウトライン埋め込みせずにビットマップとしてしまうので、このトラブルシューティングが終ったらフェイクの OCF フォントファイル群をどこか別のフォルダへ移動しておこう (無論、環境変数 `FONTPATH` を利用してフォントパスを目的に応じて切替えるのがベターな方法である)。

## 5.2 CJK TrueType フォント読み込み処理の高速化、ボールド化

先にモトヤ日本語 TrueType フォントの CIDFnmap での設定例を紹介したが、CIDFnmap を使わない、より高機能な方法を紹介しよう。

まず、参考サイト [20] から `install-cid-20020820.tar.gz` もしくは `install-cid-20020820.zip` を取得して、どの場所でも構わないので、ある作業ディレクトリに展開する。そして、その作業ディレクトリで、Unix 系の場合、

```
$ ./alias-aj1.sh install \
        FMotoyaCedar-W1p:=/pathto/MTXc1kp.ttc\
        FMotoyaCedar-W1:=/pathto/MTXc1kp.ttc,2\
        FMotoyaCedar-W1kp:=/pathto/MTXc1kp.ttc,3
$ ./alias-aj1.sh install \
        FMotoyaCedar-W1p-Bold=FMotoyaCedar-W1p,Bold\
        FMotoyaCedar-W1-Bold=FMotoyaCedar-W1,Bold\
        FMotoyaCedar-W1kp-Bold=FMotoyaCedar-W1kp,Bold
# 誌面の関係で\で折り返している。
```

Windows の場合、DOS プロンプトもしくはコマンドプロンプト上で

```
> alias-aj1.bat install \
        FMotoyaCedar-W1p:=C:\WINDOWS\Fonts\MTXc1kp.ttc\
        FMotoyaCedar-W1:=C:\WINDOWS\Fonts\MTXc1kp.ttc,2\
        FMotoyaCedar-W1kp:=C:\WINDOWS\Fonts\MTXc1kp.ttc,3
> alias-aj1.bat install \
        FMotoyaCedar-W1p-Bold=FMotoyaCedar-W1p,Bold\
        FMotoyaCedar-W1-Bold=FMotoyaCedar-W1,Bold\
        FMotoyaCedar-W1kp-Bold=FMotoyaCedar-W1kp,Bold
# 誌面の関係で\で折り返している。
```

とする。これでかなり速度向上が見込めるはずである。その上、擬似的に太文字化した FMotoyaCedar-W1p-Bold, FMotoyaCedar-W1-Bold, FMotoyaCedar-W1kp-Bold フォントが作成できる。これらは、CIDFnmap では出来ない機能である。FMotoyaCedar-W1-Bold での縦書き印字例を示しておこう。

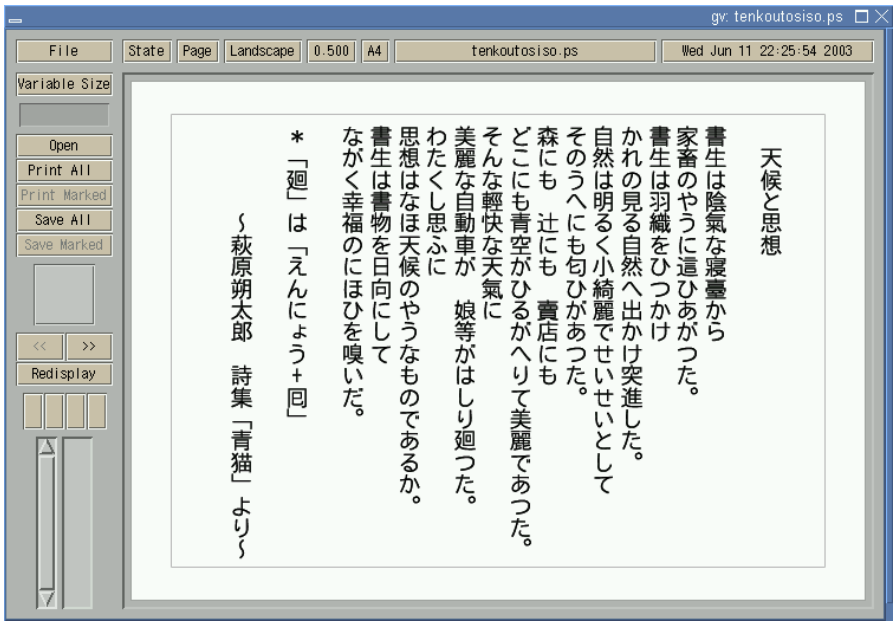



図 13: FMotoyaCedar-W1-Bold

ところで、FMotoyaCedar という PS フォント名はどうやって決めたのか疑問ではないだろうか。ユーザの好きな名前でよいのであるが、以下のように、TrueType フォント内に書いてある PS フォント名を参考にするとネーミングに困らない。

```
% gsnd -q -dBATCH -dECHOONLY -sTTFONT=/pathto/MTXc1kp.ttc ttcf2cid.ps
CIDFontName: (FMotoyaCedar-W1-90msp-RKSJ-H)
% gsnd -q -dBATCH -dECHOONLY -dTTFONT='{() (/pathto/MTXc1kp.ttc) 2}' ttcf2cid.ps
CIDFontName: (FMotoyaCedar-W1-90ms-RKSJ-H)
% gsnd -q -dBATCH -dECHOONLY -dTTFONT='{() (/pathto/MTXc1kp.ttc) 3}' ttcf2cid.ps
CIDFontName: (FMotoyaCedar-W1-90mskp-RKSJ-H)
```

だったら自動的に名前を決めてくれればよいのでは、と思われるかもしれないが、これはオペレーティング環境のファイルシステム (ファイル名に使用可能な文字等) に関わる上に、上例のように CID フォントとしては不適当な PS フォント名が登録されているので、ユーザに決めてもらうようになっている。一方で、Adobe Acrobat や Distiller では、AdobeFnt.lst というファイルを自動的に生成してフォントの実体とフォント名の対応づけをしているようである。

## 5.3 本家統合のいくつかの戦略案について

本家統合のいくつかの戦略案について、ここで箇条書きでいくつかアイデアを記しておく。主たる統合のテーマは「CIDFnmap による簡便なフォント名とフォント実体の束縛機構」、「TrueType を CID フォントとして使う技術とその品質」についてである。

⑴ gs-cjk 製 CIDFnmap そのものを統合する (Fontmap だってそのままあるじゃないか!)
⑵ 本家製 FAPI+cidfmap をなんとか gs-cjk に近付けて統合する
- TTF-CID グリフ変換アルゴリズムの再導入…
- 縦書きグリフ抽出ルーチンの再導入…

⑶ Fontmap, gs-cjk 製 CIDFnmap を統合し、TrueType のようなホストコンピュータフォントで CID フォントを理想的な方式で代替するように新たに書き換える
⑷ 初心にかえって、むしろ FAPI を楽しむ (FreeType2, VFlib3, W32API, iconv, …)
- 外部ライブラリ FreeType2 等への TTF-CID グリフ変換アルゴリズムの導入…

⑸ CJK TrueType は使えないことにしてしまう…
⑹ 古き善き OCF の時代に戻る…
⑺ 高価な CJK CID, OpenType フォントで我慢する…

まったく考えがまとまっていないことをお許し頂きたいが、おそらく⑶案「Fontmap, CIDFnmap の統合」がもっともやりやすく、理想的なコーディングもしやすいように考えている。しかし、本家開発者や CJK ユーザの声の中には FAPI を実用の域へ持っていって欲しいというのもあろうかと思われ、となると⑵か⑷案ということになるが、理想的なソリューションかどうかはいずれも疑問である。とは言え、長い間実装されてない `(begin|end)rearrangedfont` オペレータが導入されれば (導入すれば)、本家 gs8 の CJK まわりの実装を軟着陸させることは可能かもしれない。

### 5.3.1 【コラム】筆者の個人的な意見

gs8 の FAPI は、なんらかの方法で TrueType を用いたとしても、縦書きグリフが用意されておらず、しかも、CID フォントと同等に使うには正しい代替グリフが少な過ぎる。gs-cjk がやったように巧妙に代替グリフを割り当てる技術も導入しにくい。この点ははっきり言って技術の後退である。是非、日本のユーザ、いやアジアのユーザは gs-cjk のこの成果が統合されるよう本家へ圧力をかけて欲しい… というのは、筆者のかつての意見であり、同調できるか否かは自ら判断して頂きたい。というのも、TrueType を CID フォントとして使うというのはそもそも Poor People's Solution であり、CID フォントの豊富なグリフを TrueType フォントでサポートするのは、実用上はほとんど問題は表れないものの、所詮フェイクに過ぎない技術であるからである。

# 6 おわりに

Ghostscript を中心に、ページ記述言語 PostScript や PDF、CJK リソースに関する話題を広く紹介した。かねてより、PostScript は単なるプリンタ制御コードの一種であり、複雑な整形処理はホストコンピュータに任せて、非力な PostScript プリンタへ流すコードは単純・安全であるべき、という風潮は根強くある。確かに PS ファイルで入稿しなければならない場合等ではそういった可搬性には十分配慮すべきではあるが、PDF 全盛となった現在では、そういった PostScript の利用はいささか消極的であるように筆者は思っている。本論で述べたように、ホストベースプリンタが主流となっている現在、PostScript 処理系は非力ではないホストコンピュータ側にあり、最終的に PDF ファイルの形式になっていれば可搬性の問題もほとんど関係ない。それよりも、PostScript を 2 次元コンピュータグラフィックスが最も整備された高級言語と捉え、それを最大限活用するメリットは計り知れない。しかも、PostScript は他のコンピュータグラフィックス API にも多大な影響を与えており、PostScript で培ったグラフィックスのノウハウは他の場面でも十分に使える。PostScript をこのように 2 次元グラフィックス最大利用の場と捉えた場合、フリーの処理系である Ghostscript はかなり重宝する。

その上、2 次元コンピュータグラフィックスでの要であるフォント処理も PostScript では CJK へ実用的にローカライズされており、CJK フォントさえ手に入れれば漢字文化圏での文書処理にも十分活用できる。Ghostscript では CJK への実用的なローカライズが遅れていたが、gs-cjk によりある程度は使えるようになり、少なくとも gs-cjk の活動内容が本家へ影響を与えているのは紛れもない事実である。gs-cjk が完全に本家統合されることを一般ユーザが待ち望んでいるらしいことを筆者は理解している。

ただ一方で、gs-cjk 開発者のメーリングリストには、日本のユーザの声が直接届くことは少なく、むしろ TeX 関係の掲示板の方へ gs ユーザの素朴な疑問等が集まっているようである。それはそれで無論構わないのだが、掲示板という性質上、拙速なハックについての議論や対処には十分である一方、開発の重要な方向性を議論したりするにはそちらは不向きであるように感じている。同時に、議論に深みがないので、ユーザに影響力のある方が不適当なコメントをしたまま放置されることも少なくない。一方で、ユーザを甘やかしているからいつまでたっても支援者が育たない。根本的な問題は棚上げにされ、おそらく gs-cjk が重要な案件を抱えているということさえ気付いておらず、そのまま待っていれば gs-cjk が永久に統合されるようになると勘違いしているのかもしれない。一般ユーザはそれで構わないのだが、現在の gs の最大のプレーヤーである TeX において重要な立場にいるような影響力のある方が、議論なしで VFlib 版日本語化 gs を放棄して gs-cjk へ乗り換えたり、一方で日本語ローカライズに固執していたり、「やってみましたダメでした」的な作業を繰り返していたり、開発よりも本や雑誌の執筆のみに注力していたり、「そんなに実力があるんならコードを書けばいいのに」というのは、門外漢である未熟な筆者のかつての「愚痴」であった。「あなた方、情報系の専門家なんだから、僕なんかよりうまくやれるでしょう？」というわけである。

そんなことを考えているうちに、筆者も多忙になってしまって、実は一年以上も gs-cjk に関するコードは書いていなかったりして、一旦離れた立場で、gs-cjk が一人歩きし始めたのを眺めている最中である。そして今回、本稿を書く機会を与えられ、寝かされていたアイデアをある程度整理する結果となった気がしている。最近、もうずいぶんと過去に感じられる我々の成果が雑誌等で紹介され、筆者が意図していた価値観を共有するコンテキストで使われているのを知り、日本ではじめて理解されたと非常に嬉しく思った。他のオープンソースプロジェクトもこんなものなのかもしれないが、意外とユーザの声というのはそれほど届かず、開発者は結

構不安にかられていたりするものである。

知識が乏しくやみくもに開発していたほうが、迷いがなくて成果が上るケースもあるだろう。思い起こしてみると、今、gs-cjk の成果としてみなさんがお使いのものは、そういった状況の中から生み出されたものである。オープンソース開発はある程度無責任に気楽に関わるプロジェクトであるのだから、gs-cjk によって方向性が既に世に示された今、気楽に自分のしたいことを自由にやればよいのである。それに、自分の好きなようにいじれる場が用意されていることは幸福なことであり、それを利用しない手はないし、プログラミングの勉強にもなる。是非、gs-cjk プロジェクトに興味がある方は参戦していただきたい。参加方法はあなたの好きなような関わり方で構わないが、gs-cjk のサイト [6] で紹介されているメーリングリストにあなたのメールアドレスを登録するのが一般的な参加方法である。本稿に書いてあることを漠然とでも理解でき、かつ、有意義で楽しいプログラミングがしたいなら、もうあなたは既に gs-cjk プロジェクトメンバである (なんちゃって)。

### 【コラム】漢字文化圏におけるフォントの問題

漢字には、大きく分けて、古くから人々の手で脈々と書かれてきた楷書体と、活字として使われる康熙字典体を源流とする明朝体の 2 系統がある。人の手による運筆の描像である楷書体はかつて完成の域に達したが、運筆とは無関係な明朝体との影響を相互に受け、これらの系統は徐々にではあるがその形は変化している。言葉が生き物であるように、ひとつひとつの文字それぞれに意味をもつ漢字もやはり生き物であり、少しづつだが成長もしくは増殖しているのである。

漢字文化圏におけるフォント製作のコストは、ラテン文字圏のそれとは比較にならないほど高く、結果として表現のバリエーションの幅が狭まれている。ふと、「手で書いていた時代に戻ればよいのではないだろうか」と思ってしまうが、それでは技術の進歩がない。そもそも、活字というものは歴史上においても印刷業者による所有主 (proprietary) の側面がある。なぜ、もともと古くから人々の手で脈々と書かれてきた漢字が誰かの所有物であるのか、これだけディジタル技術が進歩した今、筆者は危機感を感じざるを得ない。

また一方で、ひとつの漢字をあるひとつのコードポイントに割り当てる「文字コード」は、これから先も永久にコード空間を広げていくのだろうか。文書の情報交換にはそれがディジタル処理にとって簡便とはいえ、漢字文化にはそぐわないと筆者は考えており、将来、運筆の描像を直接用いた情報交換方式が有り得るのではないかと考えている。そして、non-proprietary な漢字フォントの自由で緩やかな流通の可能性もその辺にあると筆者は考えている。

## 参考文献


[1] 安岡 孝一, 安岡 素子, “文字コードの世界,” 東京電機大出版局, September 1999.

[2] 江守 賢治, “解説 字体辞典 (普及版),” 三省堂, June 1998. 江守 賢治, “硬筆毛筆書写検定理論問題のすべて,” 日本習字普及協会, September 1995.

[3] Ken Lunde, “CJKV Information Processing,” ÓReilly & Associates, Inc. January 1999.

[4] 小林 龍生, 安岡 孝一, 戸村 哲, 三上 喜貴 編, “インターネット時代の文字コード,” Kyoritsu Shuppan Co., Ltd., January 2002.

[5] 小林 龍生, “UniCode と ISO10646,” `http://www.kobysh.com/tlk_old/bit.htm`.

[6] “gs-cjk project,” `http://www.gyve.org/gs-cjk/`,

ftp://ftp.gyve.org/pub/gs-cjk/, November 2000, gs6.50-cjk-M1-R1 April 7 2001, gs6.50-cjk-M1-R2 April 17 2001, gs6.51-cjk-M2-R1 June 29 2001, gs6.51-cjk-M2-R2 October 5 2001, gs6.51-cjk-M2-R3 December 7 2001, gs6.53 February 5 2002, 2003, gs7.05 April 22 2002, 2003, gs7.06 April 1 2003, gs7.07 May 18 2003.

[7] Adobe Systems Incorporated, "PostScript Language Document Structuring Conventions Specification Version 3.0,"
http://partners.adobe.com/asn/developer/pdfs/tn/5001.DSC_Spec.pdf.

[8] Adobe Systems Incorporated, "PostScript LANGUAGE REFERENCE third edition," Addison-Wesley Publishing Company,
http://partners.adobe.com/asn/developer/pdfs/tn/PLRM.pdf.

[9] Adobe Systems Incorporated, "PDF Reference third edition, Adobe Portable Document Format, Version 1.5 Draft Specification,"
http://partners.adobe.com/asn/acrobat/docs/filefmtspecs/PDFReference15draft.pdf.

[10] Microsoft Corporation, "OpenType specification,"
http://www.asia.microsoft.com/typography/otspec/.

[11] Adobe Systems Incorporated, "Adobe-CNS1-4 Character Collection for CID-Keyed Fonts," Technical Note #5080,
http://partners.adobe.com/asn/developer/pdfs/tn/5080.Adobe-CNS1-4.pdf.

[12] Adobe Systems Incorporated, "Adobe-GB1-4 Character Collection for CID-Keyed Fonts," Technical Note #5079,
http://partners.adobe.com/asn/developer/pdfs/tn/5079.Adobe-GB1-4.pdf.

[13] Adobe Systems Incorporated, "Adobe Japan1-4 Character Collection for CID-keyed Fonts, Technical Note #5078,"
http://partners.adobe.com/asn/developer/pdfs/tn/5078.Adobe-Japan1-4.pdf.

[14] Adobe Systems Incorporated, "Adobe Japan1-5 Character Collection for CID-keyed Fonts (Addendum), Technical Note #5146,"
http://partners.adobe.com/asn/developer/pdfs/tn/5146.Adobe-Japan1-5.pdf.

[15] Adobe Systems Incorporated, "Adobe-Japan2-0 Character Collection for CID-Keyed Fonts," Technical Note #5097
http://partners.adobe.com/asn/developer/pdfs/tn/5097.Adobe-Japan2-0.pdf.

[16] Adobe Systems Incorporated, "Adobe-Korea1-0 Character Collection for CID-Keyed Fonts," Technical Note #5093
http://partners.adobe.com/asn/developer/pdfs/tn/5093.Adobe-Korea1-0.pdf.

[17] Taiji Yamada, "PostScript CMap visualization tools,"
http://www.aihara.co.jp/~taiji/tops/cmaptools/.

[18] Taiji Yamada, "Text filter into PostScript — tops,"
http://www.aihara.co.jp/~taiji/tops/#tops.

[19] Taiji Yamada, "Japanese OCF emulation by using CIDFont,"
http://www.aihara.co.jp/~taiji/tops/ocf-j.html.

[20] Taiji Yamada and gs-cjk project, "install-cid.tar.gz and install-cid.zip for gs-cjk-merged Ghostscript,"
http://www.aihara.co.jp/~taiji/tops/install-cid-j.html, August 2002.

[21] Adobe Systems Incorporated, "Chapter 4 - Emulators and Translators," in "PostScript Language Program Design," Addison-Wesley, pp. 59–75, 1988.

[22] Adobe Systems Incorporated, "Adobe Glyph List,"

http://papers/partners.adobe.com/asn/developer/type/glyphlist.txt, September 2002.

[23] gs-cjk project, “Ghostscript CJK supplement information,” http://www.gyve.org/gs-cjk/supplement/.

[24] suzuki toshiya and gs-cjk project, “gs/doc/CJK.htm: Features to support CJK CID-keyed font in Ghostscript,” in gs-cjk-merged GNU Ghostscript documents, Dec 2001.

[25] Taiji Yamada and gs-cjk project, “gs/doc/CJKTTCID.htm: Adobe CIDs and glyphs in CJK TrueType font,”in gs-cjk-merged GNU Ghostscript documents, Dec 2001.

[26] Timothy van Zandt, “PSTricks,” http://www.tug.org/applications/PSTricks/.

[27] 山田 泰司, “日本語 Text Tips on PSTricks,” http://www.aihara.co.jp/~taiji/tex/.

[28] 森 茂樹、“PostScript 詳細解説,” CQ Publishing Co., Ltd., December 1997.

[29] “知りたい疑問に答えます — OpenType フォントのホント,” I/O 別冊 Professional DTP, 工学社, pp. 65–94, April 2002.

[30] 温 恵一、安宅 正之、“日本語環境における印刷のポイントを大公開 — プリント大全,” Softbank Publishing Inc., pp. 30–55, January 2003.

[31] 編集部 渡辺、安宅 正之、“UNIX 用ドライバなしでも何とかするぞ！ — ドライバレス仮想プリンタ,” Softbank Publishing Inc., pp. 60–73, June 2003.

[32] 中野 賢, “日本語 LaTeX2e ブック,” アスキー, November 1996.

[33] 奥村 晴彦, “LaTeX2e 美文書作成入門,” 第 2 版, 第 2 刷, November 2001.