TEX の出版への応用
— 縦組み機能の組み込み —

濱野 尚人* 田村 明史* 倉沢 良一†

概 要

日本語 TEX を出版分野で本格的に応用するため, 縦組み機能を追加した. 拡張された TEX は以下のような特長を持つ.

- 同一文書, 同一ページに縦/横組みの混在ができ, 縦組み中でも和/欧文/数式の混在が可能である.
- 英語版のオリジナル TEX や（横組み）日本語 TEX と互換性があり, 従来の TEX の文書やマクロをそのまま処理することができる.
- 縦組みを使用すると, 拡張フォーマットの `DVI` ファイルを出力する. ただし, 横組みのみ使用している場合は, 従来のフォーマットの `DVI` ファイルを出力する.

本論文ではその設計と実現方法について説明する.

1 はじめに

TEX はスタンフォード大学のクヌース教授によって開発された組版システムであり, 組版の美しさと強力なマクロ機能を特徴としている. 日本語化され [3], 高品位の写植出力を得ることができるようになり, わが国でも TEX によって組版された書籍が出版されるようになってきた.

ところが, 従来の日本語 TEX は英語向けのシステムをベースとしているため, 縦組み, ルビなどの日本語特有の機能がない. TEX を一般書籍の出版に使用するためするには, これらの機能を持った「本格的出版に耐え得る日本語 TEX」が必要であろうと判断し, 日本語 TEX の改造に取り組むことにした. この機能拡張された TEX を pTEX（publishing TEX）と呼ぶことにする. まず, 機能拡張第一弾として（株）アスキーで開発された日本語 TEX に縦組み機能を追加した.

本論文ではその設計と実現方法について報告する. 次章から 6 章までで pTEX の縦組み機能についての基本的な概念を説明し, 7 章で具体的な実現方法を説明する.

* （株）アスキー 出版局 電子編集研究統轄部 出版技術部
† （株）アスキー システムソフトウェア事業部 技術統轄部 オフィスシステム技術部

2　ディレクション

文書を組版すると, いくつかの ‘ページ’ の集合となる. ‘ページ’ は ‘行’ を “行送り方向” に並べたものであり, ‘行’ は ‘文字’ を “字送り方向” に並べたものである.

従来の TEX では, ページは vertical ボックス, 行は horizontal ボックスであり, “行送り方向”, “字送り方向” は横組み用に固定されていた. pTEX では, TEX の状態として “組み方向”（ディレクション）を考え, ディレクションによって ‘行送り方向’, ‘字送り方向’ を変えることにした. 縦組みは, 横組みの “行送り方向”, “字送り方向” を, 右に 90° 回転させたものと同じになる.

横組み 字送り方向: 水平右向き, 行送り方向: 垂直下向き

縦組み 字送り方向: 垂直下向き, 行送り方向: 水平左向き

また, 縦組み中で使われた場合と横組み中で使われた場合で動作が変わるようなマクロが書けるように, `\iftdir`, `\ifydir` というマクロも用意した.

3　ボックス

オリジナルの TEX では, 文字を並べて “行” のボックスを作るモードを h モードと言い, できたボックスを h ボックスと呼ぶ. これは TEX がアメリカで開発されたシステムであり, 英語では文字が水平（horizontal）に並ぶからである. ディレクションが横の場合はそのままで問題ない. ディレクションが縦の場合, 行の中で文字は垂直に並ぶのだが, やはり, 文字を並べて行のボックスを作るモードを h モードと言い, できたボックスを h ボックスと呼ぶことにした. 横組みと縦組みの両方で使えるマクロを作りやすいように, ‘横組みの行のボックス’ と ‘縦組みの行のボックス’ は同じ名前の方が良く, ‘横組みの行のボックス’ は従来の h ボックスと同じものだから, 互換性を考慮してそのようにした.

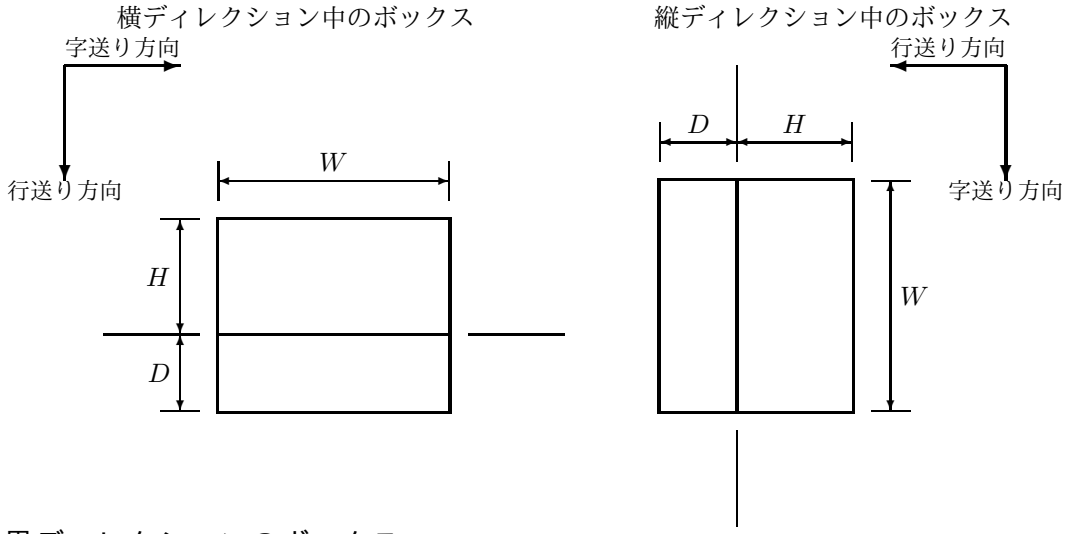
同様に, ボックスを行送り方向に並べるモードを v モード, それらを集めて作られたボックスを v ボックスと呼ぶことにした. 横ディレクションの v モードではボックスは縦（vertical）に並ぶが, 縦ディレクションの v モードではボックスが水平に右から左へ並ぶことになる.

TEX を起動した直後は横ディレクションの v モードであり, ディレクションを変えなければ従来の TEX と互換性がある. つまり, ここで文字を入力すると横ディレクションの h モードに移行し, 文字を左から右に並べていく. ラインブレークによって作られた h ボックスは, 横ディレクションの v モードで上から下に並べていく.

TEX のディレクションを変更するプリミティブとして, `\yoko`, `\tate` を用意した. ディレクションの変更は, 作成中のリストやボックスに何も入力され

ていない状態でのみ許すことにした. TEX を起動して何も入力されていない状態で\tate プリミティブを実行すると, 縦ディレクションの v モードになる. ここで文字を入力すると縦ディレクションの h モードに移行し文字を上から下に並べていく. ラインブレークによって作られた h ボックスは, 縦ディレクションの v モードで右から左に並べていく.

ボックスの大きさ（ベースラインの位置）を表す用語として, TEX では Width, Depth, Height という言葉を使っている. pTEX では, これらの言葉を, ボックスの字送り方向の大きさ（Width）, 行送り方向側の大きさ（Depth）, 行送り方向の反対側の大きさ（Height）であると定義した. よって, 縦ディレクション中のボックスでは, ボックスの高さを W といい, 左側の幅を D といい, 右側の幅を H という.



3.1 異ディレクションのボックス

一つの文書, 一枚のページであっても部分によって組み方向を変えられるように, 組み方向の混在を許すことにした. たとえば, 表の中で項目によって縦組み/横組みを使い分けることが可能になれば表現力が増すことになる. また, 縦組み中で数桁の数を表現するとき, アラビア数字を横組みにして縦組み中に挿入することがある. これを出版業界では連数字と言うが, これも組み方向の混在機能で実現できる.

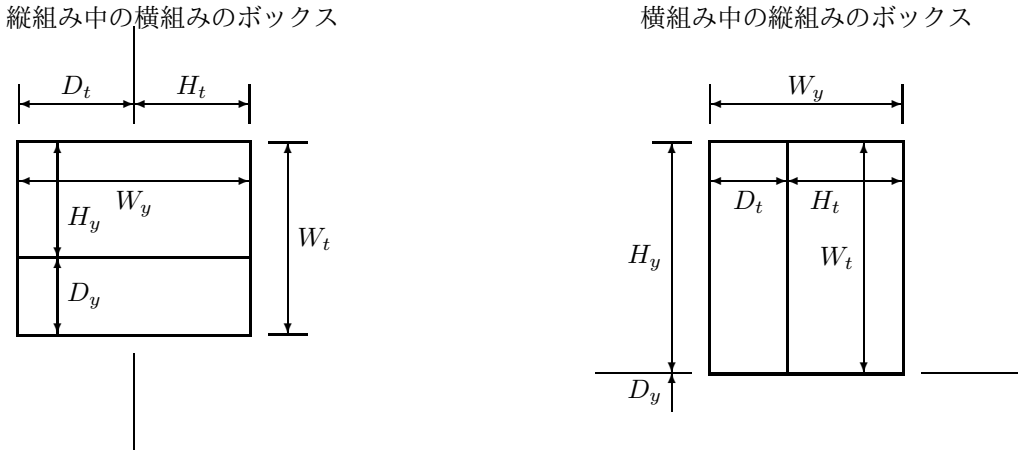
組み方向の混在を許すということは, 一つの文書に, 内部の文字が横に並ぶ横組みの h ボックスと, 縦に並ぶ縦組みの h ボックスの, 2 種類の h ボックスが存在することになる. 縦/横どちらのディレクションで作られたボックスであるか, ボックスごとに記録して, これらの組み方向の異なるボックスを区別することにした.

\hbox, \vbox などで作られるボックスは, 通常, \hbox, \vbox が実行されたときの TEX のディレクションと同じディレクションを持つ. しかし以下の

ように\yoko プリミティブを使うと, 縦ディレクションの中に, '123' という内容の横ディレクションを持つ h ボックスを置くこともできる.

```
（縦ディレクション）
\hbox{\yoko 123}
```

\hbox, \vbox でディレクションの異なるボックスを作ると, 横ディレクションを持つボックスは, 縦組み中では $W_t = H_y + D_y$, $H_t = W_y/2$, $D_t = W_y/2$（H_y は, ボックスの横ディレクション中における H）となる. 縦ディレクションを持つボックスは, 横組み中では $W_y = H_t + D_t$, $H_y = W_t$, $D_y = 0$ となる.



二つの場合で W, D, H の算出方法が異なるのは, 縦組み, 横組みでの文字の扱いの違い（4.1 参照）を反映している.

TEX には 256 個のボックスレジスタが用意されていて

```
\setbox123\hbox{\yoko ⟨ 文 ⟩}
```

と書けば, ボックスレジスタの 123 番に ⟨ 文 ⟩ を内容とする横ディレクションの h ボックスが登録される. 登録されたボックスは

```
\advance\ht123 by \dp123
\dp123=0
```

のように, \wd, \dp, \ht を使ってボックスの大きさやベースラインの位置を自由に変えることができる. pTEX では, レジスタの内容は

```
\hbox{\tate\copy123}
\hbox{\yoko\box123}
```

のようにどのディレクションで使用されるかわからず, ボックスが使用されるディレクションによって, W, D, H は異なる値を持つ. そこで\wd なども, そのときのディレクション用の W などをアクセスすることにした.

```
\hbox{\yoko \global\wd123=⟨縦高さ⟩}
\hbox{\tate \global\wd123=⟨横幅⟩}
\hbox{\tate\copy123}
\hbox{\yoko\box123}
```

のように書けば, 3行目の\copy123のWは⟨縦高さ⟩であり, 4行目の\box123のWは⟨横幅⟩である.

この場合注意しなければならないのは, \hbox, \vboxで作られたボックスは$W_t = H_y + D_y$, $W_y = H_t + D_t$であるが, \wd, \dp, \htを使ってW, D, Hを変更するとこの関係は必ずしも成立しないということである. ディレクションによってボックスの大きさが変化することになる.

3.2 異ディレクションのアジャスト, インサート

vモードや内部vモードの中のhボックスの中に\vadjustを使うと, vリストのそのボックスの次に, \vadjustの内容が挿入される. 以下のようにディレクションの異なるhボックスの中でも支障なくvadjustが使えるようにした.

```
（縦ディレクション, vモード）
\hbox{\yoko ...\vadjust{...}...}
```

\vadjustの中括弧の内側は, 自動的に, それがアジャストされるリストのディレクションである縦ディレクションになる.

\insertの場合, クラスによってディレクションを変えたい場合もある. たとえば文書全体は縦組みで, 脚注は横組みするとしよう. 図をクラス12のインサートで, 脚注をクラス34のインサートで処理することにする.

```
（縦ディレクション, vモード）
...\insert12{\tate ...}...
...\insert34{\yoko ...}...
...\insert12{\tate ...}...
...\insert34{\yoko ...}...
```

このように書けば, ボックスレジスタ12には縦ディレクションで図が集められ, ボックスレジスタ34には横ディレクションで脚注が集められるようにした. なお, 同じクラスに違うディレクションを混ぜて使うことはできない.

4　縦組みに使用する文字

4.1　縦組み用日本語フォント

TEX は各フォントに一つずつ用意されている TFM（*TEX Font Metric*）ファイルを参照して組版を行う．TFM ファイルにはそのフォントに含まれる各文字の大きさなど組版に必要な情報が定義してある．

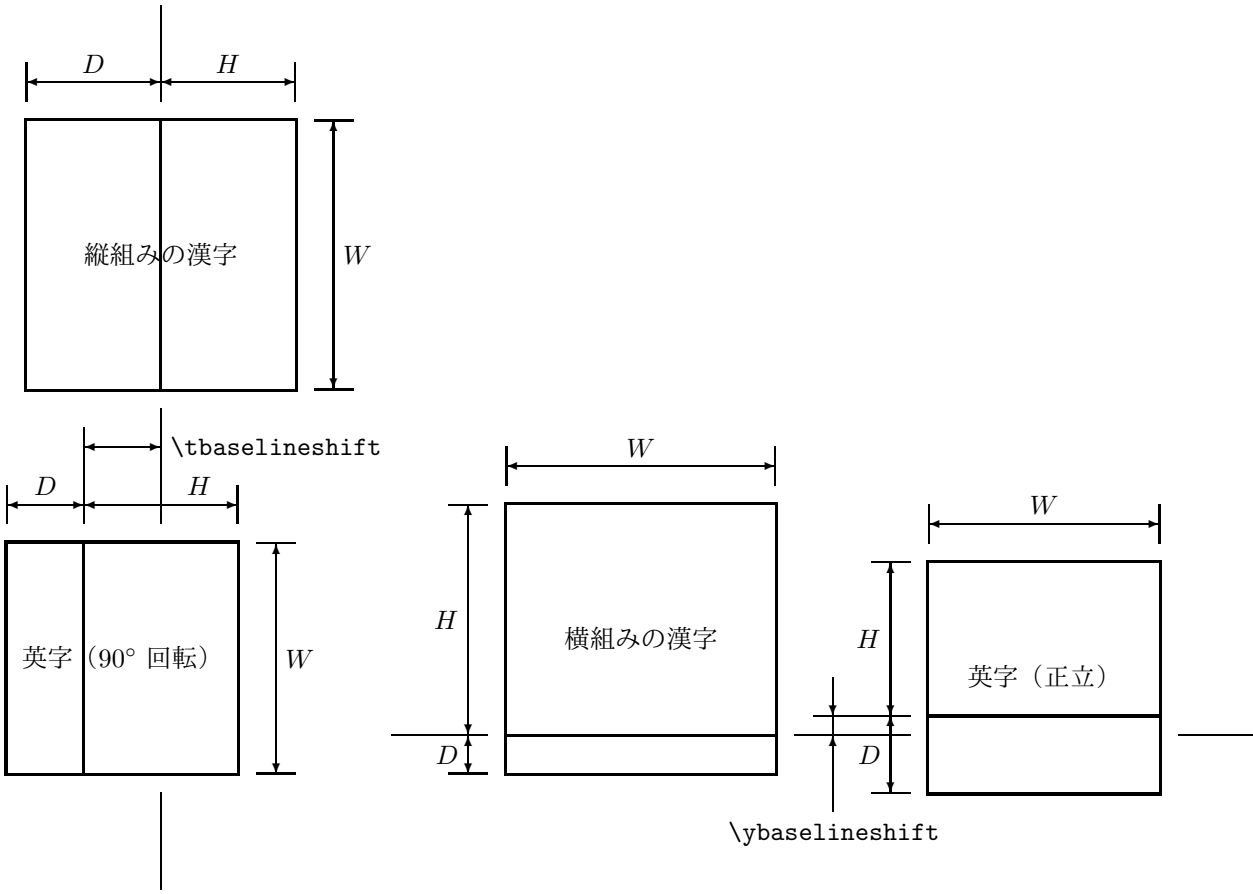
pTEX でも同様であるが, 日本語のフォントの場合, 同じ書体でも縦組みで使うか横組みで使うかによって, 組版に必要な情報は変わってくる．たとえば, “り” という文字は比較的縦に長い文字であるが, このような文字は横組みではとなりの文字との間隔をつめて組むことはできても, 縦組みではある程度間隔を空けないと読みにくくなってしまう．そこで, 横組み時と縦組み時では, 別の TFM ファイルを参照して組版することにした．TEX から見ると, 縦組みと横組みでは別のフォントを使っているように見えることになる．ただし, 組み方向によるフォントの切り替えは, ユーザーが気にする必要がないように自動的に行うようにした．和文用のカレントフォントを 2 つ用意し, ディレクションによって自動的に切り替えるようにしてある．また TFM ファイルに縦組み用か横組み用かの情報を埋め込み, フォントを指定したとき, 自動的に適切なカレントフォントを切り替えるようにした．

横組みでは, ベースラインは文字の下の方を通っている．縦組みの和文フォントは, ベースラインが文字の中央を通ることにした．行の途中で文字の大きさを変えたとき, そのほうが自然だと判断したからである．

4.2　縦組み中の欧文

縦組みの部分に欧文や数式が出てくると, 従来の TEX の欧文用フォントを 90° 回転して上から下に向かって組んでいく．これは普通に横に組んだ欧文を, 90 度回転させたような結果になる．ところが, 4.1 で説明したように, 縦組みの和文のベースラインは文字の中心を通るようにしたので, そこに欧文のベースラインを一致させると行がずれて見えてしまう．そこで, `\tbaselineshift` という dimen 変数を用意し, その寸法だけ縦組み中の欧文のベースラインをシフトさせることにした．

横組みでも, 和文と欧文のベースラインの位置を細かく調節しないと, フォントのデザインによってはバランスが悪くなる場合があり, `\ybaselineshift` を用意して調節可能にした．



5　縦組み中の数式

縦組み中の数式も, 欧文と同様にシフトさせなければならない. TEX の数式には, 分数の線の高さなどを決定するための axis（軸）という概念があるので, これでシフト量を決定することにした.

問題は次のような場合である.

（縦ディレクション, h モード）
...$ \hbox{$ 〈数式〉 $} $...

この場合, 数式モードがネスティングされることになるので, 内側の〈数式〉は 2 回シフトされることになってしまう. これを避けるため, $ の内側は, 普通の縦ディレクションではない縦数式ディレクションに移行することにした. 縦数式ディレクションでは文字やボックスの並ぶ方向は縦ディレクションと同じであるが, その他の動作は横組みと同様である. 数式ディレクション中で数式モードに移行してもシフトは行わず, また, 日本語のフォントは横組みカ

レントフォントを使用する. つまり, 数式は横組みで組んだものを時計回りに 90 度回転して, 縦組み中に挿入する形になる.

便宜的に, 横組みでも数式モードに入ると横数式ディレクションに入ることにした. 現在のディレクションが通常のディレクションであるか, 数式ディレクションであるかは, `\ifmdir` でテストできる.

6 組版結果の出力

TEX は`\shipout` プリミテイブで, ボックスの左上隅が紙の左上隅から, 右に `\hoffset`＋1 インチ, 下に `\voffset`＋1 インチ の位置になるように, `DVI`（*D*e*v*ice *I*ndependent）ファイルに出力する. pTEX も同様である. `\hoffset`, `\voffset` は, ほかのプリミティブ名と違って, ディレクションに関係なく, h, v が本当に holizontal, vertcal を意味している.

従来の`DVI` 命令は, 横組みのときファイルが小さくなるように工夫されている. たとえば, “文字の印字” と “文字幅だけ右へ移動” が, 1 つの命令でできるようになっている. pTEX では, 縦組みの場合でも横組みと同様にファイルが小さくなるように, `DVI` フォーマットを拡張した.

ただし, たとえば従来の文書をこの TEX で処理した場合など, その文書内で縦組みを使用していない場合, 従来のものと同じフォーマットの `DVI` を出力する. また, プリアンブル, ポストアンブルをテストすれば, 文書内した縦組みを使用した箇所が存在するかどうかを知ることができるようにした.

7 インプリメント

現在構築中のリストのネスティング状態を示すレコード（*list_state_record*）に, ディレクションを表すフィールドを追加し, トップのディレクションをマクロ *direction* でアクセスできるようにした. *direction* の値は *dir_yoko* か, *dir_tate* か, $-$*dir_yoko* か, $-$*dir_tate* である. $-$*dir_yoko*, $-$*dir_tate* は数式ディレクションを意味する.

各種ノードにディレクションを保持するフィールドを加えなければならない. ボックスを表す *hlist_node*, *vlist_node* では第 1 ワードの *sub_type* フィールドが未使用だったので, これを *box_dir* として使うことにした. `\insert` によって生成される *ins_node* には空いているフィールドがなかったので, サイズを 1 ワード大きくして全部で 6 ワードにし, 最後の 1 ワードを *ins_dir* として使うことにした.

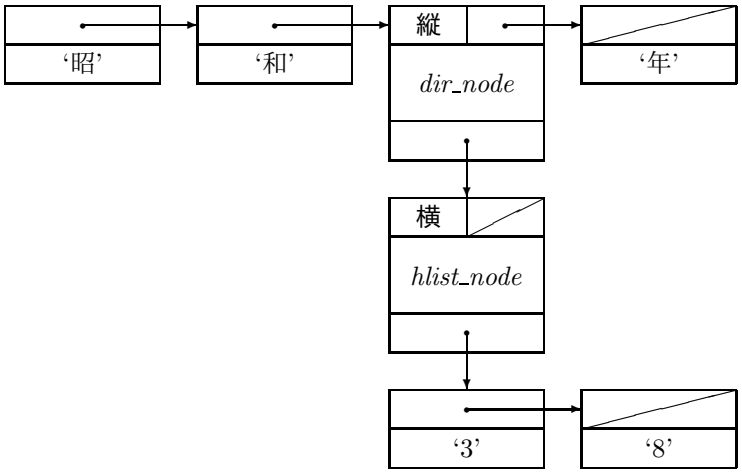
7.1 *dir_node*

縦ディレクションのリストに横ディレクションのボックスを追加するような場合, 間に *dir_node* を挿入して, ボックスの大きさなどを整合させる. *dir_node* は, *hlist_node*, *vlist_node* と全く同じ構造を持っている. リストにはこの *dir_node* が挿入され, *dir_node* の *box_dir* にはリストのディレクションが, *width*,*depth*,*height* には, リストのディレクションで見た, ボックスの W, D, H の値が入っている. *dir_node* の *list_ptr* は実体のボックスを指すポインタである. 実体のボックス（*hlist_node*, *vlist_node*）の *box_dir* にはボックスのディレクションが, *width*,*depth*,*height* には, ボックスのディレクションで見た, W, D, H の値が入っている.

たとえば

```
（縦ディレクション）
昭和\hbox{\yoko 38}年
```

は図のようなリストを作る.




```
\tmin 昭
\tmin 和
\dirboxT(5.00002+5.00002)x6.44444
.\hboxY(6.44444+0.0)x10.00003
..\tenrm 3
..\tenrm 8
\penalty 500(for \chrwidowpenalty)
\glue(\kanjiskip) 0.0 plus 0.4 minus 0.4
\tmin 年
```

‘\hbox{’ と入力されると, それまでのリストやモードやディレクションがセーブされ, 新たなリストやモードやディレクションが用意される. \tate

や\yoko は, その時点でのリストが空であればディレクションを変更する. '\hbox{' に対応する '}' が入力されると, まず, \hbox{}の内部のディレクションと同じディレクションのボックスが作られる. そのディレクションと\hbox{}の外側のディレクションが一致していなければ, *dir_node* を使って, ディレクションが整合させられる.

7.2 *disp_node*

和文中の欧文や数式のベースラインをシフトさせるために, 新種のノード *disp_node*（displacement node）を導入することにした. このノードは, リンク情報ワードのほかに *disp_dimen* フィールド（1 ワード）を持っている. *disp_node* は h リスト中にのみ存在し, このノード以降のすべてのノードのベースラインを, *disp_dimen* だけ行送り方向にシフトさせる.

たとえば, 縦ディレクションで\tbaselineshift の値が 2 ポイントのとき,

```
...この disp ノード...
```

のように入力すると 'disp' の前に 2 ポイントの *disp_node*, 後に 0 ポイントの *disp_node* が挿入される.

```
\tmin こ
\tmin の
\displace 2.0
\glue(\xkanjiskip) 2.5 plus 1.0 minus 1.0
\tenrm d
\tenrm i
\tenrm s
\tenrm p
\displace 0.0
\glue(\xkanjiskip) 2.5 plus 1.0 minus 1.0
\tmin ノ
\penalty 200(for kinsoku)
\glue(\kanjiskip) 0.0 plus 0.4 minus 0.4
\tmin ー
\penalty 500(for \chrwidowpenalty)
\glue(\kanjiskip) 0.0 plus 0.4 minus 0.4
\tmin ド
```

文字（*char_node*）, ボックス（*hlist_node*, *vlist_node*, *dir_node*）, 罫線（*rule_node*）以外のノードには, H や D がないので *disp_node* の影響を受けない. だから

```
...この dvi file フォーマット...
```

のように入力されたとき, device と independent の間の空白の前後に *disp_node* を入れるのは無駄である.

```
\tmin こ
\tmin の
\displace 2.0
\glue(\xkanjiskip) 2.5 plus 1.0 minus 1.0
\tenrm d
\tenrm v
\tenrm i
\displace 0.0
\glue 3.33333 plus 1.66666 minus 1.11111
\displace 2.0
\tenrm ^^L (ligature fi)
\tenrm l
\tenrm e
\displace 0.0
\glue(\xkanjiskip) 2.5 plus 1.0 minus 1.0
\tmin フ
\penalty 150(for kinsoku)
\glue 1.07391 minus 1.07391
\tmin ォ
```

そこで, リストに *disp_node* を加えるとき, 無駄な *disp_node* を消すようにした.

```
.\tmin こ
.\tmin の
.\displace 2.0
.\glue(\xkanjiskip) 2.5 plus 1.0 minus 1.0
.\tenrm d
.\tenrm v
.\tenrm i
.\glue 3.33333 plus 1.66666 minus 1.11111
.\tenrm ^^L (ligature fi)
.\tenrm l
.\tenrm e
.\displace 0.0
.\glue(\xkanjiskip) 2.5 plus 1.0 minus 1.0
.\tmin フ
.\penalty 150(for kinsoku)
```

```
.\glue 1.07391 minus 1.07391
.\tmin オ
```

7.3 縦組み用 TFM ファイル

縦組みフォント用拡張 TFM ファイルのフォーマットは, 基本的には, 従来の JFM フォーマット [4] と変わらない. ただ各フィールドの意味が多少変化している. 最初のハーフワード（*id*）で JFM フォーマットであることを示す. $id = 11$ の場合は横組み用 JFM, $id = 9$ の場合は縦組み用 JFM である.

width, *italic*, *kern*, *glue* など従来の横組み JFM フォーマットで横方向の意味を持っていたフィールドは, 縦組み JFM では縦（字送り）方向の意味を持つ. たとえば *width* はその文字の縦方向の大きさである.

height, *depth* など従来の横組み JFM で縦方向の意味を持っていたフィールドは, 縦組み JFM では横（行送り）方向の意味を持つ. *height* はその文字のベースラインの右側の大きさ, *depth* はその文字のベースラインの左側の大きさである.

7.4 拡張 DVI フォーマット

従来のDVI には水平方向へ移動する命令としてright, 垂直方向へ移動する命令としてdown という名前が使用されている. h ボックスを “水平ボックス” ではなく “字送り方向のボックス” と定義したようにright 命令を字送り方向へ移動する命令, down 命令を行送り方向へ移動する命令と定義した. 字送り, 行送りの方向は, 組み方向によって変わるので, DVI リーダにもディレクションを設けて, 新しい命令dir によってディレクションを切り換えることにした. push, pop で, (h, v, w, x, y, z) の他に, ディレクション d も push, pop する.

DVI の命令は最初の 1 バイトで識別できるようになっている. すでに 0–249 が使用されている[1] ので新しい命令は 255 を使うことにした. dir（255）命令は 1 バイトの引数 d を 1 つ取る.

- $d = 0 \cdots$ 横組み
- $d = 1 \cdots$ 縦組み

横ディレクションの場合は従来のDVI とコンパチブルであるが, 縦ディレクションの場合は従来のDVI で横方向に移動する命令

```
set_char_* set? set_rule right? w? x?
```

[1] TEX を, 英語などの左から右へ書く言語と, アラビア語やヘブライ語のように右から左へ書く言語を混植できるように改造した事例 [5] では, そのインプリメントの方法としてDVI 命令を拡張しており, 250, 251 を使用している.

は縦方向（字送り方向）に移動する命令となり, 従来のDVI で縦方向に移動する命令

```
down? y? z?
```

は横方向（行送り方向）に移動する命令となる.

文字は, ベースラインの方向が字送りの方向と一致するように, 必要なら回転して, 印字されなければならない. 罫線の命令もディレクションによって方向が変わる. 従来の`set_rule`, `put_rule` では, 最初の引数が高さ, 次の引数が幅を表していた. 拡張DVI では, 最初の引数が罫線の行送り方向の反対側の大きさ, 次の引数が字送り方向の大きさを表している. `set_rule` では 2 番目の引数で指定してある長さだけ字送り方向に参照点を移動させる.

各ページの先頭では, DVI リーダは横ディレクションであり, 拡張DVI 対応のプリンタドライバで, 従来のDVI もプリントアウトでき, 拡張命令（`dir`）を使っていないDVI ファイルは, 従来のプリンタドライバでプリントアウトできる.

DVI ファイルには, ファイルの先頭（preamble）と末尾（postamble）にDVI のバージョンを表す *id_byte* が書き込まれている. 従来のDVI ではどちらの *id_byte* も 2 である. 拡張命令を使っているDVI ファイルでは, postamble の *id_byte* を 3 にして, 拡張プリンタドライバでプリントアウトしなければならないことを表す.

8 日本語 TEX との互換性

pTEX は, 機能面では従来の日本語 TEX との完全な互換性を保っている.

性能面での違いを比較するために SONY NEWS 1460 上の日本語 TEX と pTEX で本論文を組版し, csh の time コマンドで CPU 時間を比較してみた. なおこのテストでは, あらかじめ aux ファイルを作成しておき, `virtex` に LATEX のフォーマットファイルを読み込ませる形で起動して計測した.

- 日本語 TEX の場合:
  `24.7u 0.3s 0:25 99% 44+123k 0+16io 0pf+0w`
- pTEX の場合:
  `25.0u 0.3s 0:25 99% 46+121k 0+16io 0pf+0w`

やはり多少負荷は重くなっているようだが, 約 1 %なのでほとんど変っていないと言っていいと思う. これは, pTEX を日本語 TEX の置き換えとして使用しても支障はないということを意味している.

9 おわりに

今回行った TEX の改造は, 相当大規模なものである. アスキー版日本語 TEX の段階で既に `trip test` をパスしなくなっており, これにさらに改造を加えたので, 'もはやこれは TEX ではない' という意見もあるかも知れない. 私は, TEX の根本は以下の 3 項目であろうと考えている.

- ボックスとグルー
- ラインブレーク, ページブレーク
- マクロ

この機構は単純であるにもかかわらず非常に強力で, 組版の諸問題をほぼ解決することができる. この基本を踏み外さなければ, TEX と呼んでいいのではないだろうか.

もともと, TEX システムはクヌース教授が伝統的な組版について調査し, 同等以上のクオリティを得ることを目標に作られたシステムである. pTEX も, 従来の日本語の組版について十分調査し, 先人の行ってきた美しい組版のための工夫を, できる限り取り込んでいきたいと考えている. それをある程度効率よく行うためには, やはり, TEX の内部に変更を加える必要がある.

今後の pTEX の環境の整備, 機能拡張として以下のものを考えている.

マクロの充実
: 縦組みに必要なマクロを洗い出し, 充実させる必要がある.

約物のチューニング
: 現在の `TFM` では, 約物（記号類）の扱いについてチューニングの足りない部分がある.

  `TFM` のチューニングで対応しきれない場合は, TEX 本体のスペーシングアルゴリズムに変更を加えなければならないかもしれない.

漢字・カナ別フォント
: 写植にはカナ書体と言って, 漢字が含まれていない書体がある. これと漢字の書体（明朝, ゴシックなど）とを組み合わせて文章を組む. カナの書体を変えただけでもかなり雰囲気が変わるものである.

  これを TEX で実現するには, カナのカレントフォントを設けて, コードによって漢字フォントとカナフォントを使い分けることになるだろう.

ルビ
: やはり日本語の書籍を組版するためには, ルビを避けるわけにはいかない.

ルビの振ってあることばの途中でラインブレークが起こる場合があるので, ラインブレーク処理に手を加えなければならない. 縦組み拡張に匹敵する大改造になると思われる.

謝辞

本研究の機会を与えてくださった（株）アスキー 電子編集研究統轄部 井芹昌信 統轄部長, 同 技術統轄部 三浦雅孝 統轄部長, ならびに, TEX システムについて貴重なご意見をいただいた 同 技術部 大野俊治 次長に感謝します.

参考文献

[1] Knuth, D. E. : *The TEXbook*, Addison-Wesley (1986)
［斎藤信男 監修, 鷺谷好輝 翻訳 : TEX ブック, アスキー (1989)］

[2] Knuth, D. E. : *TEX: the program*, Addison-Wesley (1986)

[3] 倉沢良一 : TEX システムの日本語化, 日本語 TEX 配布テープ, `./doc/jtex.tex` (1987)

[4] 倉沢良一 : JFM file format, 日本語 TEX 配布テープ, `./doc/jfm.tex` (1987)

[5] Knuth, D. E. and MacKay, P. : Mixing right-to-left texts with left-to-right texts, *TUGboat Volume 8 No. 1*, pp. 14–25 (1987)