Cahiers GUTenberg

MÉTRIQUE DES FONTES POSTSCRIPT

Jacques ANDRÉ, Justin BUR

Cahiers GUTenberg, nº 8 (1991), p. 29-50.

<http://cahiers.gutenberg.eu.org/fitem?id=CG_1991___8_29_0>

Métrique des fontes PostScript*

Jacques ANDRÉ[α†] et Justin BUR[β]

α Projet Opéra — INRIA/IRISA-Rennes,
Campus de Beaulieu, F-35042 Rennes Cedex, France. e-mail : jandre@irisa.fr
β EPFL - LSP/DI, INF Ecublens, CH-1015 Lausanne, Suisse.
e-mail : justin@lspsun1.epfl.ch

Résumé Cet article explique la métrique des fontes vue par PostScript. En particulier, après avoir présenté les notions de *bbox* et d'*AFM*, il montre comment faire, par PostScript, de la justification, du crénage, de l'interlettrage, etc.

Abstract *This note is concerned with PostScript font metrics. Bounding boxes and the Adobe Font Metric file are presented, as well as how to use them for justification, pair and track kerning, etc.*

Un article récent [André 89] a montré les propriétés métriques des caractères du temps du plomb. On retrouve, dans les langages de description de page, à peu près la même chose. Mais ces langages possèdent, en plus, des possibilités nouvelles en matière de typographie fine. Par ailleurs, plusieurs normes sont en cours de définition [Dardailler 89] [Marti 90] : nous espérons que cet article pourra aider à les comprendre.

Nous présentons ici les propriétés métriques des fontes[1] en PostScript. Les *Cahiers GUTenberg* continueront, par la suite, de faire le point sur ce sujet. Un prochain article devrait notamment décrire les métriques de TEX. Un autre devrait présenter *TrueType*.

*Cet article correspond à deux exposés réalisés lors des *Journées fontes* organisée par l'association GUTenberg à Paris le 4 décembre 1990, journée partiellement financée par le projet Didot (CEE - programme COMETT II numéro 90/3697/Cb).

†Une partie du présent travail a été réalisée lors d'invitations à l'université de Fribourg en 1988 et à l'École Polytechnique Fédérale de Lausanne en 1988 et 1989.

[1] [André 89] propose de distinguer la notion de « fonte », qui correspond exactement à la notion de *font* en PostScript, de celle de « police », qui correspond au résultat de ... `findfont [...] makefont setfont`.

Nous ne descendons pas ici au niveau de l'adaptation du tracé de caractères en fonction de la grille (bien que ce soit possible avec les *hints* des « fontes de type 1 » de PostScript), sujet qui devrait aussi faire l'objet d'un article séparé. Nous ne nous intéressons, par ailleurs, qu'aux caractères définis « par leurs contours » et non aux caractères définis uniquement par *bitmaps*.

Nous supposons, ici, que le lecteur a lu [André 89] et qu'il a une connaissance préalable de PostScript, voire même quelques notions sur les fontes en PostScript.

1. Généralités sur les fontes en PostScript

PostScript est devenu, en quelques années, une norme *de facto* en matière de langage de description de pages. Même si l'on entend tous les jours que PostScript va être déboulonné de son piédestal, on a tout lieu de croire qu'il va encore rester en usage quelques temps. PostScript n'a pu se répandre ainsi que pour les raisons suivantes :

1. il correspondait à un besoin,
2. il est défini très proprement (voir [Borghi 89]),
3. il a joué la qualité typographique de deux façons :

(a) en fournissant des polices de caractères d'une qualité alors inconnue pour les imprimantes (les créateurs de caractères ont la possibilité d'accéder aux mécanismes de *hinting*) ;

(b) en fournissant aux applications — la vocation première de PostScript est d'être le langage de sortie de formateurs comme Word 4 ou TEX[2] — des fonctionnalités permettant de « simuler » les fonctions typographiques classiques telles que le crénage, les ligatures, la justification, l'interlettrage, etc.

Cet article est essentiellement basé sur les ouvrages « officiels », les livres rouge [Adobe 87r], bleu [Adobe 89b], vert [Adobe 88v] et noir [Adobe 90n], et sur l'excellent ouvrage de Roth [88]. On trouvera aussi dans *The PostScript Language Journal* divers articles sur le sujet [Wood 89, Kochan 89, Parker 89]. En français, peu de livres sont, à notre connaissance, de quelque intérêt, sauf [Eminet 87]. Enfin, nombre de points ont été éclaircis en utilisant le logiciel *LaserTalk* qui, sur Macintosh, permet d'analyser de façon interactive le contenu des piles et dictionnaires PostScript.

1.1. La machinerie des fontes en PostScript

Avant d'entrer plus en détail sur ce mécanisme, donnons rapidement un exemple de fonctionnement de la machinerie des fontes en PostScript. Soit le programme suivant :

```
/Times-Roman findfont
8 scalefont
setfont
100 100 moveto
(a) show (b) show (a) show
showpage
```

Ce programme s'exécute en plusieurs étapes :

1. **`/Times-Roman findfont`** : Le dictionnaire décrivant la fonte « Times romain » est rendue accessible.

2. **`8 scalefont`** : à partir de la fonte, la police correspondante pour le corps 8 est construite.

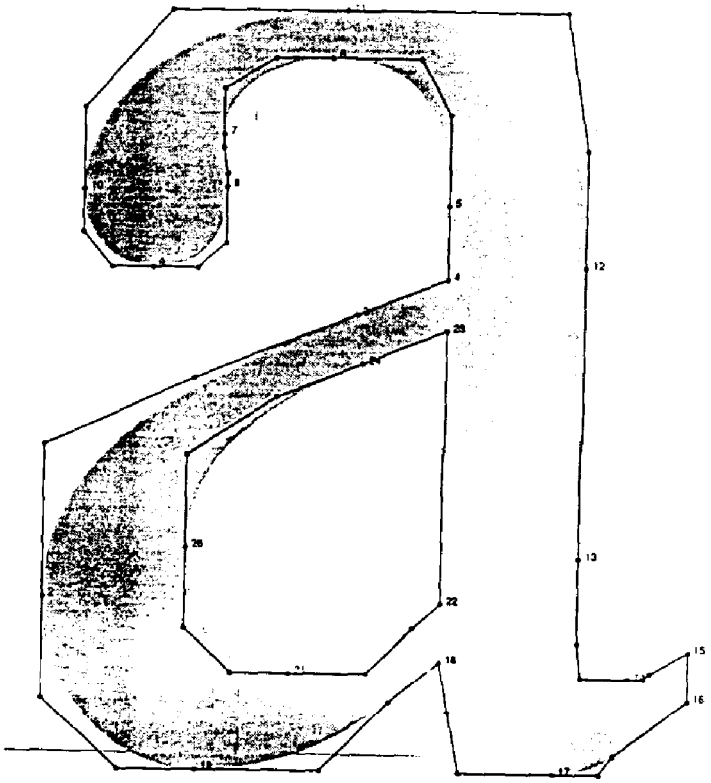
3. **`setfont`** : cette police est installée comme « police courante » *(currentfont)*. Tout appel à des fonctions utilisant des caractères (comme **`show`**, **`stringwidth`**, etc.) utilisera cette police. Voir note 1.

4. **`100 100 moveto`** : le point de coordonnées 100 100 (en points PostScript[3]) devient le point courant.

5. **`(a) show`** : **`show`** trouve dans le dictionnaire de la police les instructions de la figure 1-b : elles décrivent, en termes de morceaux de courbes de Bézier, le dessin du caractère "a" « abstrait » (comme tracé sur la figure 1-a où l'on a aussi indiqué les points de contrôle des diverses courbes de Bézier définissant ce caractère).
   Un algorithme, dit de *scan-conversion*, est alors appelé qui, compte tenu de la description du "a", de la définition de l'imprimante (par exemple 300

[2] Par exemple au travers de **`dvitops`**, **`dvi2ps`**, **`dvips`**, etc.

[3] Le point PostScript vaut 1/72 de pouce (en TEX, 1 bp) et est donc légèrement différent du point didot européen (1 dp) et du point pica anglo-américain (1 pt). Voir [André 89] section 2. Rappelons aussi que « point par pouce » est la traduction (non conforme au système métrique) de *dot per inch* mais que le mot point y signifie pixel et n'a rien à voir avec le point typographique.



```
% Char 97
/a {userdict begin 153 -12 moveto
95 -12 37 39 37 115 curveto
37 227 146 277 264 325 curveto
330 352 lineto
330 407 lineto
330 474 308 516 243 516 curveto
202 516 165 493 165 459 curveto
165 449 168 430 168 420 curveto
168 378 147 360 114 360 curveto
83 360 61 385 61 417 curveto
61 478 126 551 253 551 curveto
415 551 431 449 431 362 curveto
431 147 lineto
431 85 434 59 468 59 curveto
479 59 484 64 512 80 curveto
512 43 lineto
512 43 512 43 512 43 curveto
458 2 448 -12 415 -12 curveto
346 -12 337 32 330 69 curveto
294 39 244 -12 153 -12 curveto
closepath
143 153 moveto
143 94 178 60 220 60 curveto
277 60 310 95 330 113 curveto
330 313 lineto
269 288 lineto
207 263 143 221 143 153 curveto
closepath fill
end} def
```

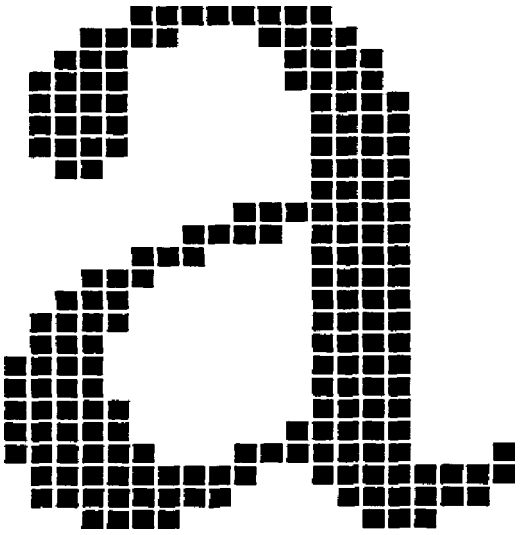


Figure 1: Impression d'un caractère en PostScript : a) dessin théorique, b) programme PostScript correspondant, c) résultat de **(a) show** compte tenu du corps demandé et de la définition de l'imprimante (agrandi ici environ 30 fois).

points au pouce) et compte tenu du corps (ici donc un corps 8), va calculer l'image *bitmap* correspondant à ce caractère (figure 1-c).
Cette *bitmap* est alors placée dans la page aux coordonnées du point courant.
Enfin le point courant est incrémenté de la chasse du caractère "a" en corps 8.

6. `(b) show` : la même chose que pour "a" se passe, mais avec une série d'instructions traçant un "b" ; on incrémente le point courant de la valeur de la chasse du "b" en corps 8.

7. `(a) show` : en fait, **show** conserve dans une mémoire cache l'image *bitmap* de chacun des caractères tracés. Le précédent appel `(a) show` a déja calculé l'image *bitmap* de "a". Plutôt que de refaire le calcul complet, **show** transfère simplement la *bitmap* du cache dans la page, puis incrémente les coordonnées du point courant de la chasse de "a".

8. **showpage** : la page est imprimée.

Nous préciserons par la suite certains détails de cette machinerie. Nous voulions d'abord montrer le principe de l'impression de caractères.

1.2. Fontes de type 1 et de type 3

PostScript connaît deux types de fontes : celles dites de type 3 qui sont créées par les utilisateurs eux-mêmes et celles de type 1 dont les secrets de fabrication, longtemps gardés jalousement par Adobe, ont été récemment rendus publiques. Quant aux fontes de type 2, apparemment prévues, elles n'existent pas ou plus.

Les fontes de type 3 nécessitent le respect d'une certaine architecture de fonte (dictionnaire minimal, noms de procédures, etc.), mais laissent à l'utilisateur une très grande liberté d'organisation de ses propres données. Le « livre bleu » [Adobe 87b] donne un schéma classique d'implémentation de fontes mais on trouve, par exemple dans [André 90], des fontes qui ne le suivent pas du tout.

Les fontes de type 1 sont définies par un sous-ensemble très restreint du langage PostScript. Mais ces fontes ont accès à des algorithmes de tracé spéciaux de l'interpréteur PostScript pour dessiner des caractères acceptables même pour les très petites tailles sur des imprimantes à basse définition. Des instructions spéciales *(hints)* identifient les parties des caractères devant recevoir un traitement particulier. Par ailleurs, les fontes de type 1 sont plus compactes et s'exécutent plus rapidement que celles de type 3. Enfin, elles sont « cryptées » pour protéger leur contenu. Il faut alors les transformer en texte PostScript normal avant de les examiner[4].

1.3. Où est la métrique de PostScript?

PostScript est un langage cible pour des applications : à part quelques cas particuliers d'utilisation de PostScript « à la main » (comme la création des figures de ce texte), ce sont plutôt des produits (comme Word 4, TEX, Illustrator, Grif, etc.) qui génèrent du PostScript. Ceci a plusieurs conséquences.

1. Un système de traitement de texte tel que Microsoft Word 4 devant, de toutes façons, faire tous les calculs de justification lors de l'affichage du texte sur écran, il serait redondant qu'un

[4] On trouve désormais sur le marché des produits comme *Font detective* dont c'est le but.

interpréteur[5] PostScript les refasse. En revanche, il faut que Word 4 connaisse la chasse des caractères pour faire ces calculs ; de même doit-il disposer des tables de crénage.

2. Puisque la métrique des caractères est supposée connue de l'application, l'intégrité de cette métrique n'a pas de raison d'encombrer l'interpréteur. Le dictionnaire d'une fonte de type 1 ne contient, par exemple, ni les tables de crénage ni les listes de ligatures possibles.

Les informations métriques (qu'on appelle *la métrique*) d'une fonte vont donc se trouver en divers endroits selon leur nature.

- Lors de l'exécution d'un programme PostScript, l'interpréteur doit connaître tout ce qui est lié au positionnement des caractères (chasse) et à la taille de l'image *bitmap* du caractère : ces informations sont stockées, en même temps que les algorithmes de tracé des caractères, dans le `FontDictionary`. Mais certaines données des fontes de type 1 ont un statut privé et ne sont pas directement accessibles à l'utilisateur ; toutefois, elles sont souvent calculables. Par exemple, on ne trouve pas la chasse d'un caractère d'une fonte de type 1 dans le dictionnaire mais par l'appel à la fonction `stringwidth`. On y reviendra en section 7.

- En plus du dictionnaire chargé dans l'interpréteur PostScript (donc dans l'imprimante ou la photocomposeuse), une fonte PostScript est aussi accompagnée d'un fichier, appelé AFM (*Adobe Font Metric file*), disponible pour les applications, fichier qui va donc contenir toutes les informations indispensables pour justifier un texte, faire du crénage ou des ligatures, etc. La section 5 va en donner le format. Dans cette première partie nous allons voir ce qui y est mis.

Des informations métriques tirées du fichier AFM se trouvent parfois sous une forme particulière à un système ou à une application. Par exemple, la ressource FOND d'une fonte pour écran Macintosh remplace le fichier AFM pour les applications sur Macintosh. De même, TEX peut utiliser les fontes PostScript si leurs AFM sont convertis en fichiers TFM.

2. Éléments de la métrique des caractères

L'opération d'écriture va montrer quels sont les éléments utiles pour la métrique des caractères pris individuellement. Ces valeurs doivent être connues de l'interpéteur PostScript et de l'application en amont.

2.1. Opérateur `show`

À quelques détails près que nous verrons plus loin, comme **kshow** (section 3.1) ou **ashow** (section 6), PostScript n'offre qu'une seule fonction de tracé de caractères : **show**. Ayant choisi une police[6], par exemple par les opérations

```
/Times-Roman findfont
[100 30 0 100 0 0] makefont setfont
```

et en supposant que le `currentpoint` est fixé au point de coordonnées (x, y),

[5] On parle d'interpréteur PostScript tout en sachant que l'on commence à trouver sur le marché des compilateurs ou en tout cas des interpréteurs adaptés aux applications interactives comme *Display PostScript*.

[6] Voir la note 1 pour le sens donné à ce mot.

l'instruction `(a) show` trace le caractère "a" et repositionne le `currentpoint` en (x', y') (voir figure 2) avec :

$$\begin{cases} x' = x + w_x \\ y' = y + w_y \end{cases}$$

Ce couple de valeurs (w_x, w_y), où w est l'abréviation de *width* (chasse), suffit complètement pour composer un texte caractère par caractère[7], pas à pas.

Figure 2: Action de l'opérateur `show`.

Une fonte PostScript est donc un dictionnaire de programmes PostScript pour dessiner chaque caractère. Dans le cas le plus fréquent, où l'on utilise la mémoire cache pour y stocker les images *bitmap*, un interpréteur PostScript a besoin, pour optimiser son espace, qu'une fonte (au sens de programme PostScript) lui précise l'espace minimum que prendra, dans cette mémoire, le caractère considéré. Cet espace est appellé *bbox (bounding box)* du caractère. C'est le plus petit rectangle ayant ses côtés verticaux ou horizontaux englobant ce caractère (figure 3 bas). Il est caractérisé par les coordonnées des coins inférieur gauche *(lower left = ll)* et supérieur droit *(upper right = ur)*. Ces coordonnées sont définies par rapport au point $(0,0)$ dans l'espace du caractère. L'opération `show`, décrite plus haut, revient alors à recopier la *bbox* du caractère dans l'espace de la page, le point $(0,0)$ étant placé en (x, y).

[7] `(abc) show` est équivalent à `(a) show (b) show (c) show`.

La *bbox* est alors définie dans la page réelle par les points suivants (voir figure 3 bas) :

$$\begin{cases} x + ll_x \\ y + ll_y \end{cases} \quad \begin{cases} x + ur_x \\ y + ur_y \end{cases}$$

Ce sont finalement les 6 valeurs

$$w_x, w_y, ll_x, ll_y, ur_x, ur_y$$

qui définissent complètement le positionnement d'un caractère par rapport au point courant.

Implémentation

- *Les six valeurs*

  $$w_x, w_y, ll_x, ll_y, ur_x, ur_y$$

  se trouvent dans l'AFM dans la rubrique `StartCharMetrics` *avec les codes* `WX` *et* `WY` *d'une part, et* B(ounding box) *d'autre part.*

- *Dans le cas d'une fonte de type 3, ces valeurs sont passées à l'interpréteur comme paramètres de l'opération* `setcachedevice`. *Le schéma classique d'implémentation d'une fonte de type 3 est d'avoir la métrique dans la description de chaque caractère. Si on veut modifier ces valeurs par défaut, on utilise souvent les dictionnaires* `Metrics` *et* `BBox`. *En cas de création de fonte par* `setcharwidth`, *c'est-à-dire lorsque l'on n'utilise pas la mémoire cache, seules les valeurs de chasse sont utiles.*

- *Il n'y a pas de différence conceptuelle entre les fontes de type 3 et celles de type 1. Mais, dans le cas d'une fonte de type 1, la* `bbox` *est calculée automatiquement au moment d'interpréter la procédure du caractère. La valeur w_x est fourni en paramètre à l'opération* `hsbw` (horizontal sidebearing and width), *qui doit paraître au*

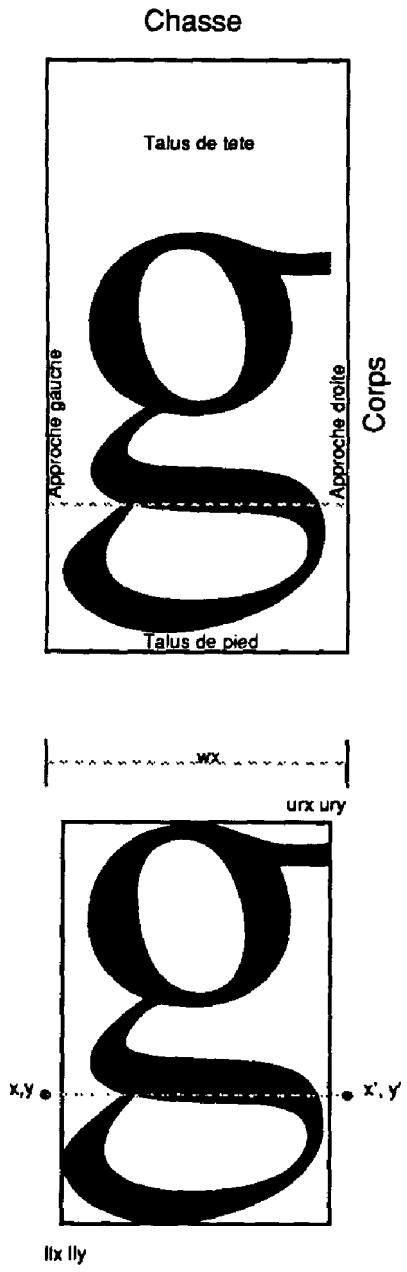


Figure 3: Métrique des caractères en plomb (en haut) et en PostScript (ici $w_y = 0$). Du temps du plomb, le caractère incluait talus et approches. En PostScript, seule la *bbox* (boîte englobant l'œil) compte. La figure 13 montre comment cette figure-ci a été tracée.

début de la procédure si $w_y = 0$; sinon, les deux sont donnés en paramètre à `sbw` (sidebearing and width).

Ceci n'est visible que pour une fonte décryptée.

Pour des raisons d'optimisation de l'espace de la mémoire cache, la machinerie des fontes demande la `FontBBox`, c'est-à-dire les coordonnées des coins inférieur gauche et supérieur droit de la boîte minimale qui recouvre tous les caractères.

Implémentation

- *La* `FontBBox` *est dans l'AFM.*
- *La* `FontBBox` *doit être donnée dans tout dictionnaire de fonte, qu'elle soit de type 1 ou de type 3.*

2.2. Chasses et approches

Tous les alphabets n'ont pas le même sens d'écriture[8]. Par exemple,

- l'hébreu et l'arabe qui se lisent de droite à gauche ont
$$w_x < 0$$
$$w_y = 0$$
- le chinois, qui se lit de haut en bas, a
$$w_x = 0$$
$$w_y < 0$$
- les langues européennes, qui se lisent de gauche à droite, ont
$$w_x > 0$$
$$w_y = 0$$

Nous nous plaçons désormais dans ce dernier cas.

Les valeurs a_g et a_d des approches gauche et droite *(sidebearings)* sont faciles à calculer à partir des informations métriques. Toujours dans le cas des langues européennes,

$$a_g = ll_x$$
$$a_d = w_x - ur_x$$

[8] Il existe même des alphabets dont le sens d'écriture dépend des lettres : *Scrabble*, par exemple, est défini de façon que les caractères s'alignent verticalement par `(ab) show` et horizontalement par `(AB) show` [André 90].

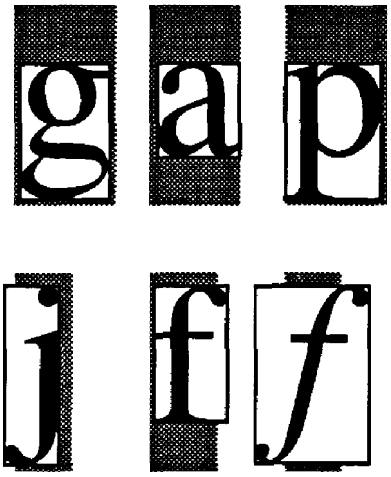


Figure 4: Divers cas d'approche gauche a_g et droite a_d.
pour "g" $a_g > 0$ et $a_d > 0$
pour "a" $a_g > 0$ et $a_d = 0$
pour "p" $a_g = 0$ et $a_d > 0$
pour "j" $a_g < 0$ et $a_d > 0$
pour "f" $a_g > 0$ et $a_d < 0$
pour "*f*" $a_g < 0$ et $a_d < 0$
le grisé part du point courant et a pour largeur la chasse du caractère.

La figure 4 montre que ces valeurs a_g et a_d peuvent être positives, négatives ou nulles.

Implémentation

- *Puisqu'elles peuvent être calculées, les approches ne font pas partie de la métrique.*

- *Cependant, l'approche gauche est donnée explicitement pour chaque caractère d'une fonte de type 1 (ll_x). Cette valeur, ainsi que la chasse du caractère, peut être changée par l'utilisateur en ajoutant un dictionnaire* `Metrics`. *La forme du caractère reste inchangée ; la* bbox *garde les mêmes dimensions mais peut subir une translation.*

2.3. `makefont`, `scalefont` et facteur d'échelle

PostScript permet de faire des transformations affines sur les caractères d'une fonte. Ces transformations sont définies par une matrice, dite *font matrix*, que l'on passe à `makefont`. Cette matrice est de la forme

$$[h\ \alpha\ \beta\ v\ d_x\ d_y]$$

avec :

h un facteur d'échelle horizontal,
$\alpha = h \times$ tg(angle avec l'horizontale)
$\beta = v \times$ tg(angle avec la verticale)
v un facteur d'échelle vertical,
d_x, d_y les valeurs d'une translation appliquée sur le caractère.

La figure 5 a été construite avec les diverses matrices suivantes pour les dessins de gauche à droite et de haut en bas :

```
[100     0          0        100   0  0]
[200     0          0        100   0  0]
[100     0          0        200   0  0]
[100 100tg(30)      0        100   0  0]
[100     0      -100tg(60)   100   0  0]
[100 100tg(30) -100tg(60)    100   0  0]
[100     0          0        100  20 30]
[100 100tg(30) -200tg(60)    200  20 30]
```

Le cas le plus fréquent est celui où l'on utilise le même facteur d'échelle horizontalement et verticalement et où on ne fait aucune rotation ni déplacement. Une écriture simplifiée de

```
[ f 0 0 f 0 0 ] makefont
```

est `f scalefont` où le facteur d'échelle `f` correspond au corps du caractère : en fait PostScript ne connaît pas la notion de *corps* (voir 2.4) mais simplement des facteurs d'échelle ; d'ailleurs le terme employé pour choisir un *corps* est `scalefont` et non, par exemple, `pointsize`. Ces facteurs

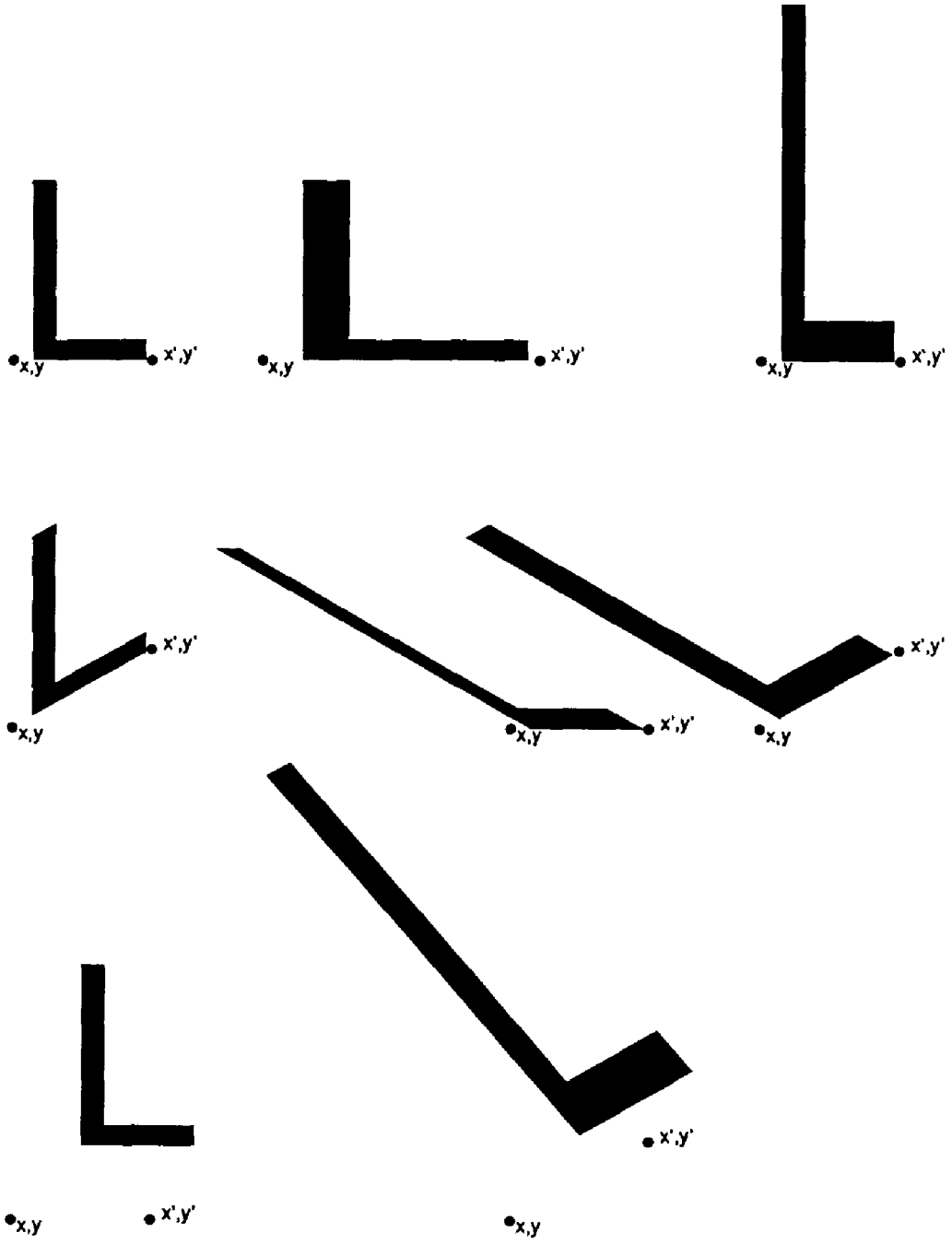


Figure 5: Transformations affines sur une fonte par le biais de **makefont** – voir section 2.3.

d'échelle dépendent du dessin original. En général, les caractères sont dessinés à la taille du corps 1000 (c'est-à-dire que les divers `curveto` ou `lineto` ont des paramètres qui définissent le caractère pour un corps 1000).

Pour que le fait d'écrire `50 scalefont` corresponde à un corps 50, il faut ramener le dessin théorique du caractère de 1000 à 1, puis à 50, ce qui revient à multiplier la CTM (*Current Transformation Matrix*) par la `Font Matrix`

```
[ 0.001 0 0 0.001 0 0 ]
```

Implémentation

- *La* `FontMatrix` *doit toujours être présente dans un dictionnaire de fonte. Ce rapport de 1 à 1000 est une convention respectée par la grande majorité des fontes, notamment celles de type 1. Mais d'autres valeurs sont acceptables.*

2.4. Corps, talus et interlignage

Pour PostScript, seule compte donc la *bbox*, approches et talus exclus. La figure 6 compare les métriques verticales traditionnelles et de PostScript. La notion de corps correspond, en fait, à la notion d'interlignage. On vérifie, par exemple, que la distance entre les pieds des lettres de la figure 6 est égale au corps des caractères (c'est-à-dire à la hauteur des boîtes des caractères de gauche). Cette distance peut toutefois poser des problèmes pour certains caractères accentués comme Å et la différence entre la taille du point pica et du point didot est alors significative (voir par exemple [Romberger & Sunblad 84]).

Si l'on utilise, par exemple, un corps 12, l'interlignage minimal idéal entre chaque ligne est de 12 points. Pour écrire plusieurs lignes, une application utilisera donc, par exemple, les instructions suivantes :

```
... findfont 12 scalefont setfont
100 600 moveto (ligne 1) show
100 588 moveto (ligne 2) show %600-12
100 576 moveto (ligne 3) show %588-12
...
```

Mais les coordonnées exactes des talus sont complètement inconnues en PostScript[9] où ces talus n'ont pas d'existence réelle !

Implémentation

- *Les notions de corps, talus et interlignage n'ont pas de sens pour PostScript. On ne trouve pas ces valeurs dans la métrique.*

2.5. Encodage

Il n'y a pas de correspondance exacte entre les codes d'un jeu de caractères comme l'ASCII ou l'ISO Latin-1, d'une part, et les caractères typographiques[10] d'une fonte, d'autre part. Plusieurs raisons à celà : parfois, un caractère essentiel en typographie ne se trouve pas dans l'ASCII (comme par exemple les tirets) ; dans d'autres cas, un caractère ASCII peut être ambigu (comme le trait d'union qui représente également le signe moins) ; parfois il s'agit de ligatures (comme fi, fl) ou d'autres cas où la meilleure représentation imprimée est différente de la représentation codée la plus logique.

Les signes d'une fonte PostScript sont alors identifiés non pas par un code arbitraire, mais plutôt par un nom descriptif : `a` pour la lettre « a minuscule », `ampersand` pour la « perluète » (&), `eacute` pour la lettre « e minuscule avec

[9] Toutefois, une bonne approximation est de prendre 3/4 du corps pour la partie au-dessus de la ligne de base et 1/4 pour celle en dessous. C'est ce que nous utilisons par exemple ici en figure 3 et en figure 6. De plus, elles varient beaucoup d'une fonte à l'autre, ou d'un fabricant à l'autre (voir [Tracy 86]).

[10] On dit aussi, maintenant, une « glyphe ».

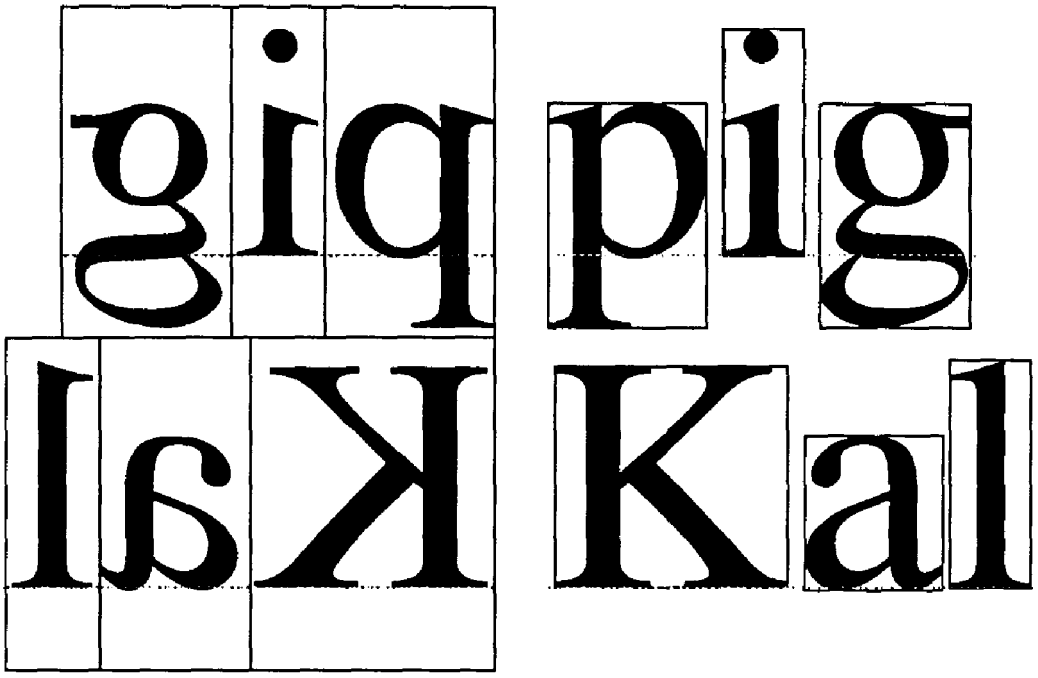


Figure 6: Caractères au plomb (à gauche) et en PostScript.

accent aigu », etc. Pour faciliter l'usage de la fonte, une affectation initiale de codes aux symboles est proposée : cette correspondance est un codage (*encoding*) qui se trouve dans le vecteur **Encoding** du dictionnaire de la fonte. Pour la plupart des fontes de type 1, un code de 149 caractères basé sur l'ASCII est utilisé. Il s'appelle *Adobe Standard Encoding*. Puisqu'une fonte contient typiquement plus de symboles, plusieurs n'ont pas de code initial et ne peuvent être utilisés à moins de modifier le vecteur **Encoding** (voir [Borghi 90]).

Pour générer une image sur la page, l'opérateur **show** passe par deux niveaux d'indirection. Pour chaque caractère dans la chaîne à imprimer, d'abord il trouve le nom du symbole correspondant dans le vecteur **Encoding** ; ensuite il utilise ce nom pour chercher la procédure qui génère la forme du symbole.

3. Métrique des caractères entre eux

Les approches sont calculées par le dessinateur de caractères pour que deux caractères placés côte à côte ne se chevauchent pas et pour que l'espace entre eux permette une bonne lisibilité du texte (y compris son rythme, la régularité optique des espacements, etc.) Il y a toutefois des exceptions quantifiables que l'on signale donc dans l'AFM.

Alors que la métrique liée aux caractères individuels est indispensable pour la machinerie de fonte, les propriétés à venir sont optionnelles.

3.1. Crénage

Le crénage de caractères (en anglais *pair kerning*) se fait en reculant le point courant ; par exemple, pour créner "A" et "V", on peut écrire :

```
(A) show
k 0 rmoveto  % k < 0
(V) show
```

Cette valeur `k` peut, bien sûr, être déterminée de façon approchée (prendre par exemple $-1/5$ de la chasse de V). Il est toutefois préférable de faire confiance au dessinateur du caractère qui fournit une valeur subjective pour les paires de lettres à créner.

Implémentation

- *L'AFM contiendra donc les paires de lettres à créner et la valeur recommandée de crénage (*`A V -12.9` *par exemple).*

PostScript dispose aussi d'outils pour faire du crénage par table et, bien que cette méthode ne soit pas prévue dans l'AFM aujourd'hui, signalons-la quand même rapidement.

Soit T une table de valeurs $T_{i,j}$ de modification d'approche (ou de crénage) entre le caractère de code i et celui de code j. Soit `K` une procédure PostScript qui exécute l'instruction

$$T_{s-1,s}\ \texttt{0 rmoveto}$$

où s et $s-1$ indiquent les deux valeurs au sommet de la pile, alors, si $c_0c_1\ldots c_n$ indique une chaîne formée des caractères $c_0, c_1, \ldots c_n$, l'appel

$$\texttt{K}\ c_0c_1\ldots c_n\ \texttt{kshow}$$

met en marche le processus suivant : c_0 `show` est appelé ; le caractère c_0 est donc mis dans la page au point courant et celui-ci est incrémenté de la chasse de c_0. Les codes de c_0 et de c_1 sont mis au sommet de la pile d'exécution et la procédure `K` est appelée. Elle recule donc le point courant de la valeur de crénage de la paire c_0, c_1. Et le processus reprend avec c_1, c_2, etc.

Cette méthode a l'inconvénient d'utiliser une matrice de crénage qui est en général creuse. Mais elle s'applique bien dans le cas où il y a beaucoup de paires de crénages (le cas limite étant celui où chaque paire de caractères a sa propre approche) ou dans celui où les caractères peuvent être regroupés en classes, ce qui donne une petite matrice. C'est ce qui est employé pour le Delorme [André & Delorme 90].

3.2. Modification de l'interlettrage

L'opération `ashow` permet de modifier l'espacement entre les caractères. Elle est utilisée en général pour augmenter l'interlettrage (voir ci-dessous, section 6, une application à la justification). Si l'on diminue l'interlettrage, c'est-à-dire si l'on serre les caractères, on ne peut le faire que légèrement pour les petits corps alors que pour les grands corps on peut serrer, proportionnellement, d'avantage.

PostScript propose, dans l'AFM, plusieurs valeurs caractérisant cet interlettrage réduit (en anglais *track kerning*). Pour chaque dégré de resserrement (caractérisé par un petit entier négatif) sont donnés les corps minimal et maximal pour lesquels la réduction d'interlettrage est variable et le nombre de points d'espace à retirer entre chaque paire de lettres aux corps minimal et maximal. Entre ces limites, la réduction de l'interlettrage est une fonction linéaire. En-dessous du minimum et au-dessus du maximum, on devrait utiliser la valeur minimale ou maximale.

La réduction de l'interlettrage est utile surtout pour les affiches et la publicité pour créer des effets spéciaux ; elle peut servir exceptionnellement pour faire rentrer un texte trop long dans un espace fixe. Toutefois, son usage est déconseillé pour la plupart des travaux de

composition de texte continu.

Implémentation

- *Les valeurs de* `TrackKern` *font partie de l'AFM.*

3.3. Caractères composites

Un caractère composite est formé en superposant deux caractères (ou plus) qui existent déjà dans la fonte. L'application la plus courante est liée aux lettres accentuées. C'est le créateur de caractères qui est le mieux placé pour décider du positionnement exact des composants d'un tel caractère.

Une partie du fichier AFM fournit les instructions pour créer des caractères composites : les noms des caractères de base à utiliser et leur position rélative. Une application peut se servir des instructions pour créer explicitement ces caractères – par exemple, dans une fonte virtuelle (`VF`) de TEX. Mais les mêmes instructions se trouvent aussi dans les fontes de type 1. Pour y accéder, il suffit de mettre les noms des caractères composites voulus dans le vecteur `Encoding`[11]. Voir figure 7.

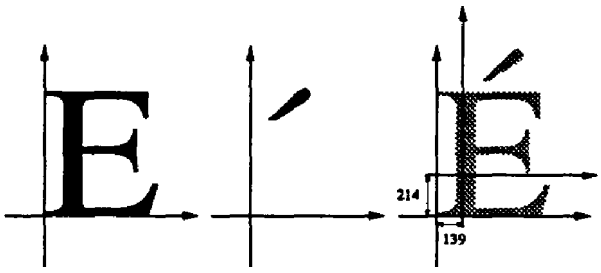


Figure 7: Création du caractère É à partir du E et de l'accent aigu après translation de ce dernier (voir figure 11).

[11] Les composants doivent aussi être présents dans le vecteur `Encoding`. Par exemple, pour utiliser le caractère `Eacute`, `E` et `acute` doivent être là.

Implémentation

- *Une partie du fichier AFM, délimitée par les lignes* `StartComposites` *et* `EndComposites`, *contient les instructions pour les caractères composites. Pour chaque caractère composite, la commande* `CC` *donne le nom du caractère et le nombre de parties ; chaque commande* `PCC` *donne le nom d'une partie et la position rélative de la partie.*
- *Dans une fonte de type 1, un caractère composite est crée par l'instruction* `seac` *(*standard encoding accented character*).*

3.4. Ligatures

Dans certaines fontes, il arrive qu'un dessin unique de caractère remplace ceux de deux caractères voisins (par exemple, figure 9, la paire « fi » est remplacée par le caractère unique « ﬁ »). C'est ce que l'on appelle une ligature[12].

Une application doit donc savoir quelles sont les ligatures existantes. Pour cela, il suffit de signaler que, par exemple, un « f » peut donner lieu à ligature avec un « i », un « l », etc.

La figure 9 a été produite en écrivant

```
(la \336gure) show
```

la ligature « ﬁ » se trouvant à la position \336 du codage *Standard Encoding.*

Implémentation

- *Les caractères pouvant être liés à un autre caractère sont donnés dans la métrique de ce dernier (champs* `L`*) dans l'AFM.*

[12] A la bibliographie donnée dans [André 89], il convient d'ajouter l'article suivant : Jérome Peignot, « Petit traité de la ligature », *Communication et langages*, 73, 1987, 20-36.

4. Métrique globale à une fonte

Tout ce que nous avons décrit concernait les caractères pris individuellement ou pris deux par deux. À présent, nous considérons les problèmes globaux à une fonte. Toutes ces informations sont purement informatives.

Les valeurs que nous allons voir à présent font partie de la métrique, mais aussi (c'est toujours vrai pour les fontes de type 1) du dictionnaire `FontInfo` que peut lire un programme PostScript.

4.1. Lignes de repère

Tous les caractères d'une fonte s'alignent par rapport à une ou plusieurs lignes de repère. En PostScript, on a, du haut vers le bas (voir figure 8) :

1. la ligne des ascendantes (en PostScript : *ascender*),
2. la ligne des capitales (*cap height*),
3. la ligne des bas de casse (*x-height*),
4. la ligne de base (*baseline*),
5. la ligne des descendantes (*descender*).

La connaissance de ces valeurs permet, par exemple, à l'application de faire de l'interlignage plus serré ou de tracer des boîtes autour de textes.

En fait, en typographie, on a besoin d'autres lignes (voir [André 89, fig. 7] et [Karow 87, p. 28]). Dans les fontes de type 1, d'autres lignes de repère peuvent être spécifiées pour définir les zones d'alignement (`BlueValues`, `OtherBlues`). Mais ces valeurs sont cachées dans le dictionnaire `Private`, donc ne sont pas accessibles en PostScript. Si on lit le texte decrypté de la fonte, on trouve les valeurs mais rien n'indique le nom de

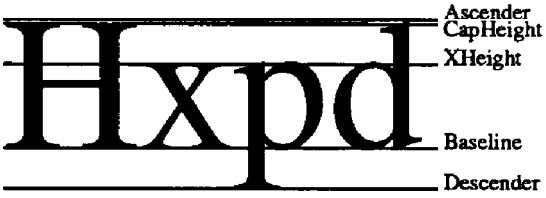


Figure 8: Principales lignes de repère.

chaque ligne : seule la ligne de base est déterminée (elle doit être la première ligne dans `BlueValues`).

Implémentation

- *Les 5 lignes de repère font partie de l'AFM.*

4.2. Soulignement

Comme pour la valeur des crénages, c'est le dessinateur de caractère qui est le mieux placé pour décider où souligner un mot. Les valeurs suffisantes sont la distance en dessous de la ligne de base et l'épaisseur du trait (voir figure 9).

Implémentation

- *L'AFM contient donc les deux valeurs* `UnderlinePosition` *et* `UnderlineThickness`.
- *De même dans* `FontInfo`.

4.3. Italique

L'angle que fait l'italique avec la verticale est également caractéristique d'une fonte. Cette valeur peut, par exemple, affiner le soulignement d'un mot (voir figure 9).

La correction d'italique de TEX n'est pas une fonction de base de PostScript, mais elle peut facilement se faire en augmentant l'interlettrage (voir [André 89] figure 18).

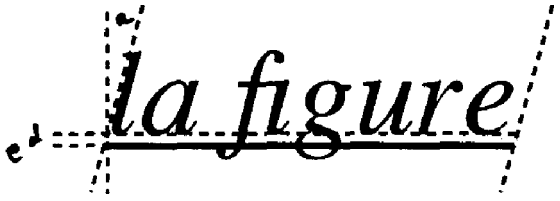


Figure 9: Texte italique souligné avec
d = `UnderlinePosition`,
e = `UnderlineThickness` et
a = `ItalicAngle`.

Implémentation

- `ItalicAngle` *est dans l'AFM,*
- *et dans* `FontInfo`.

4.4. Graisse

La graisse n'est pas un facteur que l'on sait, aujourd'hui, quantifier. On la caractérise par des mots comme « gras », « extra-gras », etc.

Implémentation

- *Une caractérisation de la graisse (*`Weight`*) par des mots standard (*light, bold, medium, *etc.) est dans l'AFM,*
- *et dans* `FontInfo`.

5. AFM *(Adobe Font Metric File)*

L'*Adobe Font Metric File* est donc un fichier donnant, en amont de PostScript, les dimensions significatives de chaque fonte traitée par un interpréteur PostScript.

Le contenu de ces fichiers et leur syntaxe sont décrits dans [Adobe 90f][13].

Les figures 10 et 11 donnent un extrait d'un tel fichier pour le Times romain[14].

On y trouve donc :

1. des informations générales sur la fonte (nom, copyright, version, etc.) ;
2. des informations métriques globales (soulignement, lignes de repère, `FontBBox`, etc.) ;
3. les informations métriques liées à chaque caractère, par exemple pour le "f" on trouve :
   - `C 102` qui donne le code interne du caractère selon l'`Adobe Standard Encoding` (voir 2.5),
   - `WX 333` qui donne la chasse w_x du caractère ($w_y = 0$),
   - `N f` pour le nom du caractère,
   - `B 21 0 382 685` qui donne les coordonnées de la *bbox* du "f" : $ll_x = 21, ll_y = 0, ur_x = 382$ et $ur_y = 685$,
   - `L i fi ; L l fl` qui indique que le "f" peut donner lieu à ligature avec un "i" (le caractère unique s'appelle, dans cette fonte, `fi`) ou un "l" (`fl`) ;
4. les informations globales sur la réduction de l'interlettrage (*track kerning*) ; ici il y a trois dégrés de resserrement, appelés `-1`, `-2`, et `-3` ; pour le dégré `-3`, le plus serré, on retire 0.1 point entre chaque paire de lettres jusqu'à six points ; entre 6 et 72 points l'espace à retirer varie linéairement de 0.1 à 3.78 points ;

[13] Ce document, ainsi que beaucoup d'autres d'Adobe, est accessible par courrier électronique : envoyer un message contenant seulement le mot *help* à `ps-file-server@adobe.com`.

[14] Dans ce cas il s'agit de *Times Roman*. La version Monotype s'appelle *Times New Roman* : le nom fait partie de la fonte. Voir [Bertrand 91] pour les problèmes de copyright des fontes.

```
% ------------------------------------ INFORMATIONS GENERALES SUR LA FONTE
StartFontMetrics 2.0
Comment Copyright (c) 1984 Adobe Systems Incorporated.
Comment All Rights Reserved.
FontName Times-Roman
FullName Times Roman
FamilyName Times
Weight Medium
ItalicAngle 0.0
IsFixedPitch false
UnderlinePosition -98
UnderlineThickness 54
Version 001.000
Notice Times is a trademark of Allied Corporation.
% ------------------------------------------INFORMATIONS METRIQUES GLOBALES
EncodingScheme AdobeStandardEncoding
FontBBox -167 -252 1004 904
CapHeight 673
XHeight 445
Descender -219
Ascender 686
% ------------------------------INFORMATIONS METRIQUES POUR CHAQUE CARACTERE
StartCharMetrics 210
C 32 ; WX 250 ; N space ; B 0 0 0 0 ;
C 33 ; WX 333 ; N exclam ; B  128 -17 240 673 ;
C 34 ; WX 408 ; N quotedbl ; B 46 445 313 685 ;
C 35 ; WX 500 ; N numbersign ; B 20 -17 481 673 ;
C 36 ; WX 500 ; N dollar ; B 45 -92 456 726 ;
C 37 ; WX 833 ; N percent ; B 63 -36 771 655 ;
% lignes omises . . .
C 101 ; WX 444 ; N e ; B 24 -17 416 469 ;
C 102 ; WX 333 ; N f ; B 21 0 382 685 ; L i fi ; L l fl ; % ligatures
C 103 ; WX 500 ; N g ; B 24 -220 469 468 ;
C 104 ; WX 500 ; N h ; B 11 0 487 686 ;
C 105 ; WX 278 ; N i ; B 25 0 255 685 ;
C 106 ; WX 278 ; N j ; B -56 -217 205 685 ;
C 107 ; WX 500 ; N k ; B 7 0 496 686 ;
% lignes omises . . .   Caracteres exotiques
C 248 ; WX 278 ; N lslash ; B 0 0 283 685 ;
C 249 ; WX 500 ; N oslash ; B 30 -104 469 566 ;
C 250 ; WX 722 ; N oe ; B 30 -10 684 462 ;
C 251 ; WX 500 ; N germandbls ; B 13 0 468 686 ;
C -1 ; WX 722 ; N Aacute ; B 22 0 702 873 ;
C -1 ; WX 722 ; N Acircumflex ; B 22 0 702 875 ;
C -1 ; WX 722 ; N Adieresis ; B 22 0 702 819 ;
% lignes omises . . .
EndCharMetrics
% Suite figure suivante
```

Figure 10: Extrait de l'AFM pour le Times Roman (partie 1/2) – légende dans le texte.

```
% Suite metrique                          % Cette partie concerne l'interlettrage
StartKernData
StartTrackKern 3
Comment Light kerning
TrackKern -1 14 0 72 -1.89
Comment Medium kerning
TrackKern -2 8 0 72 -3.2
Comment Tight kerning
TrackKern -3 6 -.1 72 -3.78
EndTrackKern
StartKernPairs 2                          % listes de crenage
KPX V A -129
KPX A Y -92
. . . % lignes omises
EndKernPairs
EndKernData                               % accents composites
StartComposites 1
CC Eacute 2; PCC E 0 0; PCC acute 139 214;
EndComposites
EndFontMetrics
```

Figure 11: Extrait de l'AFM pour le Times Roman (partie 2/2).

pour les corps supérieurs à 72, on retire toujours 3.78 points entre chaque paire de lettres ;

5. les tables pour le crénage, par exemple `A Y -92` ;

6. les informations pour la composition des lettres accentuées (voir section 3.3).

Comment se procurer une AFM ? Si on achète une fonte commercialement, le fichier AFM devrait toujours être fourni, même si ce n'est pas strictement nécessaire pour l'application prévue. Les AFM des fontes Adobe de type 1 (y compris celles qui sont fournies en standard dans les imprimantes et photocomposeuses) sont disponibles par courrier électronique (voir note 13).

6. Utilisation de la métrique pour justifier une ligne

Considérons la figure 12 et voyons comment une application peut produire ces diverses lignes. Supposons que les lignes verticales représentent les marges d'un texte et qu'elles soient aux abscisses $m_g = 100$ et $m_d = 300$ (soit une justification de 200 points). Soit y l'ordonnée d'une ligne.

La ligne 1 peut être générée par :

```
100 y moveto
(Il a plu ... un mois. Mais ...) show
```

Cette ligne n'est pas justifiée. Pour ce faire, l'application peut employer l'une des méthodes suivantes.

6.1. Justification par accroissement des espaces

Le principe est d'agrandir uniformément l'espace inter-mots[15] de façon que le

[15] Dans une même ligne. Rappelons que TEX optimise cet espace sur tout un paragraphe.

1 Il a plu sur Rennes pendant un mois. Mais ...

2 Il a plu sur Rennes pendant un mois.

3 Il a plu sur Rennes pendant un mois.

4 Il a plu sur Rennes pendant un mois.

5 Il a plu sur Rennes pendant un mois. Mais

6 Il a plu sur Rennes pendant un mois.

Figure 12: Différentes justifications d'une ligne.

dernier mot ait son extrémité droite appuyée sur la marge droite du texte. Soit J la justification de la marge (ici 200 points), L la longueur de la chaîne sans espaces (*Ilaplu...unmois.*) et e le nombre d'espaces (ici 7). Alors, le blanc à répartir entre chaque mot est égal à $b = (J - L)/e$. La longueur de L est calculable puisque l'AFM donne la chasse des caractères et que l'application sait en quel corps on travaille. L'application calcule alors :

$a_1 = m_g$
$a_2 = a_1 +$ chasse (Il) $+ b$
$a_3 = a_2 +$ chasse (a) $+ b$
$a_4 = a_3 +$ chasse (plu) $+ b$
...
$a_{e+1} = a_e +$ chasse (un) $+ b$

puis génère les instructions PostScript :

```
a1 y moveto (Il) show
a2 y moveto (a) show
a3 y moveto (plu) show
...
a7 y moveto (un) show
a8 y moveto (mois.) show
```

La ligne 2 de la figure 12 a été produite ainsi. C'est la méthode utilisée pour TEX par `dvitops`.

La ligne 3, identique à la ligne 2, utilise une variante de la fonction **show** de PostScript :

d_x d_y car chaîne **widthshow**

fonctionne comme **show** mais chaque fois que l'un des caractères de la chaîne *chaîne* a pour code *car*, le point courant est déplacé de d_x, d_y.

Si on appelle L' la longueur de la chaîne avec espaces *(Il a ...un mois.)* et $b' = (J - L')/e$, alors l'application peut générer l'instruction suivante (32 est le code du caractère espace) :

```
mg y moveto
b' 0 32 (Il a plu ... un mois.) widthshow
```

6.2. Justification par interlettrage

Une autre façon de justifier est de modifier légèrement l'interlettrage, c'est-à-dire de modifier les approches entre chaque lettre. On utilise pour ça une autre variante de **show** :

d_x d_y chaîne **ashow**

imprime la chaîne comme **show**, mais en déplaçant les coordonnées du point courant de d_x, d_y entre chaque caractère (ce qui revient à modifier l'approche gauche de cette valeur). Soit alors $a = (J - T)/(n - 1)$ où T est la longueur de la chaîne (espaces compris) et n le nombre total de caractères dans cette chaîne. L'application peut alors générer les instructions suivantes :

```
mg y moveto
a 0 (Il a plu ... mois.) ashow
```

ce qui donne la ligne 4 de la figure 12.

Cet interlettrage peut aussi se faire en diminuant les approches, c'est-à-dire en prenant des valeurs négatives pour a_x. La ligne 5 a été produite par :

```
mg y moveto
a'0 (Il a plu ... mois. Mais)
ashow % avec a' < 0
```

Avant de faire ceci, une application devrait d'abord vérifier que cet interlettrage resserré est compatible avec les données du *track kerning* de l'AFM.

6.3. Interlettrage et modification des blancs

PostScript permet enfin un mélange des deux méthodes précédentes : le blanc en trop est réparti, pour une part, entre les espaces et, pour une autre part, entre les lettres. Si on prend un rapport de 10%, on calcule : $b_2 = (J-L)/(n+9e)$ et $b_1 = 10b_2$.

La ligne 6 a été générée par :

```
mg y moveto
b1 0 32 b2 0 (Il a plu ... mois.) awidthshow
```

C'est la méthode employée, par exemple, par Word 4.

7. Utilisation de la métrique sans AFM

Ainsi que nous l'avons signalé, toutes les propriétés métriques ne sont pas directement accessibles par un programme PostScript. Par exemple, le dictionnaire d'une fonte de type 1 ne donne, en principe, ni les valeurs de chasse, ni les coordonnées de la *bbox* de chaque caractère. Or, on ne dispose pas toujours de l'AFM (c'est notamment le cas d'un utilisateur normal d'une imprimante à laser).

Mais ces valeurs peuvent être calculées. Ainsi, la fonction **stringwidth** donne la chasse d'une chaîne, donc d'un caractère. De même, un caractère[16] est un dessin formé de chemins ; or l'opérateur **pathbbox** rend les coordonnées de la plus petite boîte englobant un ensemble de chemins. La figure 3 a ainsi pu être tracée, sans accès à l'AFM, par les instructions de la figure 13.

8. Propriétés dynamiques

Contrairement à d'autres méthodes (contrairement notamment à ce qui existait du temps des photocomposeuses de seconde génération et, bien sûr, du temps du plomb), PostScript calcule ses caractères soit à leur première occurrence, soit à chaque occurrence. Ceci ouvre très largement les voies de la création typographique. Voyons deux exemples.

8.1. Mesures dépendantes du corps

Du temps du plomb, les caractères avaient un dessin dépendant du corps : un "e" avait un contre-poinçon (la partie creuse de la boucle) plus grande et une graisse plus forte, proportionnellement, en corps 5 qu'en corps 50, de façon à éviter les risques de bouchage tout en restant visible. Cette finesse s'était un peu perdue, les caractères ayant souvent un tracé moyen pour toute une gamme de corps[17].

Or, PostScript permet le retour à un tracé défini en fonction du corps. En effet, lorsque la *bitmap* d'un caractère est calculée, le facteur d'échelle a déja été fixé et se trouve donc dans le dictionnaire de la fonte où sa valeur peut être accessible par l'algorithme définissant le contour du caractère.

La figure 14 montre deux caractères à la Didot produits avec la définition suivante qui calcule la force des filets horizontaux de façon inversement proportionnelle au facteur d'échelle (troisième élément de la *FontMatrix*) :

```
/I {.5 100 currentfont /FontMatrix get
    3 get 1000 mul    % facteur d'echelle
    div log 8 mul add % fct inv. prop
```

[16] Du moins ceux définis par des courbes. Ce n'est pas vrai pour les caractères batons comme *Courier*.

[17] Sauf quelques caractères particulièrement adaptés aux petits corps comme ceux définis pour l'annuaire téléphonique [Mandel 88]. Signalons qu'Adobe commence à fournir, lentement, des contours destinés à certaines gammes de corps, comme le *Times Ten* fait exprès pour le corps 10. La nouvelle famille *Minion* est dotée d'une *Minion Display* spécialement adaptée aux grands corps.

```
%! trace metriques d'un caractere en postscript et compare avec plomb

/metriquepb { % metrique plomb avec boite du caractere (talus+approches)
/car exch def
/chasse car stringwidth pop def % par calcul
currentpoint /y exch def /x exch def
x y moveto car  show
gsave
newpath x y  translate
/corps currentfont /FontMatrix get 3 get 1000 mul def % 3eme elt
0 corps -4 div moveto % valeur approximative pour base caractere
chasse 0 rlineto 0 corps rlineto chasse neg 0 rlineto % bords car
closepath stroke
grestore } def


/metriqueps { % imprimer un caractere dans sa bbox
/car exch def
/chasse car stringwidth pop def
currentpoint /y exch def /x exch def
x y moveto car  show
gsave
newpath 0 0 moveto
car true charpath flattenpath pathbbox % det bbox d'un chemin
/ury exch def /urx exch def /lly exch def /llx exch def % coins
grestore
gsave
newpath x y  translate
llx lly moveto urx lly lineto urx ury lineto llx ury lineto
closepath stroke
grestore} def

/Times-Roman findfont  250 scalefont setfont
100 600 moveto (g) metriquepb
100 300 moveto (g) metriqueps
showpage
```

Figure 13: Programme PostScript pour dessiner les boîtes de la figure 3.

```
setlinewidth      % force du filet
0 0  moveto       % tracer filets
360 0 rlineto stroke
0 660 moveto 360 0 rlineto stroke
120 0 moveto 120 0 rlineto % fut
0 660 rlineto 120 660 lineto
closepath fill } def
```

Un même algorithme suffit donc à définir tous les tracés quel que soit le corps. Richard Southall a employé une méthode équivalente avec Metafont [Southall 86].

Cette même connaissance du facteur d'échelle permet aussi de calculer le crénage en fonction du corps.

8.2. Dépendance du contexte

Rappelons enfin que PostScript permet de calculer le tracé d'un caractère au moment où ce caractère est utilisé et non une fois pour toutes pour une police donnée (au sens de la note 1). Il suffit pour cela d'utiliser la procédure `setcharwidth` au lieu de `setcachedevice`.

Dans [André & Ostromoukhov 89], ceci a été appliqué pour rendre dynamique la

Figure 14: À gauche : caractère en corps 100, à droite : caractère en corps 2 agrandi 50 fois (d'après [André 91]) ; notez la différence d'épaisseur des filets.

fonte *Punk* de Knuth. On trouvera dans [André & Borghi 89a et 89b], [Joshi 90] et [van Rossum 90] d'autres exemples d'applications. Enfin, le Delorme [André & Delorme 90] utilise cette propriété pour, par exemple, tracer automatiquement des ligatures (par exemple figure 15 haut) ou choisir le dessin d'un caractère en fonction de ses voisins (figure 15 bas).

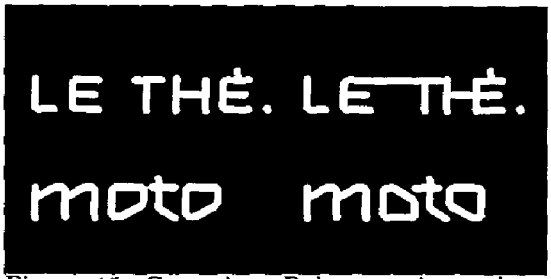


Figure 15: Caractères Delorme ; à gauche : version normale, à droite version dépendante du contexte.

9. Remerciements

Nous tenons à remercier ici Beat HIRSCHBRUNNER et Roger HERSCH pour leurs invitations à Fribourg et Lausanne.

Nous tenons, par ailleurs, à remercier les personnes qui ont bien voulu nous aider à approfondir certains points de PostScript ou à corriger une version préliminaire de cet article, en particulier Bruno BORGHI (Metasoft), Jean-Daniel FEKETE (École Estienne), Jeanine GRIMAULT (CompoScript), Victor OSTROMOUKHOV (EPFL) et Éric PICHERAL (CICB).

Bibliographie

[Adams & Southall 89] Debra ADAMS and Richard SOUTHALL, "Problems of quality assessment", in [André & Hersch 89], 213–222.

[Adobe 87b] ADOBE SYSTEMS INCORPORATED, *PostScript Language Tutorial and Cookbook*, Addison-Wesley Publishing Company, Reading, MA (USA), 1987. Traduit en français : *PostScript par l'exemple*, InterÉditions, 1987.

[Adobe 87r] ADOBE SYSTEMS INCORPORATED, *PostScript Language Reference Manual*, Addison-Wesley Publishing Company, Reading, MA (USA), 1987. Une seconde édition, plus complète notamment en ce qui concerne les fontes de type 1, vient de paraître en 1991.

[Adobe 88v] ADOBE SYSTEMS INCORPORATED, *PostScript Language Program Design*, Addison-Wesley Publishing Company, Reading, MA (USA), 1988.

[Adobe 90f] *Adobe Font Metric Files*, Specification version 3.0, PostScript Developer Tools & Strategies Group, mars 1990.

[Adobe 90n] ADOBE SYSTEMS INCORPORATED, *Adobe Type 1 Font Format*, version 1.1, Addison-Wesley Publishing Company, Reading, MA (USA), 1990.

[André 89] Jacques ANDRÉ, « Métrique des fontes en typographie traditionnelle », *Cahiers GUTenberg*, n° 4, décembre 1989, 9–21.

[André 90] Jacques ANDRÉ, « The Scrabble font », *PostScript Language Journal – International Edition*, vol.3, no.1, 1990, 53–55.

[André 91] Jacques ANDRÉ, « Adaptating contour character shape to point size », *PostScript Language Journal – International Edition*, vol.3, no.3, 1991, (à paraître).

[André & Borghi 89a] Jacques ANDRÉ and Bruno BORGHI, « Dynamic fonts », in [André & Hersch 89], 198–204.

[André & Borghi 89b] Jacques ANDRÉ and Bruno BORGHI, « Dynamic fonts », *PostScript Language Journal - International Edition*, vol.2, no.3, 1989, 6–8.

[André & Delorme 90] Jacques ANDRÉ and Christian DELORME, « Le Delorme, un caractère modulaire et dépendant du contexte », *Communication et langages*, n° 86, 1990, 65–77.

[André & Hersch 89] Jacques ANDRÉ et Roger HERSCH (eds.), *Raster imaging and digital typography*, Cambridge University Press, 1989.

[André & Ostromoukhov 89] Jacques ANDRÉ et Victor OSTROMOUKHOV, « Punk : de Metafont à PostScript », *Cahiers GUTenberg*, 1989, 4, 23–28.

[Bertrand 91] André BERTRAND, « La typographie et la loi », *Cahiers GUTenberg*, 1991, 8, 10-19.

[Borghi 89] Bruno BORGHI, « Les concepts graphiques de PostScript », *Cahiers GUTenberg* numéro 1, avril 1989, 45–51.

[Borghi 90] Bruno BORGHI, « Écrire en français avec PostScript », *Cahiers GUTenberg* numéro 6, juillet 1990, 60–64.

[Dardailler 89] Daniel DARDAILLER, « Normes et fontes », *Cahiers GUTenberg*, numéro 4, janvier 1990, 2–8.

[Eminet 87] Bernard-Paul EMINET, *Le livre de PostScript*, éditions PSI, 1987.

[Joshi 90] R.K. JOSHI, « Computerised Latin and Hindi Scripts: Back to the hand », paper presented at the non-Roman type meeting, *Type 90*, Oxford, August 1990.

[Karow 87] Peter KAROW, *Digital Formats for Typefaces*, URW Verlag, Hamburg (RFA), 1987.

[Kochan 89] Stephen G. KOCHAN, "More on Fonts", *PostScript Language Journal*, vol.2, no.2, 1989, 7–20.

[Knuth 89] Donald E. KNUTH, « TEX 3.0 ou le TEX nouveau va arriver », *Cahiers GUTenberg* n° 4, décembre 1989, 39–45.

[Mandel 88] Ladislas MANDEL, « L'écriture typographique : vers une prise de conscience », *Communication et langages*, n° 77, 1988, 5–30.

[Marti 90] Bernard MARTI et co-auteurs, *Télématique, techniques, normes, services*, Dunod-Informatique, Paris, 1990.

[Parker 89] Mike PARKER, "Fonts and PostScript", *PostScript Language Journal*, vol.2, no.2, 1989, 25–29.

[Romberger & Sunblad 84] Staffan ROMBERGER and Yngve SUNDBLAD, « Size measures and national characters in computer generated text with typography », *Protext I, proceedings of the first international conference on text processing systems* (J.J.H MILLER ed.), Boole Press, Dublin, October 1984, 222–226.

[Roth 88] Stephen F. ROTH (ed.), *Real World PostScript*, Addison-Wesley Publishing Company, Reading (USA), 1988.

[Rubinstein 88] Richard RUBINSTEIN, *Digital Typography, An Introduction to Type and Composition for Computer System Design*, Addison-Wesley, Reading (USA), 1988.

[Southall 86] Richard SOUTHALL « Designing a new typeface with Metafont », *TEX for scientific documentation* (J. DÉSARMÉNIEN ed.), Springer-Verlag, Berlin, 1986, 161–179.

[Tracy 86] Walter TRACY, *Letters of credit: a view of type design*, Gordon Fraser, Londres 1986.

[van Rossum 90] Just van ROSSUM, *Outlines from BeoSans*, FontShop International GmbH, Berlin, 1990.

[Wood 89] Patrick WOOD, "PostScript and type", *PostScript Language Journal*, vol.2, no.2, 1989, 4–6.