書いてみよう！ 組版処理系

文　編集部　いなにわうどん

課題に忙殺される筑波大の学生は日夜、レポート作成に際して「Microsoft Word では文書のクオリティが――でも LaTeX よりも直感的に使いたい……」とお悩みのことと思います。この夏に、組版システムを独自に実装してみるのはいかがでしょうか。

1　組版ってなに?

組版とは？ を入れる

組版は空気のような存在であり実際に意識されることは少ないですが、文章一つを組むにも複雑な規則が存在していることは忘れてはなりません。例えば、

こ　の　よ　う　な　体　裁　で　文　章　を　組　ま　れ　た　ら

、ど　う　感　じ　る　でしょうか？ 読みにくいことこの上ないですよね。

1.1　日本語組版は難しい!?

日本語は縦書きと横書きが混在する稀少な言語であり、特に特殊な組版処理が必要とされます。例えば、縦書きの文章に英数字が混入した場合は、正立のまま配置するのも手ですし、数字だけ横書きにする縦中横と呼ばれる手法もあります。単語を 90 度倒立させて配置するのも一般的です（図 1）。

図 1　縦書き中に欧文が混在する処理

漫画の写植（台詞）部分に目を向けてみると、漢字の部分にはゴシック体が、ひらがなにはアンチック体[*1]が充てがわれていることが判ります。これは混植と呼ばれる技術で、テキストをかな、漢字、欧文、約物[*2]類といった**文字クラス**に分類し、それぞれに異なるフォントを適用するものです。文字クラスは多くの処理に利用されており、例として行分割処理に際しては、行頭行末に位置する文字の文字クラスを基に、禁則処理[*3]を実行します。

[*1]明朝体の系統を汲んだかな書体であり、漫画や辞書等に多く採用されています。

[*2]句読点や括弧に代表される記号のこと。

[*3]行頭・行末に行頭禁止文字（句読点、閉じ括弧類、撥音、促音等）・行末禁止文字（開き括弧類等）がそれぞれ

こうした細密な組版要件を網羅的に調査、文書化したものが W3C[*4] より、「日本語組版処理の要件」[1] として公開されています。Requirements for Japanese Text Layout より JLReq が通称です。

1.2 組版処理の流れ

さて、日本語組版の実現にあたって必要な最小限の流れは以下の 3 つです。本記事では OpenType フォントを入力として与え、PDF で出力することを想定しています。

1. フォント情報を取得する
2. 組版処理・版面の計算
3. PDF として出力

2 フォント情報の取得

フォントのパース機能は OS レベルでも提供はされているものの、高度な OpenType 機能や日本語組版に特有の機能への対応は十分ではなく、正直あまり期待はできません。リガチャ[*5]も怪しいレベルです[*6]。

したがって、高度な組版処理の実現には自前でのフォントファイルのパースが欠かせません。本記事ではフォントファイルのフォーマットに関しては概説程度に留め、実装に関しては **opentype.js**[2] に投げることとします。OpenType とありますが、TrueType についても解釈してくれる優れたライブラリです[*7]。

2.1 OpenType フォントの仕様

OpenType フォントは、各種情報を含んだ複数の**テーブル**と、各テーブルへのオフセット情報からなるヘッダによって構成されます。日本語組版で必要となるテーブルを表 1に示しました。

幸い PDF では CID を指定すればレンダリングについては担当してくれますので、CID とメトリクス情報を取得できれば問題はありません。

2.2 opentype.js

今回の記事では JavaScript を用いて、ブラウザ上でフォントのパースを行うことにします。opentype.js の Readme にもある通り、CDN から読み込みます。（Listing 1）

出現した際には、約物の前後のアキや行中の字送りを調節し、改行位置を変更する処理が通例です。

[*4] World Wide Web Consortium. Web 技術の標準化団体。`https://www.w3.org/`

[*5] fi, ffi, st 等の特定の 2 文字が並んだ際、2 文字を 1 つのグリフに置換する処理。すなわち合字。

[*6] 標準「テキストエディット」から OpenType 機能を利用できるなど、Mac OS では比較的高水準な組版の素地が整備されているように見受けれられます。故スティーブ・ジョブズ氏がタイポグラフィに興味があったという逸話は有名です。一方で……

[*7] OpenType は TrueType のフォントデータに CFF（Compact Font Format）と呼ばれる PostScript 形式のアウトラインデータを融合させたものであるため、TrueType のスーパーセットと考えることも出来ます。

[*8] フォント内におけるグリフ番号。詳細は後述。

表 1　日本語組版に必要となる主要 OpenType テーブル

テーブル	解説
post, name	フォントの基礎情報。
cmap	CMap（文字コード↔CID[*8]の対応テーブル）情報。
hhea/vhea	水平／垂直方向のレイアウト情報。文字幅の情報を取得する。
GSUB	字形置換の情報。単純置換、複数置換、代替置換、合字置換が代表的。
GPOS	文字詰め情報。プロポーショナルメトリクスとペアカーニングに分かれる。

Listing 1　opentype.js の読み込み

```
<script src="https://cdn.jsdelivr.net/npm/opentype.js@latest/dist/opentype.min.js
    "></script>
```

フォント情報を取得します。`psName` は PostScript 名と呼ばれる情報で、PDF 上でフォントを使用するにはこちらの名称を使用します。また、アセンダ（ascender）・ディセンダ（descender）はそれぞれベースラインから上底・下底までの高さを表しています[*9]。（Listing 2）

Listing 2　フォントの解析

```
const src = "FOT-HummingPro-B.otf";
opentype.load(src, (error, font) => {
  if (error) {
    console.warn(error);
  }
  else {
    const psName = font.tables.name.postScriptName.en;
    console.log(psName); // HummingPro-B
    console.log(font.ascender, font.descender); // 880, -120
  }
});
```

CID・文字幅の取得

実際の文字情報（グリフ）を取得するには、`cmap` テーブルを利用します。OpenType フォントでは、CID（Character Identifer）と呼ばれる固有の番号によってグリフが管理されており、Unicode 等を始めとするエンコードと CID の変換テーブルにあたるのが CMap になります。

CID の文字コレクションには、Adobe-Japan1-x と呼ばれる文字コレクションが利用されることが殆どです。「ヒラギノ角ゴシック **Pro** W3」といった具合に、Std, Pro, Pr6N 等の記述を含んだフォント名を目にしたことはないでしょうか。これは、フォントが準拠する文字コレクションの違いを表しています。

[*9] タイポグラフィ的には別の解釈もありますが、OpenType に定義される `ascender, descender` の情報は先述の定義に基づきます。

本来であればここで **GSUB テーブル**を利用した字形置換についても触れるべきですが、今回は紙面の制約上割愛します。GSUB テーブルを利用することで、Unicode 上からは利用できない包摂[*10]されたグリフの表現が実現されます。

文字幅（`advance width`）等のメトリクス情報は、本来 `hmtx`（垂直方向の場合は `vmtx`）テーブル上に定義されていますが、opentype.js では `glyph` オブジェクトから取得できます。

Listing 3　CID・文字幅の取得

```
1  const cid = font.charToGlyphIndex(char);
2  const advanceWidth = font.glyphs.glyphs[cid].advanceWidth;
```

3　PDF に描画する

PDF（Portable Document Format）はデバイスに描画を依拠せず完全な互換性を保持できる点が評価され、今日の文書交換フォーマットのデファクトスタンダードとして君臨しています。そして実は PDF、**手書きでも書けます**。

適当なエディタに Listing 4のコードを打ち込み、`sample.pdf` などに保存してください。お手持ちの PDF リーダーで、A4 サイズの白紙が表示されるはずです。

Listing 4　白紙の PDF 文書を生成

```
1  %PDF-1.7
2
3  1 0 obj
4  <<
5    /Type /Catalog
6    /Pages 2 0 R
7  >>
8  endobj
9
10 2 0 obj
11 <<
12   /Type /Pages
13   /Kids [ 3 0 R ]
14   /Count 1
15 >>
16 endobj
17
18 3 0 obj
19 <<
20   /Type /Page
21   /Parent 2 0 R
22   /MediaBox [ 0 0 595.3 841.9 ]
23   /Contents 4 0 R
24   /Resources 5 0 R
25 >>
26 endobj
```

[*10]「あ」の文字は、Unicode 上では U+3041 の 1 つのみですが、CID 上では普通のあ、半角のあ、丸文字のあ……と数多の字形が定義されています。このように、フォント内には異体字が存在するものの、文字コード上では同一の符号位置に割り当てられることを包摂といいます。

```
27
28 4 0 obj
29 << /Length 0 >>
30 stream
31 endstream
32 endobj
33
34 5 0 obj
35 << >>
36 endobj
37
38 xref
39 0 6
40 0000000000 65535 f
41 0000000009 00000 n
42 0000000062 00000 n
43 0000000128 00000 n
44 0000000227 00000 n
45 0000000227 00000 n
46
47 trailer
48 <<
49   /Size 6
50   /Root 1 0 R
51 >>
52 startxref
53 296
54 %%EOF
```

PDF は *no* `0 obj ... endobj` で定義される**間接オブジェクト**（Indirect Object）の集合からなり、*no* `0 R` の形で間接オブジェクト同士が参照を繰り返すことで、1 つの文書を構成するタイプが一般的です。ファイル末尾にはクロスリファレンステーブルが置かれ、間接オブジェクトのオフセットが指示されます[*11]。

<< /*key value*0 *value*1 ··· >> の形で記述されるのが、辞書と呼ばれるオブジェクトです。ドキュメントのルート要素は、トレーラー（trailer）と呼ばれる辞書になります。なお PDF では寸法を pt（ポイント、1pt = 1/72inch）で指定し、左下が原点となります。

詳しくは、PDF Reference[3] に仕様が公開されています。ただし原文は 1300 ページ以上の英文となりますので、要点を和訳した文書 [4] を参照されることをおすすめします[*12]。

3.1 グラフィックスとオペレータ

Listing 4の `4 0 obj` 以降に続く、`stream ... endstream` の領域は**コンテンツストリーム**（Content Streams）と呼ばれます。PDF には PostScript を起源とした 74 個の命令（Operator）が存在しており、コンテンツストリーム内に命令を記述することでページ上に表示されるオブジェクトを制御しています。

[*11] この実装により、PDF 文書における各ページへのランダムアクセスが可能となっています。
[*12] 同サイトには OpenType の仕様書訳文も掲載されているので、そちらもご参照ください。

ひとまず、テキスト周りの記述を抑えておきましょう。Listing 5の出力結果が、図 2になります。

Listing 5　テキストの描写

```
stream
BT
/Helvetica 40 Tf
100 100 Td
(University of Tsukuba) Tj
ET

5 0 obj
<<
  /Font << /Helvetica 6 0 R >>
>>
endobj

6 0 obj
<<
  /Type /Font
  /Subtype /Type1
  /BaseFont /Helvetica
>>
```

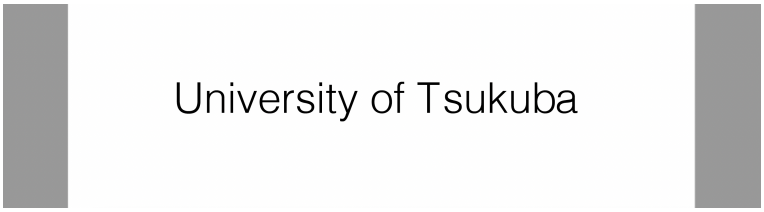

図 2　テキストの描写結果

BT, ET で囲われた部分はテキストとして認識されます。*number number* Td（*number* は整数または実数。符号は問わない。）で座標を指定し、続いて (*String*) Tj を用いてテキストを描画する流れとなります。

フォントの指定には /*FontName* Tf を利用しますが、この際 Listing 5の 6 0 objにみられる**フォント辞書**を定義し、Page オブジェクトの Resources に対して Font を登録する必要があります。欧文フォントの場合は SubType を Type1 に指定します。

3.2　和文フォントの指定

日本語を指定する場合は、より複雑なフォント辞書が要求されます。SubType は Type0 になり、さらに DescendentFont に CIDFont 辞書を定義します。CIDFont 辞書からはさらに、FontDescriptor を参照させます。（Listing 6）

Listing 6　和文フォントの指定

```
7 0 obj
<<
```

```
  /Type /Font
  /Subtype /CIDFontType0
  /BaseFont /HummingPro-B
  /CIDSystemInfo <<
    /Registry (Adobe)
    /Ordering (Japan1)
    /Supplement 4
  >>
  /FontDescriptor 8 0 R
>>
endobj

8 0 obj
<<
  /Type /FontDescriptor
  /FontName /HummingPro-B
  /Flags 1
  /FontBBox [ 0 0 1000 1000 ]
  /ItalicAngle 20
  /Ascent 880
  /Descent -120
  /CapHeight 0
  /StemV 0
  >>
  endobj
```

フォント辞書の Encoding には、Unicode を利用する場合は UniJIS −UTF16 −H を指定します。今回はフォントに内容される CMap を経由してグリフを指定するため、Identify-H を指定します[*13]。CIDSystemInfo には、先述した文字コレクションの情報を記述します。

説明を省いた項目もありますが、基本的には OpenType フォントから同名のフィールドを探してきて値をそのまま代入すれば大丈夫です。フォントを埋め込む場合は、さらに CFF（Compact Font Format）を解析して、埋め込みサブセットを作成する必要があります[*14]。

そのほか、日本語組版処理に関係がありそうな命令を表 2に抜粋しました。

レンダリングモードでは、塗り（フィル）や線（ストローク）といった装飾に関する項目を指定します。$T_r = 0$ は塗りのみ、$T_r = 1$ では線のみの描画となります。また仕様書によれば、$T_r = 2$ の場合は Fill then stroke となるため、塗り＋縁取りが実現できる——かと思いきや、フィル→ストロークの順で描写を行うため、塗りの部分が侵食されてしまいます。したがって、$T_r = 1$ の状態でまず描写し、続いて $T_r = 0$ に切り替えて描写を加えることで、予期した通りの描写が実現できます（図 3)。

[*13] /Encoding に指定している内容が CMap になります。UniJIS-UTF16-H は Adobe-Japan1 の文字集合において、UTF-16 を用いて横書き用に指定するといった意味合いを持ちます。Identify-H/V では文字通り、記述した内容と同一のグリフが利用されます。

[*14] 和文フォントでは、2 万字以上のグリフを含む場合もざらですので、グリフ数を削減しないと PDF が重すぎて開くのに数十秒と待たされる羽目になります。加えて多くの和文フォントでは制約が厳しく、サブセット化しないとライセンスに引っ掛かる恐れもあり……日本語の取り扱いは茨の道です。

表 2　日本語組版に関連した PDF オペレータ

構文	概要	備考
/name number `Tf`	フォント、テキストサイズ	
number `Tl`	行送り	
number `Tc`	文字間（トラッキング）	
number `Tz`	水平方向の縮尺	正体のとき、$T_z = 100$。100 以下で長体。
number `Ts`	ベースラインシフト	
number `Tr`	レンダリングモード	詳細は後述。デフォルトでは $T_r = 0$

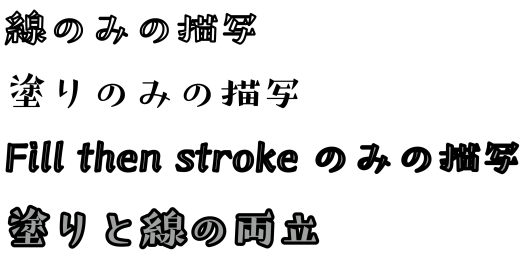

図 3　PDF 上での縁取り

4　むすびにかえて

以下に、今回紹介できなかった代表的な組版処理の項目を列挙します。文字→行→ページ単位での処理になるほど、実装の難易度は跳ね上がります[*15]。

文字単位での調節を必要とする処理

縦中横、合字（リガチャ）、斜体・回転処理、カーニング、ルビ処理、文字アキ量設定、合成フォント

行単位での調節を必要とする処理

禁則処理、ぶら下がり、ハイフネーション、割注処理

ページ単位での調節を必要とする処理

段組み・段抜き処理、脚注処理

4.1　欧文組版

欧文組版では、日本語よりも考慮すべき材料は少ない一方、行分割処理やハイフネーションに関して煩雑な処理が要求されます。幸い、こちらに関しては TEX で採用されている Knuth-Plass[5]（欧文処理における行分割処理アルゴリズム）を初めとする、多数のアルゴリズムや様々な言語においての実装も整備されています。

今回紹介した事柄はほんの触りに過ぎず、実用に耐えうる組版システムの開発には本記

[*15] ただし、実際は文字単位の調節のはずが、気が付けば行分割処理に関与していたりと、明確なラインを引くことは難しいものです。

事の数十倍以上に亘る実装が必要となります。LaTeX に疲れた方、他の文書作成ソフトウェアでは機能不足と感じた方、そして組版に若干でも興味を持たれたみなさん！！！ ぜひ奥深き自作組版処理システムの世界へ足を踏み入れてみてはいかがでしょうか。

5 参考文献

1. W3C Working Group (2020), "Requirements for Japanese Text Layout," https://www.w3.org/TR/jlreq/
2. opentypejs, "opentypejs/opentype.js: Read and write OpenType fonts using JavaScript.," https://github.com/opentypejs/opentype.js
3. Adobe System Incorporated (2006), "PDF Reference sixth edition," https://ghostscript.com/~robin/pdf_reference17.pdf
4. Azel, "PDF フォーマット," https://aznote.jakou.com/prog/pdf/index.html
5. Donald E. Knuth and Michael F. Plass (1981), "Breaking Paragraphs into Lines," http://www.eprg.org/G53DOC/pdfs/knuth-plass-breaking.pdf