Chapitre 6

PILES, DICTIONNAIRES, FICHIERS

PostScript utilise couramment plusieurs piles, dont trois sont importantes pour l'utilisateur :

- celle des *états graphiques*, dans laquelle il range l'état actif sur l'ordre **gsave** et le retire sur l'ordre **grestore**, l'état empilé avant lui remontant en haut de pile et devenant état graphique actif (cf page 21) ;

- celle des *dictionnaires* qu'on étudiera page 34 ;

- la pile opérationnelle ou pile des opérandes ou *pile* tout court, déjà souvent rencontrée dans notre approche de PS et dont on va parfaire l'étude.

A la fin de ce chapitre, on parlera brièvement des fichiers, lesquels échangent des données avec la pile en entrée comme en sortie au moyen d'opérateurs spécifiques.

1 - PILE OPERATIONNELLE

Comme on l'a déjà dit, c'est dans la pile opérationnelle que l'interpréteur met en attente tous les objets lui parvenant, s'il ne les reconnaît pas comme des opérateurs (primitives ou procédures comportant des primitives). Cette pile n'est normalement accessible que par le haut (pile LIFO). C'est également là que les opérateurs déposent éventuellement le ou les objet(s) qu'ils génèrent.

La pile comporte donc des objets rangés par ordre d'arrivée. Or il se peut que cet ordre ne corresponde pas à celui des opérandes exigés par les opérateurs. Heureusement, PS possède des opérateurs capables de manipuler la pile et, en particulier, d'extraire des objets un peu *profonds* pour les faire remonter au sommet. Dans la liste qu'on va en donner, les écritures *a* et *b* symboliseront un objet quelconque et *n* un entier :

---

pop : *b a* **pop** → *b* retire (détruit) l'objet *a* situé en haut de la pile

exch : *a b* **exch** → *b a* permute les deux objets du haut de la pile

dup : *a* **dup** → *a a* duplique l'objet du haut de pile

clear : $a_n \ldots a_0$ **clear** → ∅ vide la totalité de la pile

count : ... *a* **count** → ... *a n* *n* = nombre d'objets dans la pile

index : $a_n \ \ldots \ a_0 \ n$ **index** → $a_n \ldots a_0 \ a_n$ duplique en haut de la pile son *(n*+1)ième objet

copy : $a_n \ldots a_1 \ n$ **copy** → $a_n \ldots a_1 \ a_n \ldots a_1$ duplique l'ensemble des *n* premiers objets de la pile

mark : ... *b* **mark** → ... *b* **[** place une marque en haut de la pile

[: ... *b* **[** → ... *b* **[** idem

cleartomark : b **[** $a_n \ a_{n-1} \ldots a_0$

b **[** $a_n \ a_{n-1} \ldots a_0$ **cleartomark** → b retire tous les objets au-dessus de la marque, puis la marque elle-même

counttomark : **...** **[** $a_n \ \ldots \ a_0$ **counttomark** → **...** **[** $a_n \ \ldots \ a_0 \ \ n$ compte les objets au-dessus de la marque

roll : $a_n \ \ldots \ a_{j+1} \ a_j \ldots \ a_1 \ n \ j$ **roll** → $a_j \ldots \ a_1 \ a_n \ \ldots \ a_{j+1}$

Si $j = 0$,	ne fait rien.
Si $0 < j < n$,	ôte les *j* premiers objets et les place derrière le *n*ième (*n* toujours positif).
Si $j<0$ ou $j>n$,	ajouter à $j \pm kn$ (*k* entier) pour obtenir $0 < j < n$, puis comme ci-dessus.

Exemples de **permutations extrêmement courantes** avec **roll** :

```
a b c  3 2  roll   → b c a   % très utilisé par exemple avec put (cf. page 26, fin § 3)
a b c  3 1  roll   → c a b
a b c  3 −1 roll   → b c a   % (comme 3 2 roll, puisque −1 + 3 = 2 )
```

Encore plus courant : *val* **/***var* **exch def** % quand la *val*eur à affecter à la *var*iable apparaît avant elle.

2 - DICTIONNAIRES

PS met en œuvre une pile de dictionnaires. Au fond de la pile, se trouve **systemdict** et, juste au-dessus, **userdict** . Sur ces deux dictionnaires, aucune manipulation n'est possible. Par défaut, **userdict** sert de dictionnaire courant.

Rappelons comment PS traite les caractères reçus sur la ligne d'entrée : il les groupe en mots en fonction des espaces. Si le mot représente un objet constant, il l'empile. Si c'est un nom littéral (donc précédé de la barre oblique **/**), il crée une nouvelle entrée dans le dictionnaire courant. Sinon, il recherche dans les dictionnaires l'entrée représentée par ce mot en commençant par le dictionnaire courant. Si l'objet ne figure que dans **systemdict**, il s'agit d'un opérateur et PS exécute l'instruction. S'il figure dans un autre dictionnaire, il empile sa valeur. Dans tous les autres cas, PS signale une erreur.

Le nombre d'entrées dans **userdict** est limité : 200 en général, parfois moins. Il n'est pas rare de recevoir le message

dictfull

On pourra demander avant ce message les capacités d'un dictionnaire grâce à

dico **maxlength** *dico* **length**

qui donnent le nombre maximal d'entrées et celui des entrées occupées dans le dictionnaire *dico* ; il faut remplacer *dico* par le nom du dictionnaire sondé, en général **userdict**. Si on ne peut pas restreindre le nombre d'entrées (celui des variables), il faudra installer sur la pile un nouveau volume par l'instruction

n **dict**

où *n* est le nombre d'entrées désirées. Pour en faire le dictionnaire courant, il faut écrire l'instruction

begin

dont l'action devra être annulée plus tard avec **end**, qui retire ce volume de la pile. C'est dans ce dictionnaire que s'empileront les entrées correspondant aux nouvelles variables. Mais **userdict** et les autres dictionnaires empilés restent toujours actifs (en arrière-plan) en ce qui concerne la recherche de la signification des variables anciennement enregistrées.

3 - FICHIERS

On a signalé l'existence des fichiers (pp. 5 et 9). Deux sont créés par PS à l'ouverture d'une session : celui d'entrée normale *(standard input file)* et celui de sortie normale *(standard output file)*. PS est capable de créer, d'ouvrir, de fermer d'autres fichiers et de s'en servir pour y lire et y écrire. Près d'une vingtaine d'opérateurs sont disponibles pour manipuler ces fichiers, tel celui d'ouverture :

/*fich nomdufichier* **(***c***) file def**

où *c* est un caractère autorisant écriture ou lecture : *c* = **r** ou **w** . Le *nom-du-fichier* est en général lui aussi **entre parenthèses** ; dans ce nom, le séparateur **** s'écrit **** ou **/** . L'opérateur **file** (arguments : deux chaînes) empile un identificateur qu'on affecte ici à *fich* et qui servira d'argument à tous les autres opérateurs agissant sur les fichiers. **currentfile** désigne le fichier programme en cours. On peut mentionner les opérateurs suivants *(ch*=chaîne, *ssch*=sous-chaîne):

---

write :	*fich numcar* **write**		
writestring :	*fich ch* **writestring**		**writehexstring** : *fich ch* **writehexstring**
read :	*fich* **read**	→	*numcar* ou **false**,
readline :	*fich ch* **readline**	→	*ssch booléen*,
readstring :	*fich ch* **readstring**	→	*ssch booléen*,
readhexstring :	*fich ch* **readhexstring**	→	*ssch booléen*.
exec :	*fich* **cvx exec**	→	(exécution du fichier, s'il s'agit de commandes)

Les opérateurs de type **write** écrivent dans le fichier identifié par *fich* soit le caractère de numéro *numcar*, soit la chaîne *ch*, normale (**writestring**) ou sous forme hexadécimale (**writehexstring**).

La lecture est plus délicate. Si on lit une ligne avec **readline**, la chaîne *ch* se remplit jusqu'à rencontre du code fin de ligne (exclu) **\n** (ASCII 10). Avec les opérateurs en –**string**, on lit autant de caractères que *ch* en compte. Mais si, avant la fin de ligne (avec –**line**), ou la fin de chaîne (avec –**string**), on rencontre la fin du fichier, ces opérateurs renvoient **false** (sinon **true**).

L'exécution d'un fichier-programme s'obtient avec **cvx exec** ou simplement avec (*nom-du-fichier*) **run**. On peut ainsi par exemple faire exécuter un **prologue**.

On peut également sauvegarder dans un fichier la mémoire virtuelle. On ne détaillera pas davantage ces opérateurs, car on ne peut s'en servir – à part le cas du fichier **currentfile** (cf. fichiers images p. 46) – qu'en présence d'un système d'exploitation (grosses imprimantes) ou bien avec **Ghostscript**, qui lui est capable de lire, modifier et créer des fichiers sur les disques de l'ordinateur où il est implanté.

Chapitre 7

CARACTERES ET POLICES

Le dessin vectoriel en PostScript constitue l'objectif principal de ce livre, écrit spécialement pour des informaticiens voulant créer des illustrations pour des documents. Cependant, on ne peut passer sous silence les remarquables atouts de PostScript en matière de texte, le lecteur pouvant désirer incorporer des légendes à ses dessins ou bien améliorer les prestations d'un logiciel de traitement de texte. Nous décrirons rapidement ces possibilités, sans trop nous laisser entraîner vers la typographie professionnelle, que nous ne pourrons cependant pas totalement ignorer.

1 - CARACTERES

En PS, un caractère imprimable est un dessin. C'est une plage encrée en général définie par un contour [(1)], un chemin, lui-même formé parfois de plusieurs sous-chemins. Ces chemins sont la plupart du temps définis par des courbes de Bézier. L'utilisateur peut créer des caractères, mais ce travail est plutôt œuvre de spécialiste. L'ensemble des caractères de l'alphabet (au sens large) répondant à un même souci artistique est réuni en une table appelée *police*. Les concepteurs de PS ont dessiné de nombreux caractères (370 environ) dans des polices variées fournies avec l'interpréteur, dont les plus courantes portent les noms de Times, Courier, Helvetica, Symbol...

Les caractères PS sont produits – dessinés – par des procédures dont ils portent le nom. Ce nom, formé uniquement de lettres, est toujours très significatif. Par exemple,

a ... z, A ... Z	pour les lettres de même nom
zero, one, two, ... nine	pour les chiffres 0 ... 9
plus, hyphen, comma, semi-colon, period, ...	pour les signes + - , ; .
eacute, egrave, ccedilla, ntilde, icircumflex, ...	pour les diacritiques é, è, ç, ñ, î

La liste complète en est donnée dans l'annexe 7.

Il ne faut pas oublier qu'un clavier de console informatique n'émet que des nombres, recodés bien sûr par divers logiciels pour représenter des caractères, mais que ces caractères en machine restent des nombres. La correspondance entre ces nombres et les caractères lus à l'écran est en général assurée par la table ASCII. Les conventions de PS admettent que les caractères peuvent être écrits *en clair* s'ils font partie de chaînes, donc s'ils sont placés entre parenthèses ; mais quand un caractère est hors chaîne, il doit être désigné par son numéro. On se reportera au paragraphe consacré à la manipulation des caractères (page 25) et à celui des conversions (page 30) pour bien saisir ces nuances importantes.

2 - POLICES

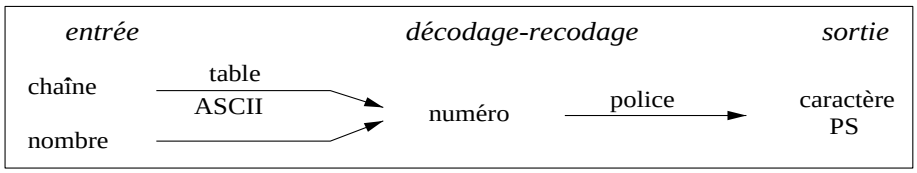
Si PS admet bien le codage ASCII en ce qui concerne l'écriture des programmes, dès qu'on veut citer des caractères imprimables, dans des chaînes ou hors chaîne, c'est une autre table de correspondance qui sera utilisée. Elle s'appelle *table de codage* (**Encoding**) de *la police en cours*. La police est un dictionnaire dont la table de codage est un sous-tableau.

D'après *Adobe*, cette table comprend 256 entrées, mais les 32 premières n'étant pas affectées, il reste 224 entrées disponibles. C'est dans cette table qu'à tout nombre compris entre 31 et 256 on associe le nom d'un caractère. Il faudra bien sûr faire un choix parmi ceux disponibles (370 environ), mais on pourra utiliser plusieurs polices, donc davantage de caractères ou bien en écrire soi-même. Les *procédures* dessinant les 224 caractères du tableau **Encoding** sont regroupées dans une autre entrée-tableau – appelée **CharStrings** – de la police concernée.

---

1 - Quelques nuances doivent être apportées à cette affirmation. Voir la section 6.

A partir des caractères-clavier ou du texte lu à l'écran, on a donc affaire à une succession de codages symbolisée par le schéma suivant :



Pour les polices ordinaires fournies par Adobe (Times, Helvetica ...), les caractères 32 à 126 compris sont ceux de la table ASCII. Il n'en va pas de même avec les polices spéciales (Symbol par exemple), ni avec les numéros supérieurs à 127. En particulier, Adobe ignore l'extension IBM au code ASCII et on ne peut pas en principe introduire les diacritiques du français (lettres accentuées, c cédille) par frappe directe sur un clavier AZERTY **(2)**.

Contrairement aux polices employées dans les autres imprimantes, une police PS ne possède pas de taille – de corps – déterminée, parce qu'elle est définie de manière vectorielle. N'importe quelle valeur de corps pourra lui être affectée. Selon une habitude courante en imprimerie, la plupart des polices comportent des **variantes**, comme l'italique *(italic)*, le gras *(bold)* et l'italique-gras *(bolditalic)* ou parfois l'oblique avec sa variante grasse **(3)**.

Une police PS est donc un ensemble de 224 caractères maximum, sans corps déterminé et admettant les variantes ci-dessus. Les polices peuvent être classées en familles : humanes, garaldes, réales ... Pour les informaticiens, il est seulement nécessaire de savoir qu'il existe des polices *fixes* – dont les caractères ont une largeur uniforme pour un corps donné – telle **`Courier`**, et des polices appelées *proportionnelles*, dont les caractères ont chacun une largeur propre. On distingue également :

- les polices à empattement *(serif)*, qui passent pour rendre la lecture plus facile, telle la très ancienne et élégante police *Garamond* ; sa cousine germaine, la *Times*, est très employée en informatique ;

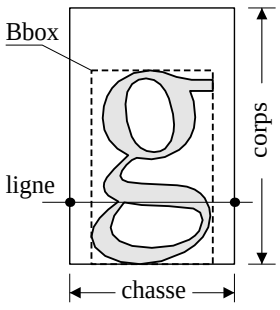
- les polices sans empattement (ou *sans serif)*, qui sont préférables dans les titres ; l'une des plus connues est l'**HelveticaNarrow**, utilisée par exemple dans ce livre pour écrire les mots réservés de PostScript ;

- des polices spéciales (Symbol, Dingbats ...) regroupant des caractères peu communs ou propres à certaines disciplines (mathématiques par exemple).

3 - METRIQUE D'UN CARACTERE

Le tracé d'un caractère est contenu dans une sorte de boîte, appelée *BoundingBox* ou *Bbox*, rectangle l'enserrant au plus près (en pointillé dans la figure ci-dessous). Les typographes appellent *chasse* la largeur totale occupée par le caractère sur la ligne : elle est supérieure à la largeur de sa *boîte*, car elle inclut deux zones de garde, une antérieure et une postérieure.

Chaque caractère possède un point origine et un point terminal, indiqués sur la figure. Ces points sont situés normalement sur la ligne d'écriture (mais pas sur la boîte) et le point origine du caractère suivant correspond au point terminal du précédent.



Dans le dictionnaire, le caractère possède une hauteur d'un pica (≈0,35mm). Il serait peu lisible, mais PS l'agrandit grâce à deux opérateurs. Le premier agit dans l'instruction

police corps **scalefont**

sur tous les caractères du premier argument *(police)*. Le deuxième argument de **scalefont**, un nombre, s'appelle *corps* de la police (par exemple, ce texte est écrit dans le corps 10 de la police Times-Roman). Le corps exprime en points typo (72 par pouce), hauteur du plus grand caractère de la police. Mais l'opérateur **scale** de la CTM agit aussi bien sur le texte que sur le dessin. Tout agrandissement provoqué par **scale** multiplie le corps défini par **scalefont**. L'argument de **scalefont** – unique – assure une transformation conforme des caractères (proportions conservées) ; par contre, **scale** , avec deux arguments différents, déforme le caractère, effet qui peut être recherché.

On ne peut connaître les dimensions d'un caractère que par des moyens indirects. Tout d'abord par **stringwidth** , qui, appliqué à une chaîne, place sur la pile 2 nombres, sa largeur et sa hauteur. Si cette chaîne a pour *longueur* 1 et si on y a placé le caractère de numéro *numcar*, on obtient la différence *l* entre les positions en *x* du point origine et du point terminal, i. e. la chasse, ainsi que *h*, nul en général, différence en *y* entre les deux points énoncés.

2 - Cette question sera réexaminée page 42, § 9-b et c.

3 - Avec l'interpréteur, Adobe vend un lot de polices, (17, 35 ...) selon le niveau de la machine (attention, dans ce nombre, chaque variante est comptée comme une police).

/chaîne **1 string def** *chaîne* 0 *numcar* **put**
chaîne **stringwidth** → *l h*

On accède aux dimensions de la boîte du caractère au moyen de **pathbbox** :

pathbbox → x_0 y_0 x_1 y_1

Si un chemin existe, cet opérateur sans argument dépose sur la pile quatre nombres, les coordonnées de M_0 et de M_1, sommets diagonaux de la boîte rectangulaire bordant le chemin. Le chemin peut ne comporter qu'un seul caractère, dont le contour aura été saisi avec **charpath** (cf. § 6).

Deux remarques : *primo*, les arêtes de la boîte sont toujours parallèles aux bords du papier, quelle que soit la CTM. *Secundo*, **charpath** fournit un chemin comprenant tous les points de construction du caractère, en particulier ceux de guidage des courbes de Bézier. Ce n'est en général pas souhaité. Aussi peut-on demander à PS de transformer un chemin (donc un caractère), initialement construit avec des courbes, en une suite polygonale de segments de droite, cela grâce à l'opérateur **flattenpath** . Ainsi

chaîne **1 string def** *chaîne* 0 *numcar* **put**
0 0 **moveto** *chaîne* **flattenpath false charpath**
pathbbox → x_0 y_0 x_1 y_1

Bien sûr, si on s'intéresse à un caractère *constant*, c'est-à-dire bien précis, on écrira seulement la seconde ligne en remplaçant *chaîne* par ledit caractère placé entre parenthèses.

Pour bien comprendre ce qu'est un caractère, il faut se placer dans le contexte de l'imprimerie traditionnelle. Les caractères sont gravés en relief dans de petits blocs rectangulaires de métal. Ceux d'une police et d'un corps déterminé ont tous la même hauteur, le *corps*. Par contre, leur chasse – largeur – est variable. Le typographe prend les caractères dans la *casse* (casier), les assemble sur une ligne à partir de la gauche et éventuellement les aligne sur la marge de droite (les *justifie)* avec des cales d'égale épaisseur placées entre les mots. L'interligne est égal – ou au moins égal – au corps, du moins dans le cas de la typographie classique. C'est également sa valeur par défaut dans la composition électronique.

4 - APPEL D'UNE POLICE

On rend une police disponible et active par l'instruction suivante

/police **findfont** *c* **scalefont setfont**

findfont va chercher la *police* dans les dictionnaires et la place sur la pile ; **setfont** en fait la police en cours, après que **scalefont**, facultatif, lui a donné le corps de taille *c* (exprimé en unités courantes).

Si l'on veut disposer de plusieurs polices, il faut les définir comme des variables et leur donner un nom avec **findfont** et **def** . Une seule d'entre elles, la dernière désignée par **setfont** , sera active à un moment donné. Pour en changer avec peu d'écritures, il vaut mieux passer par des procédures. Ainsi, pour disposer des polices **Times-Roman**, **Times-Italic** et **Times-Bold**, en corps 11, on pourra écrire :

/TRdroit **/Times-Roman findfont 11 scalefont def**
/TRital **/Times-Italic findfont 11 scalefont def**
/TRgras **/Times-Bold findfont 11 scalefont def**

/Sd **{** *TRdroit* **setfont show }** **def**
/Si **{** *TRital* **setfont show }** **def**
/Sg **{** *TRgras* **setfont show }** **def**

On remplacera alors l'opérateur **show** habituel (§ 5) par le nom de l'une des procédures ci-dessus, *Sd*, *Si*, *Sg*, selon qu'on veut écrire en caractères droits, italiques ou gras.

On a dit que les caractères des polices sont produits par des procédures. C'est un procédé intéressant (souplesse dans le choix du corps entre autres), mais lent à exécuter. Aussi, quand l'interpréteur a *calculé* le dessin d'un caractère, il en fait une image de points (comme pour tout autre objet) et range cette image dans la mémoire VM, de sorte qu'un nouvel appel à ce caractère ne donne pas lieu à sa reconstruction. Malheureusement, la capacité de cette mémoire est limitée. C'est une raison (outre celles d'ordre esthétique) pour ne pas utiliser dans un même document un trop grand nombre de polices ou de corps différents.

5 - IMPRESSION D'UNE CHAINE

L'opérateur le plus simple et le plus employé est

show

il imprime une *chaîne* à partir du point courant le long d'une ligne parallèle à l'axe *Ox* du repère en cours. Comme les 4 autres opérateurs, il exige :

- un point courant ;
- la disponibilité d'une police (grâce à **setfont**) ;
- une chaîne : l'opérateur **show** ne peut pas imprimer un ou des caractères, ils doivent être intégrés dans une chaîne avec **put** ou **cvs** .

show : *x y* **moveto** *chaîne* **show**

par exemple : 10 10 **moveto** **(** *abcdef* **)** **show**

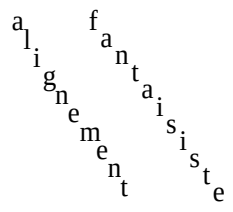
L'opérateur **show** place au point courant le point origine du premier caractère, puis le point origine de chaque caractère sur le point terminal du précédent. Après exécution, **show** établit comme point courant le point terminal du dernier caractère. Plusieurs **show** peuvent donc se succéder directement. L'impression d'une chaîne avec un seul **show** est identique à celle provoquée par des **show** successifs.

Quatre autres opérateurs autorisent moins de monotonie. Ce sont les suivants :

ashow : *dl dh chaîne* **ashow**

Cet opérateur imprime la *chaîne* comme **show** , mais ajoute après chaque caractère les déplacements *dl* et *dh* (en *x* et en *y)*. Les unités sont celles du système utilisateur. L'exemple ci-contre est obtenu avec l'instruction

```
/ch 1 string def 0 25 moveto (alignement)
{ ch 0 3 2 roll put ch stringwidth pop 2 div neg .9 add –2.7 ch ashow}
forall 15 25 moveto .4 -2.5 (fantaisiste) ashow
```



Pour le mot *alignement*, on a aligné sur l'oblique le centre de chaque caractère. Cela n'est pas nécessaire pour *fantaisiste*, dont les caractères ont des chasses plus homogènes.

widthshow : *δl δh numcar chaîne* **widthshow**

Cet opérateur imprime la *chaîne* comme **show** , mais ajoute, en plus, après chaque caractère de numéro *numcar*, un déplacement *δl* et *δh* . Cet opérateur permet en particulier la **justification** (alignement du texte à la fois à gauche et à droite). Pour cela, on doit ajouter après chaque *espace* un déplacement égal à la différence entre la largeur du paragraphe et la longueur de la ligne à imprimer (obtenue avec **stringwidth**), divisée par le nombre d'espaces [4].

awidthshow :

δl δh numcar dl dh chaîne **awidthshow**

Cet opérateur combine les effets de **ashow** et de **widthshow** , ajoutant après le caractère de numéro *numcar* le déplacement *δl δh* , en plus du déplacement *dl dh* après chaque caractère. Il peut servir à pratiquer ce qu'on appelle la **justification fine**, ajout d'espace entre chaque caractère quand la justification entraîne de grands espaces disgracieux entre les mots (la césure remédie également à cette inesthétique).

kshow : *proc ch* **kshow**

Cet opérateur imprime la chaîne *ch* de la même façon que **show** , mais il exécute la procédure *proc* entre chaque caractère. Plus exactement, après avoir imprimé le *i*ème caractère c_i , **kshow** place le point courant au point terminal de c_i , pousse sur la pile les deux caractères c_i et c_{i+1} et exécute *proc*. Par ce moyen, on peut réaliser des décalages sélectifs entre des couples de caractères donnés. Cet opérateur permet en particulier le **crénage** *(kerning* en anglais, d'où le *k* de **kshow**), c'est-à-dire la réduction de l'espacement entre des lettres telles que Ta, Va, fl ... Pour l'instant, le crénage n'est pratiqué que dans l'édition de haute qualité [5], alors que la justification est courante dans la plupart des traitements de texte.

☞ Tous les opérateurs de type **show** sont soumis à l'état graphique en cours : ils sont affectés par la matrice CTM (grandissements en *x* et en *y*, translation, rotation), par le niveau de gris et par le pochoir en cours.

6 - COMBINAISON DESSIN-CARACTERE

L'insertion globale d'un dessin dans une page déjà occupée partiellement par du texte sera développée dans le chapitre 9. C'est un cas très important et typique de l'illustration d'un article. Nous mentionnerons ici trois procédés permettant de faire participer les caractères à l'illustration du texte.

Contour d'un caractère

On peut utiliser les contours [6] d'un texte comme pochoir pour du dessin. On transforme en chemin le contour d'un caractère ou d'une chaîne avec l'opérateur **charpath** précédé de **false** [7] et suivi de **clip** :

x y **moveto (***chaîne***) false charpath clip**

---

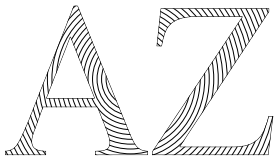
4 - PS ne fournit pas d'opérateur provoquant directement la justification, il faut écrire soi-même la procédure reposant sur **widthshow** (on justifie un paragraphe à la fois, mais les calculs nécessaires sont en général conduits en C au niveau du traitement de texte, à l'aide de tables de largeur de caractères disponibles auprès d'Adobe).

5 - En fait, pour fl, oe et ae, PostScript dispose de caractères spéciaux, fl, œ et æ, réunissant les deux lettres par *ligature*, conformément à la tradition dans l'imprimerie.

6 - Certaines polices ne sont pas définies par leur contour, mais par leur squelette, encré avec une épaisseur constante par **stroke** (par ex. *Courier)*. D'autres sont définies par des points. Les polices définies par leur contour sont en principe celles présentant des *pleins* et des *déliés*.

7 - Pour les polices définies par leur contour, le booléen exigé par **charpath** peut être aussi bien **true** que **false**. Avec les polices à squelette, **false charpath** fournit ledit squelette, tandis que **true** fournit les contours des segments de construction.

Le dessin créé ensuite ne sera visible qu'à l'intérieur du pochoir ainsi défini. Le dessin ci-contre est produit par les instructions :



```
/Times-Roman findfont 32 scalefont setfont
0 0 moveto (AZ) false charpath clip stroke
1 0.7 25 { dup 21 add 12 moveto
21 12 3 2 roll 0 360 arc stroke } for
```

Ce même **charpath** peut être employé pour fournir, grâce à **stroke** , la variante *évidée* d'une police donnée. L'instruction ci-après provoque l'écriture de la ligne suivante (police de corps nominal 10, multiplié par 2,83 avec nos coordonnées millimétriques ; même remarque pour *AZ* ci-dessus) :

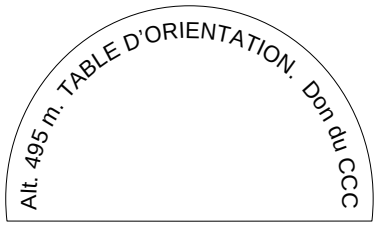
```
0 0 moveto (Bon Anniversaire) false charpath
0.02 setlinewidth stroke      % figure ci-dessous
```

7 - ECRITURE LE LONG DE COURBES

On peut écrire du texte le long d'une **courbe définie vectoriellement**. Elle peut être suffisamment simple pour qu'on puisse calculer déplacement et changement d'orientation entre chaque caractère. Si on utilise une police proportionnelle, il faut tenir compte des largeurs grâce à **stringwidth** (page 37).

Dans l'exemple ci-après, on écrira en corps "3,6 mm" la chaîne *text* au-dessous d'un cercle de rayon *r*=25mm. On commence par mesurer la longueur totale de la chaîne et la valeur de l'arc qu'elle soustend (2**depar*) afin de la centrer "à 12 heures". Partant de l'angle –*depar*, à chaque chaîne-caractère *ch*, on ajoute deux fois à l'angle actuel une rotation *dec* égale à la moitié de l'arc sous-tendu par *ch* : la première fois, pour trouver l'axe de *ch*, la seconde pour trouver l'angle de départ du caractère suivant. Une fois atteint l'axe de *ch*, on recule bien sûr de la moitié de sa chasse *l* avant de l'imprimer.

```
40 4 translate  /r 25 def  /r2 30 def  /a 3.5 def
/c  a  r2 atan def  /ch 1 string def
0  a neg moveto 0 0  r2 c neg 180  c add arc
closepath stroke                    % pourtour
/text
(Alt. 495 m. TABLE D'ORIENTATION.  Don du CCC) def
/depar text stringwidth pop  90 mul 3.14 div
r div def  0 0 moveto      depar rotate
text { ch 0 3 2 roll put
       /l ch stringwidth pop def
       /dec l r atan 2 div neg def
       0 0 moveto dec rotate l neg 2 div  r rmoveto
       ch show      dec rotate
     } forall
```



Si la courbe à suivre est **définie de manière moins simple**, par exemple avec des courbes de Bézier, on en fait un chemin, dont on demande les coordonnées des points de construction par l'opérateur **pathforall** . En plus d'un chemin, cet opérateur exige 4 procédures en argument.

De type boucle, **pathforall** décompose un chemin en ses tronçons initiaux dans l'ordre de construction. Pour chaque tronçon, il empile les coordonnées des points (arguments de l'opérateur employé), puis appelle l'une des 4 procédures, la première, la seconde la troisième ..., selon que l'opérateur de tracé était

moveto, lineto, curveto, closepath

Ces 4 procédures doivent exploiter les points de construction, présents alors sur la pile, pour guider l'écriture du texte. Deux remarques s'imposent :

- les opérateurs de type **arc** sont convertis par PS en **curveto**,

- les points associés à **curveto** sont pour moitié des points de guidage très éloignés de la courbe elle-même. Si on veut la suivre au plus près, on ne peut pas les utiliser. Aussi, on demandera que la courbe soit convertie en une suite de segments de droite par l'opérateur **flattenpath** . Dans ce cas, il est inutile de traiter le cas **curveto** qui n'apparaîtra plus.

Après avoir défini les 4 procédures, on écrit l'instruction ainsi :

```
chemin flattenpath { procmv} { procli} { procvt}
                   { proclo} pathforall newpath
```

On va donner deux exemples de texte curviligne, l'un autour d'une arche, l'autre sur l'axe d'un tracé symbolisant une rivière. Ce long programme commence par la définition de 8 procédures :

- *arcpo* (5 arguments) remplace **arcto** et les **pop** ensuite exigés ;

- *biss* (arguments: 2 nombres) calcule la bissectrice de deux angles ;

- *init* (argument: une chaîne) affecte la chaîne à la variable *text*, calcule sa longueur *nct* (nombre de caractères) et effectue des initialisations ;

- *promov* (arguments: 2 nombres), sera appelée par **pathforall** lorsqu'il rencontrera un **moveto**, initialise avec les arguments les coordonnées finales du segment courant et préserve ces mêmes valeurs dans x_{00} et y_{00} (point initial M_0) pour terminer le tracé avec *proclo* ;

- *proclo* (arguments : 2 nombres), appelée par **pathforall** s'il rencontre l'opérateur **closepath**, terminera le tracé du texte en le dirigeant vers le point initial (x_{00}, y_{00}); se borne à appeler *prolin* en lui en passant les coordonnées de M_0;

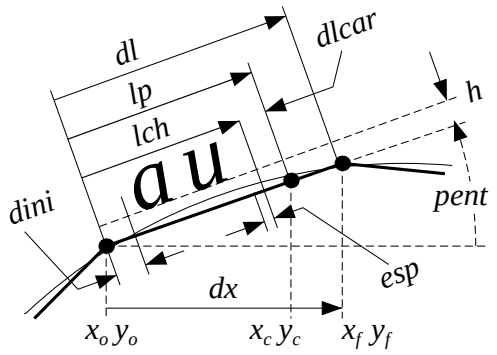
- *apelcar* (sans argument), extrait de la chaîne *text* le caractère suivant celui qui vient d'être imprimé, en fait le caractère courant *car*, calcule sa chasse (2**dlcar*) et indique avec *att* mis à *vrai* qu'un caractère est en attente ;

```
/arcpo  { arcto 4 { pop } repeat } def
/biss   { 2 copy add 2 div 3 2 roll sub abs
          180 ge { 180 add } if
        } def
/init   { dup length 0 sub  /nct exch def
          /text exch def  /nce 0 def  /att false def
          /car 1 string def /ang 0 def
        }  def
/promov {/yf exch def /xf exch def /x00 xf  def
          /y00 yf def
        } def
/proclo         { x00 y00 prolin } def
/apelcar        { /car text nce 1 getinterval def
          /nce nce 1 add def     /att true def
          /dlcar car stringwidth pop 2 div def
        }  def
```

- *ecrit* (sans argument) calcule les coordonnées x_c, y_c du point courant, fait pivoter le référentiel de l'angle *pent* autour de ce point, imprime le caractère courant après recul de la moitié *dlcar* de sa chasse et élévation d'une hauteur *h*.

- *prolin* (2 nombres en argument), procédure principale, prend les coordonnées finales du nouveau segment empilé par **pathforall**, les place dans x_f et y_f après avoir transféré dans x_0 et y_0 (début du segment en cours) les coordonnées finales du segment précédent. Elle calcule la longueur *dl* de ce segment, initialise *lch*, longueur occupée par les caractères le long de *dl*, à 0, ou à *dini* si cette distance initiale est positive. Puis elle calcule la pente du segment.

Si *lch* est nulle, elle fait écrire le caractère-en-attente au début du segment avec une pente égale à la bissectrice des segments adjacents et incrémente *lch* de la chasse plus l'espacement *esp*. Puis on entame une boucle qui, tant qu'il reste de la place, écrit les caractères le long du segment courant à la cote *lp*, incrémentant à mesure *lch*. Quand la longueur *lp* est trop faible, *dini* transmet au segment suivant la valeur de la place libre. *Prolin* ne s'exécute pas s'il n'y a plus de caractères à placer *(nce=nct)*, même si **pathforall** continue à déposer des points sur la pile et à l'appeler.

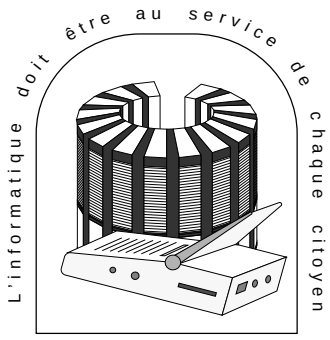



```
/ecrit  { /xc x0 pent cos lp mul add def
          /yc y0 pent sin lp mul add def
          gsave xc yc translate pent rotate
              dlcar neg h moveto car show
          grestore          /att false def
          /lch lch dlcar 2 mul add esp add def
        } def

/prolin
{ nce nct lt
  { /y0 yf def   /x0 xf def   /angp ang def
    /yf exch def  /xf exch def  /dx xf x0 sub def
    /dy yf y0 sub def
    /dl dx dup mul dy dup mul add sqrt def
    /lch dini 0 gt { dini }{ 0 } ifelse def
    dl abs  0   gt {  /ang dy dx atan def } if
    lch 0 eq
     { /pent angp ang biss def  /lp 0 def ecrit
       /lch dlcar esp add def
     } if
  } if
  { nce nct ge {exit} if    /pent ang def
    att not { apelcar } if
    lp dlcar .8 mul add dl le { ecrit } { exit } ifelse
  } loop
 ldini lch dl sub def
} def
                                  % programme "principal"
20 10 moveto     20 70 45 70 20  arcpo
70 70 70 10 20  arcpo     70 10 lineto closepath
0.25 setlinewidth stroke

/h 0 def        /dini 4 def         /esp 1.14 def
17 10 moveto     17 73 45 73 23  arcpo
73 73 73 10 23 arcpo  73 10 lineto closepath
(L'informatique doit être au service de chaque citoyen)
   init
flattenpath {promov} {prolin} {} {proclo} pathforall
newpath
```

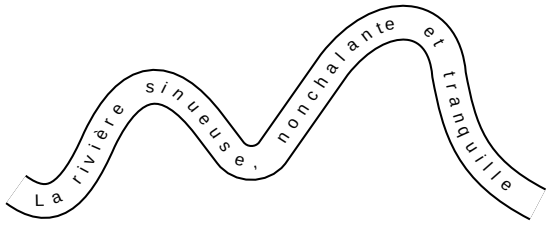
Le *programme principal* au bas de la page précédente trace l'arche ci-dessous (3 premières lignes), puis fixe la distance initiale de garde *dini*, le décalage *vertical h* du texte par rapport au chemin de référence et la valeur de l'espacement *esp*. Les 3e et 4e procédures *arcpo* construisent un chemin courant (sans l'encrer) chemin qu'exploitera **pathforall** après sa polygonisation par **flattenpath.** Les deux ordinateurs figurés sous l'arche ne sont pas dessinés par ce programme.




```
/riv
{   0 15 moveto 6.5 10 11 13 16 25 curveto
    22 38 29 39 37 28 curveto 45 17 52 26 5 arcto
   61 40 lineto  70 52 82 51 83 38 curveto
    85 27 87 19 100 12 curveto
} def

/dini 4 def      /esp 1.5 def      /h –1 def
riv gsave 7 setlinewidth stroke      riv
        6.2 setlinewidth 1 setgray stroke grestore
(La rivière sinueuse, nonchalante et tranquille) init
flattenpath { promov } { prolin } { } { proclo }
pathforall newpath
```

La procédure *riv* (rivière) est *encrée* deux fois *en différentiel* par **stroke** pour ne garder que les berges. L'exemple utilise un chemin non fermé constitué initialement (avant action de **flattenpath**) de courbes de Bézier, d'un arc de cercle et d'un segment de droite.



8 - MODIFICATION D'UNE POLICE

Si on désire modifier une police existante, par exemple la Times-Roman, on procédera par étapes de la façon suivante :

a - créer un dictionnaire (on l'appellera ici *dico*), de même taille (**length**) que celui contenant la police Times-Roman,

```
/Times-Roman
findfont dup length dict /dico exch def
```

b - recopier dans *dico* tout le contenu de Times-Roman, sauf les entrées FID,

```
{ 1 index /FID ne { dico 3 1 roll put } { pop pop }
    ifelse } forall
```

c - placer dans cette police (à l'entrée **Encoding**), la table de codage *tabcod* qu'on aura auparavant construite (voir page suivante),

```
dico /Encoding tabcod put
```

d - indiquer que *dico* est un dictionnaire de police et lui donner un nom, par exemple *polf*,

```
/polf dico definefont pop
```

e - établir cette police *polf* comme police courante, ou bien la rendre disponible, selon les procédés de la page 37, §4.

```
/polf findfont 10 scalefont setfont
```

On trouvera un exemple de modification partielle de la table de codage dans la section ci-après (§ 9b). Au cas où l'on voudrait redéfinir complètement une police, cette table devrait comprendre toutes les entrées de 32 à 255 comprises, sauf celles auxquelles on ne veut pas affecter de caractère, mais qui recevront tout de même l'affectation **.notdef** (ne pas oublier le point). On pourra trouver dans *Adobe* un exemple de création de police.

9 - FRANCISATION D'UNE POLICE

Les polices fournies par PS ne permettent pas d'imprimer les diacritiques (lettres accentuées, ç, ...). La solution rigoureuse de ce problème consiste dans la redéfinition de la table de codage. Mais les usagers occasionnels de PS, surtout ceux qui ne désirent écrire que quelques légendes dans leurs figures, pourront utiliser l'un des procédés ci-après.

a - Accents et cédille existent bien dans la table de codage des polices normales Adobe, mais comme caractères séparés (n° 193-196 et 203). On peut mesurer la largeur l_1 de la lettre à accentuer et celle de l'accent (l_2), imprimer le texte jusqu'à cette lettre comprise, reculer de la moitié de la somme ($l_1 + l_2$), imprimer l'accent et revenir au point courant pour continuer. Les procédures *aig, gra, cir, til, ced, tre* ci-après traduisent ce mécanisme et impriment accent grave, aigu, circonflexe, tilde, cédille ou tréma sur (ou sous) la dernière lettre de la chaîne-argument (pour le tréma, il a fallu auparavant effacer le point éventuel sur le *i*). Employer **show** pour les chaînes sans accents.

```
/ch1 1 string def
/recul {stringwidth pop neg 2 div 0 rmoveto } def
/ecr{dup dup show currentpoint
     /y exch def /x exch def length 1 sub get
      ch1 0 3 2 roll put ch1 recul
    } def
/acc {ch1 0 3 2 roll put ch1 recul ch1 show
      x y moveto} def
/aig {ecr 194 acc} def /gra {ecr 193 acc} def
/cir {ecr 195 acc} def /til {ecr 196 acc} def
/ced {ecr 203 acc} def
/tre {ecr 199 gsave 3 1.1 scale 1 setgray acc
          grestore 200 acc} def
```

Ex : (Ac) *ced* (ores ou Carai) *tre* (bes, i) *cir* (les ce)
aig (le) *gra* (bres) **show**
→ *Açores ou Caraïbes, îles célèbres*

b - La seconde méthode est beaucoup plus pratique, mais elle viole les règles de PS (cf. *Adobe)* indiquant que les caractères numéro 127...160 (177...240 en octal) sont des caractères de commande ASCII. On peut essayer de redéfinir cette partie de la table de codage avec des instructions qui affectent les lettres accentuées aux entrées de même numéro que celui émis par la touche sur un clavier AZERTY MS-DOS :

```
/tabcod StandardEncoding 256 array copy def
0 2 35
{ /i exch def
[ 129 /udieresis   130 /eacute      131 /acircumflex
  132 /adieresis   133 /agrave      135 /ccedilla
  136 /ecircumflex 137 /edieresis   138 /egrave
  139 /idieresis   140 /icircumflex 145 /ae
  146 /AE          147 /ocircumflex 148 /odieresis
  150 /ucircumflex 151 /ugrave      164 /ntilde
]  dup i get exch i 1 add get tabcod 3 1 roll put
} for
```

Avec cette modification, l'écriture **sous DOS** de diacritiques dans les chaînes ou bien leur désignation sous forme de séquences d'échappement provoque leur impression correcte, mais sans garantie quant à la *portabilité* du procédé.

Pourtant, le Macintosh possède un clavier dont le codage déborde dans le domaine déclaré réservé par *Adobe*. Sa table de codage est donnée en annexe 10.

Bien sûr, dans le cas MS-DOS comme dans celui des Macintosh, il faut suivre toute la méthode développée dans la section précédente (§8) pour rendre ces modifications opérationnelles.

c - Il existe un moyen bien plus simple, si le travail en PS se borne à créer des dessins destinés à être importés dans un texte écrit sous logiciel francisé. En effet, ce logiciel chargera certainement une police française qui fera partie de l'état graphique. Si dans le dessin on ne redéfinit aucune autre police, selon la règle de la page 21, elle restera active avec le dernier corps utilisé. Les procédés d'importation sauvegardent l'état graphique du logiciel père, mais le dessin héritera des caractéristiques non modifiées, donc de la police. Il faut cependant veiller à ne pas modifier la police en service juste avant l'appel du dessin ni son corps. Si, dans le dessin, on modifie les échelles, ne serait-ce que pour travailler en millimètres, le corps de la police s'en trouvera modifié. Il faudra corriger cet effet pervers en remettant à la bonne échelle la police en cours (**currenfont**) grâce à un **scalefont** muni d'arguments divisant le corps par l'opérande du **scale** précédent. Par exemple, si on travaille en millimètres, on donnera le corps 11 à la police grâce à :

```
currentfont 11 2.835 div scalefont setfont
```

d - Sous **Windows**, c'est encore mieux, puisque la table Windows ne diffère pas beaucoup (à 32 lettres peu utiles près, sauf œ et Œ) du codage ISOlatin1 que reconnaît PS niveau 2. Pour que PS niveau 2 transcrive correctement les caractères écrits sous Windows, il suffit d'écrire :

```
/tabcod ISOLatin1 Encoding def
tabcod 156 /oe put tabcod 140 /OE put
/polf {findfont dup length dict /dico exch def
{1 index /FID ne {dico 3 1 roll put}
        {pop pop} ifelse } forall
dico /Encoding tabcod put /ppp dico definefont pop
/ppp findfont exch scalefont setfont } bind def
```

On appellera par exemple la police Times 9 ainsi :

```
9 /Times-Roman polf
```

La ligne injectant œ et Œ dans *tabcod* n'est utile que si on utilise ces caractères. Sinon, on peut l'omettre. Si oui, ces lettres seront écrites soit par les moyens Windows, soit sous la forme **\234** et **\214**.

Chapitre 8

IMAGES EN DEMI-TEINTES

Après l'étude des moyens de PostScript en matière de dessin vectoriel et d'impression des caractères, il nous reste à examiner ses possibilités concernant l'illustration par des images en demi-teintes.

L'étude des niveaux de gris et celle du remplissage des surfaces nous ont déjà amenés à envisager des possibilités d'images plus nuancées – pas uniquement formées de traits – comme celles des artistes ou les photographies.

En effet, l'incorporation de telles images dans une publication a toujours été une préoccupation constante pour les imprimeurs. Pour mieux en saisir l'importance, on va commencer par un bref survol historique de la question.

1 - HISTORIQUE

L'imprimerie a toujours eu beaucoup de difficultés à reproduire les images. Avant Gutenberg, on peut dire que le texte lui-même était une image, car il était gravé comme elle *en relief* sur des plaques de bois. Le procédé était si laborieux et si lent qu'il rivalisait mal avec le travail des copistes. En introduisant le caractère métallique mobile, Gutenberg a fait faire un bond de géant à l'impression du texte, lui permettant de disposer de caractères préfabriqués et réutilisables. Cependant, le procédé n'améliorait en rien la reproduction des images, qui continuèrent pendant longtemps à être gravées en relief sur des formes en bois, heureusement insérables entre celles supportant les caractères, d'où possibilité d'illustrations dans le texte.

Quand la gravure sur cuivre fut découverte, l'image devint bien plus facile à graver dans la matière (c'est sans doute depuis cette époque qu'on appelle une image une gravure). Mais le type de transfert d'encre (à cause de la gravure qui sera désormais *en creux)* et l'épaisseur des supports étaient si peu compatibles avec ceux des caractères qu'on les mélangeait rarement et qu'on rejetait les illustrations dans des pages hors-texte.

On gagna encore en qualité quand, après la *taille-douce* (au burin ou à la pointe), on inventa la gravure *à l'eau-forte* (à l'acide). Mais on ne pouvait graver ainsi que des traits et l'imprimerie n'était capable que de déposer du noir sur une page blanche ; c'était du tout-ou-rien, ou, si l'on veut, un procédé binaire. Pour simuler la peinture ou la réalité, les graveurs inventèrent les hachures. De loin, ces hachures, selon leur densité, simulaient différentes teintes, différents niveaux de gris, bien que, vu de près, le procédé restât toujours binaire. On commença peut-être à parler de résolution. La lithographie (dessin au crayon gras sur une pierre spéciale) marqua un grand progrès pour le graveur, affina la représentation des demi-teintes (dues maintenant à la porosité de la pierre), améliora la résolution, mais exigea encore plus impérieusement l'impression séparée des illustrations et du texte.

Parallèlement, la composition manuelle *(à la casse)* cédait le pas à une composition mécanique caractère par caractère *(monotype)* ou ligne par ligne *(linotype)*. Sur le clavier de ces machines, le typographe sélectionnait des moules qui, après assemblage automatique, recevaient du plomb fondu et donnaient une ligne convenant à l'impression. Mais la reproduction de l'image ne tirait toujours aucun avantage de ce procédé.

Puis vint Nicéphore Niepce et la photographie. La photogravure s'ensuivit : elle consistait à impressionner un cliché, puis une plaque métallique (en zinc initialement) avec l'image de la page à imprimer. La méthode était rapide et la résolution excellente. De plus, la flexibilité du support permettait de l'enrouler sur un cylindre, d'où apparition de l'imprimerie rotative, à haute productivité. Texte et illustrations pouvaient se rejoindre sur la même plaque, mais l'encrage de la plaque et l'impression elle-même étaient et sont restés toujours binaires. Comment alors reproduire les photos dont la résolution avoisinait le micron ? En traduisant encore les niveaux de gris en hachures ? Presque, puisqu'on inventa le tramage **(1)**.

---

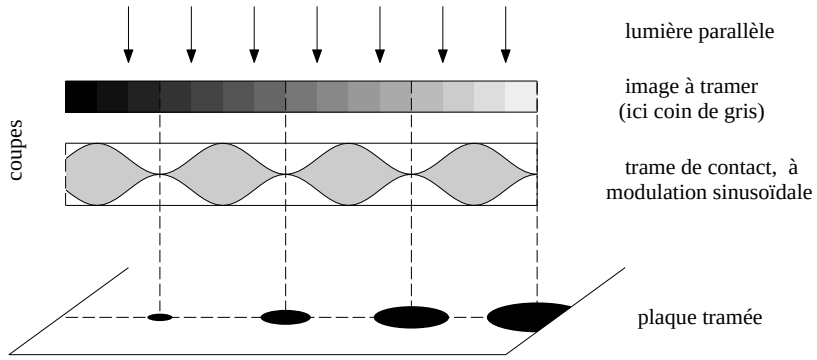
1 - Nuance importante : les hachures simulent les gris avec des traits d'épaisseur constante à intervalle variable ; le tramage avec des taches de diamètre variable à intervalle fixe.

2 - LE TRAMAGE

C'est le procédé encore utilisé de nos jours pour imprimer les photos. Il consiste, partant d'une image semi-transparente, à fabriquer un négatif tramé, c'est-à-dire échantillonné. Le dessin initial est transformé en un *réseau de taches* à **intervalles fixes**, mais dont le diamètre varie avec le niveau de gris de l'original en cet endroit. Pour y parvenir, différents procédés peuvent être employés. Le plus ancien utilisait la diffraction de la lumière par une grille faite de fils fins parallèles croisés *(trame cristal)* et dont on formait l'image dans une chambre noire (dite de reproduction) sur une plaque photosensible après traversée de l'original. Actuellement, on utilise plutôt une trame à variation progressive, formée en photographiant un réseau de franges laser. Ce genre de trame, placée au contact de la plaque, module le diamètre des taches plutôt par absorption que par diffraction (figure ci-contre).

Pour imprimer correctement une image tramée, il faut que les procédés d'encrage-transfert maintiennent la séparation des taches, sinon tout serait noir. On est donc limité par deux bornes : le pas de la trame doit être suffisamment élevé pour que les taches ne soient pas jointives et suffisamment faible pour que l'œil ne sépare pas les taches et en retire seulement l'impression d'un gris plus ou moins profond. L'inverse du pas de la trame s'appelle la *linéature* de l'image. On utilise des linéatures de 70 *lignes* par pouce pour imprimer les quotidiens et des linéatures plus élevées (150, 250 ...) pour les revues de qualité **(2)**. En anglais, le tramage s'appelle *screening*, la linéature *screening frequency* et la demi-teinte *half-toning*.

Nous ne sommes maintenant pas très éloignés de PostScript, mais avant d'y revenir, il nous faut étudier les liens apparus entre l'image et l'informatique.



3 - L'IMAGE SERIALISEE

Pour communiquer une image, on peut la reproduire sur papier et la faire parvenir au destinataire par poste. C'est une transmission de type parallèle. On doit utiliser une transmission série si le canal de communication ne peut transmettre qu'une information à la fois. C'est le moyen utilisé par le **bélinographe** pour envoyer des images par téléphone. L'image est décomposée en lignes, chaque ligne étant scrutée par un capteur et le niveau de gris de chaque point envoyé sur le canal, sous forme d'un signal continu. La transmission est de type série analogique. C'est également le principe des émissions de télévision.

Quand l'informatique a voulu *saisir* les images ainsi transmises, il a fallu numériser l'amplitude analogique des signaux et d'abord les échantillonner, c'est-à-dire les décomposer en une suite de points équidistants. Les bélinos ne posèrent guère de problèmes, leur faible débit n'exigeant pas des codeurs analogiques-numériques (CAN) très rapides. Les émissions de télévision ne purent être numérisées qu'après l'apparition de codeurs très rapides, appelés codeurs vidéo, capables de coder (avec une résolution de 8 bits) **(3)** un point toutes les 75 ns au moins, pour un débit de 25 images par seconde. La *numérisation* de la télévision pose donc de gros problèmes de bande passante, qui ne seront correctement résolus que par les procédés de compression d'image.

Plus simple apparaît la décomposition en éléments binaires d'une image fixe, une photo par exemple. Le bélinographe en est capable, s'il est suivi d'un échantillonneur codeur. Mais c'est l'arrivée sur le marché grand public des réseaux de diodes, puis des CCD, capteurs photoélectriques fonctionnant en parallèle, qui a popularisé le procédé. Le bélino s'est appelé *scanner* et l'image décomposée en rectangles, dont chacun éclaire un des éléments (pixels) rectangulaires du capteur. Le résultat est donc tout à fait voisin de celui apporté par le tramage des clichés imprimés.

2 - N.B. : Certaines imprimantes de bureau, encore peu courantes, n'utilisent pas le procédé du tramage pour représenter les niveaux de gris. Elles sont capables d'imprimer directement un *à-plat* (une plage) uniformément gris ou coloré par mélange intime des couleurs composantes

3 - L'œil moyen ne distinguerait pourtant, paraît-il, pas plus de 64 niveaux de gris (6 bits).

4 - MEMORISATION DE L'IMAGE

Une image télévisée est normalement formée de 625×625×4/3 pixels (plus d'un demi-million). En traduire le niveau de gris ou l'intensité à mieux que 0,5 % près exige 8 bits par pixel. Une image N&B de ce type occupe donc un demi-mégaoctet de mémoire. Une image couleur en exige trois fois plus. Ce sont des quantités considérables. On se contente souvent de moins : les niveaux de gris sont codés sur 8, 4, 2 ou 1 bit selon le rendu souhaité (donc avec 256, 16, 4 ou 2 valeurs). Malgré tout, il sera souvent indispensable de procéder à une *compression* de l'image.

Dans le plus simple des procédés de compression, on examine s'il existe des séquences de pixels répétitives, ce qui est fréquent dans les dessins. On répartit les codes de gris en deux catégories : ceux qui se répètent pour former une séquence de n+1 pixels identiques et ceux, en nombre m+1, qui se suivent de manière disparate. La séquence répétitive est traduite par deux octets, un nombre-guide n ($129 \leq n \leq 255$), puis l'octet indiquant la valeur de gris, unique, qu'il faudra répéter 257–n fois à la décompression **(4)**. Les séquences non répétitives débutent par un nombre-guide de valeur m ($0 \leq m \leq 127$) donnant le nombre m+1 de pixels dans la séquence. Chaque séquence est donc limitée à 128 octets avant compression.

La structure des fichiers informatiques étant linéaire, la mémorisation des images exige des informations pour les reconstituer. En télévision, on dispose de signaux de synchronisation signalant le début des lignes et des images. De telles informations doivent nécessairement être ajoutées aux fichiers d'image. Si l'on veut échanger de tels fichiers, il faudra s'accorder sur l'écriture de ces renseignements. D'où l'établissement de conventions. Ces conventions étant assez nombreuses, un marquage très lisible doit permettre d'abord d'identifier le procédé employé. Sous DOS par exemple, un suffixe comme BMP, GIF, TIF ... indique que le fichier contient une image décrite selon des conventions spécifiques.

Nous ne décrirons – et encore très succinctement – que les fichiers TIFF (suffixe TIF sous DOS et OS2), parce que c'est sans doute le format le plus général. Presque tous les *scanners* offrent la possibilité de créer des fichiers TIFF, ainsi que beaucoup d'appareils professionnels générant des images.

5 - FICHIERS TIFF (version 5)

Un fichier TIFF comprend trois groupes d'information, très inégaux :

- un en-tête très court de 8 octets
- des zones de descripteurs *(tag)*
- les zones d'image proprement dites.

En-tête (8 octets) : les 2 premiers sont 4949h (=$73_{10}73_{10}$=II, comme *Intel)* si les nombres (entiers ou les longs) doivent être lus dans l'ordre poids-faible-poids-fort, sinon ces 2 octets sont 4D4D (=MM, comme *Motorola)*. Les octets 2 et 3 (la numérotation commence à zéro) donnent la version TIFF et les octets 4,5,6,7 l'adresse de la première zone de champs descripteurs.

Zone de descripteurs : elle se décompose elle-même en 3 parties :

a - la première, de 2 octets, donne le nombre de champs n de la partie principale.

b - la seconde, la partie principale, comprend n champs, chacun de 12 octets, décrits ci-après.

c - la dernière (4 octets) est 0000 si la description du fichier est terminée, sinon elle donne l'adresse de la zone de descripteurs suivante.

Un champ de descripteurs (12 octets) apporte 4 informations :

- 2 premiers octets : titre (*étiquette*) du champ
- 3e et 4e octet : type du champ
  - 01 : type octet (L=1 octet)
  - 02 : type ASCII (L=1)
  - 03 : type entier sans signe (L=2 octets)
  - 04 : type long sans signe (L=4 octets)
  - 05: type fraction (2 longs: numérateur, dénominateur)
- 5,6,7,8e octets : nombre N de données dans le champ
- 9,10,11,12e octets : valeur du champ si le produit NL est au plus égal à 4 octets, sinon adresse où cette valeur est contenue.

Les étiquettes des différents champs, leur type, leur longueur (ou nombre de données) ainsi que leurs valeurs possibles sont résumés dans l'annexe 3.

Zone image : elle commence à une adresse donnée par le descripteur 273; la largeur et la hauteur de l'image sont données par les descripteurs 256 et 257. Une image peut être répartie en plusieurs (N_s) bandes *(stripes)*, qu'on identifie grâce à leurs N_s adresses dans le descripteur 273.

---

4 - Si on accepte que ces octets soient signés, le nombre guide n sera négatif, compris entre -127 et -1 et le facteur de répétition sera égal à $1 + |n|$.

Le nombre de bits (1, 2, 4 ou 8) codant le niveau de gris est donné par le descripteur 258. Si l'image est en couleurs, ce descripteur comporte 3 valeurs, une par composante RVB. C'est le descripteur 262 (polarité) qui indique si l'image est en couleurs (valeur 2 ou 3) ou en noir et blanc (valeur 0 si le blanc correspond au niveau de gris 0, 1 si c'est au 1).

6 - TRAITEMENT DES IMAGES EN POSTSCRIPT

En matière d'images, PS possède peu d'opérateurs, mais ils sont puissants. Le plus important, **image** , imprime une image en demi-teintes. Il s'écrit :

nx ny nb **[** *matrice* **]** **{** *procédure* **}** **image** (1)

- *nx* et *ny* : nombre d'échantillons (ou pixels) de l'image en *x* et en *y*,
- *nb* : nombre de bits par pixel (codage du gris),
- *matrice* : tableau de 6 nombres, formant une matrice au sens de PS,
- *procédure* : procédure qui sera exécutée jusqu'à ce qu'il n'existe plus de données ou jusqu'à ce que les $nx \times ny \times nb$ bits nécessaires à la construction de l'image aient été fournis à l'opérateur **image**.

La matrice est celle qu'il faudrait appliquer à une image de dimensions 1×1 unité orientée selon les conventions PS (origine en bas à gauche) pour en faire l'image générée par la *procédure*. On peut dire aussi que l'inverse de la matrice transforme l'image d'origine ($nx \times ny$ unités) en une image PS de 1×1 point. Il faudra donc redimensionner cette image de trop petite taille avec **scale** et **translate** qui fixent le point origine et les dimensions de l'image imprimée, leurs arguments étant exprimés en unités courantes (ici mm) .

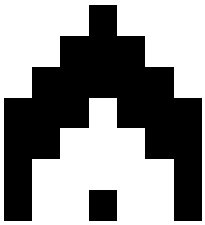
En ce qui concerne la *procédure*, il serait préférable d'écrire *"chaîne ou procédure créant une chaîne ou l'extrayant d'un fichier*". On peut en effet placer directement une chaîne entre les accolades. Adobe précise que cette chaîne doit être écrite sous forme hexadécimale (cf. p. 5). L'opérateur **image** interprète tous les bits de la chaîne comme des valeurs du niveau de gris (2^{nb} niveaux) des $nx \times ny$ échantillons successifs de l'image. Par exemple :

```
48 21 scale 16 1 4 [16 0 0 1 0 0]
    { <0123456789abcdef> } image          (2)
```

imprime l'image ci-après, formée de 16 barres ombrées avec un niveau de gris variant de 0/16 à 15/16, le dénominateur 16 étant dû à la valeur 4 pour *nb*. L'opérateur **image** appelle 16×1×4 fois un bit de la chaîne-argument et en fait une image de 16×1 pixels, dont la matrice fixe les dimensions à 16×1 picas. Tandis que :

```
24 30 scale   8 7 1 [8 0 0 –7 0 7]
      { <EFC78311397D6D> } image
```

réalise la figure pyramidale suivante, avec des pixels noirs ou blancs (*nb* = 1 bit). Chacune des 7 lignes de l'image comprend 8 pixels définis par 1 bit.



.... 1110 1111 = EF
.... 1100 0111 = C7
.... 1000 0011 = 83H
.... 0001 0001 = 11H
.... 0011 1001 = 39H
.... 0111 1101 = 7D
.... 0110 1101 = 6D

1 2 3 4 5 6 7 8 pixels

Cependant l'opérateur **image** sera en général plutôt appelé non pour créer une image, mais pour imprimer une photo au préalable mémorisée dans un fichier. On écrira alors :

```
/ima { gsave nx ny nb [ηnx 0 0 εny αnx βny]
       { currentfile ch readhexstring pop} image
         grestore showpage
       } def                                  (3)
/ch nx string def x0 y0 translate l h scale ima
```

Les nombres hexadécimaux traduisant les octets de l'image doivent être situés juste après le retour-chariot qui suit l'appel de la procédure *ima*. L'opérateur **readhexstring** viendra les lire, puisqu'ils sont dans le fichier **currentfile**. Il remplit la chaîne *ch*, puis dépose **true** sur la pile (qu'on retire avec **pop**). La matrice réoriente l'image. Normalement, $\varepsilon=\eta=1$ et $\alpha=\beta=0$. Si dans le fichier, l'image commence par le haut, on fera $\varepsilon=-1$ et $\beta=1$. Si elle est inversée (droite pour gauche), on prendra $\eta=-1$ et $\alpha=1$.

Si l'image est contenue dans un fichier non comprimé, on pourra extraire en C la seule zone image et la faire précéder de la procédure *ima* ci-dessus, après avoir extrait du fichier les descripteurs de l'image. Si le fichier est comprimé, l'opérateur **image** est d'un emploi délicat, mais PS niveau 2 fournit des opérateurs décodant directement de tels fichiers, sans toutefois en extraire les descripteurs, travail qu'il faudra assurer à part. Rappelons qu'au lieu d'inclure le fichier image dans le fichier PS, on peut, dans certains cas, faire ouvrir ce fichier par l'interpréteur (voir p. 34, §3). **currentfile** est alors remplacé par le pointeur défini par **file**. Il est de toutes façons, nécessaire d'avoir des notions précises sur les fichiers images pour programmer leur impression.

Si l'image est mal orientée, on la redressera à l'aide de la matrice (avec η=–1 et α=1). On se rappellera également que la longueur en octets de la partie image d'un fichier est souvent

$$l = ny * ((nx + 7) / nb) / 8$$

où la division par 8 est de type entier (perte de la fraction décimale). Ceci est dû au fait que les lignes contiennent en général un nombre entier d'octets.

Normalement, il faudrait convertir chaque octet du fichier en une chaîne de deux caractères représentant sa valeur hexadécimale, donc ne contenant que des symboles **0...F**. En fait, il semble qu'on ne soit pas obligé de coder les données image en nombres hexadécimaux. Si le fichier est sous forme binaire (octets de 0 à 255 compris), on peut essayer de remplacer **readhexstring** par **readstring**. Cet opérateur lit une chaîne normale (binaire) et certaines imprimantes acceptent cette simplification. Mais les normes PS (jeu ASCII réduit à 32-127) ne seront pas respectées.

Encore deux remarques : *primo*, si la taille du fichier est inférieure à *(nx×ny×nb)*/8 octets, **image** reprendra la lecture de la chaîne de données jusqu'à construction totale de l'image, qui sera donc en partie dédoublée *(fantôme)*. *Secundo*, au lieu d'envoyer le fichier image après appel de la procédure *(ima)*, le logiciel de mise en forme peut inclure le fichier, codé en hexadécimal, entre les symboles **{ <** et **> }** , à la place de la procédure argument de l'opérateur **image**. On utilisera alors l'instruction de la forme (2). Il ne faudra donc pas oublier de faire suivre le fichier-image par les symboles appropriés, puis par les opérateurs de traitement, c'est-à-dire par :

>} image grestore showpage

Reste à signaler le deuxième opérateur d'images de PS. Il s'agit de **imagemask** qui s'utilise comme **image**, à un argument près :

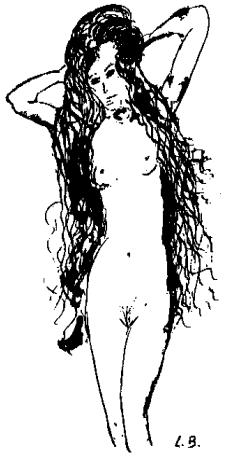
nx ny booléen **[** *matrice* **]** *procédure* **imagemask**

Cet opérateur ne peut traiter que des fichiers N-B (2 niveaux de gris, un seul bit par pixel). Il *dépose* sur le papier seulement les pixels noirs et laisse la page intacte à l'endroit des pixels blancs (valeur 1). Cela à condition que le *booléen* (3e argument) soit égal à **false**, sinon c'est l'inverse. Cet opérateur est spécialement utilisé pour créer des caractères *bit-map*.

Exemples :

La première image ci-après est un dessin d'artiste saisi par un *scanner* avec 2 niveaux de gris (1 bit par pixel) sous forme de 512 lignes de 411 pixels, puis mémorisée dans un fichier TIFF non comprimé. Un logiciel en langage C en a extrait la seule partie image, l'a codée, insérée entre les groupes **{<** et **>}** et encapsulée dans les instructions appropriées, pour envoyer finalement à l'imprimante :

```
%!PS-Adobe-2.0
gsave 72 25.4 div dup scale 121 116 translate
54 72 scale 411 512 1 [-411 0 0 -512 411 512]
{ <FFFF... (52608 octets) ...FFFFFF> } image
grestore showpage
```



La deuxième image a été saisie sur photo par *scanner* avec 256 niveaux de gris en 254 lignes de 172 pixels. Mémorisée dans un fichier TIFF, elle est traitée comme la précédente, puis imprimée par les instructions suivantes :

```
%!PS-Adobe-2.0
gsave 2.835 dup scale 116 33 translate
50 69 scale 172 254 8 [172 0 0 -254 0 254]
{ <6F7476... (87376 octets) ... 2120 > } image
grestore showpage
```

Le rendu d'une telle photo dépendra beaucoup des moyens d'impression.

Dans l'annexe 6, on montrera comment imprimer en PS niveau 2 des fichiers images comprimés et spécialement ceux de type JPEG, en pratique réservé aux photos (on utilisera des *filtres* décodeurs).

7 - LES NIVEAUX DE GRIS EN POSTSCRIPT

PS éprouve, à représenter les demi-teintes, les mêmes difficultés que celles rencontrées par l'imprimerie. L'imprimante laser ne sait déposer sur le papier que des points d'encre noire. Sa résolution (en points par pouce) donne leur densité linéique. Leur diamètre est constant et tel que, s'ils sont tous imprimés dans une zone donnée, cette zone sera noire. Si un certain pourcentage des points est absent, la zone paraîtra grise. En principe, le niveau de gris de PS, déterminé par **setgray** est égal au rapport des points absents.

L'opérateur **settransfer** permet d'établir une meilleure correspondance entre ce rapport et le niveau de gris, ceci pour tenir compte d'une réponse non linéaire dans l'appréciation humaine des demi-teintes. Ainsi, avec beaucoup d'imprimantes, la gamme 0,9-1 paraît contenir plus de nuances que celle s'étendant de 0 à 0,1 (exemples page 19). On aurait alors avantage à modifier la loi d'évolution de la densité de points en fonction du niveau de gris souhaité. L'opérateur **settransfer** est également utilisé pour inverser la polarisation d'une image (en faire le *négatif*, noir pour blanc et vice versa). On écrit simplement :

```
currenttransfer { 1 sub exch } settransfer
```

De même, l'opérateur **setscreen** permet de composer la loi la plus appropriée à la génération des taches. Le procédé normal de rendu des demi-teintes est calqué sur celui du tramage. La densité de taches est constante, mais leur diamètre varie : dans un pixel, le gris le plus faible est traduit par un unique point central ; à mesure que le niveau de gris croît, on agglutine autour de ce germe les points les plus proches, pour obtenir, comme dans l'imprimerie, un réseau tramé de maille constante dont les taches ont un diamètre croissant. PS définit un rayon *r*, fonction de la densité de gris, tel que seront imprimés tous les points de l'imprimante disponibles dans le cercle de rayon *r*.

Il existe évidemment une relation entre les trois caractéristiques de la demi-teinte : linéature, résolution (mécanique, ultime) de l'imprimante et nombre de niveaux de gris. Ce dernier est égal, à une unité près, au nombre maximum de points-imprimante contenus dans l'échantillon élémentaire

$$\textit{nombre de niveaux de gris} = 1 + (\textit{résolution} / \textit{linéature})^2$$

Une imprimante dont la résolution est de 300 dpi peut rendre 65 niveaux de gris en linéature 37,5 ou 17 en linéature 75 (qualité des images des journaux) tandis qu'une imprimante de résolution 1200 dpi dispose de 65 niveaux de gris en linéature 150, qualité d'impression des hebdomadaires.

8 - AFFICHAGE DES IMAGES A L'ECRAN

Bien que non lié directement à PostScript, le procédé d'**affichage** des images en demi-teintes se doit d'être rappelé. Les fichiers images n'indiquent pas comment représenter les gris. Le programmeur qui veut les afficher doit en principe définir une surface échantillon permettant, selon l'équation précédente, de représenter un pixel avec ses niveaux de gris ; il sera limité par la résolution de son écran. Par exemple, 16 niveaux de gris exigeant un échantillon de 4×4 pixels, un écran VGA permettra alors de représenter environ 160 pixels de ce type en *x* et 120 en *y*.

On n'emploie pas un procédé de type tramage, mais celui du **matriçage**. L'informaticien définit point par point comment une matrice élémentaire représentant un pixel (ou échantillon) sera remplie en fonction du niveau de gris. Si le nombre de points de la matrice est n_g, il crée un tableau de 2 n_g entiers, contenant les coordonnées *x* et *y* de tous les points de la matrice dans leur ordre d'*allumage*. Ainsi, pour le gris de niveau 5, on allumera 5 pixels de la matrice dans un ordre judicieux, de façon à lisser au mieux l'image en évitant de créer zébrures ou moirés. Ce matriçage a fait l'objet de nombreuses études fondamentales au cours des deux précédentes décennies **(5)**.

La plus grande partie de ce chapitre a été consacrée à l'art difficile du rendu des demi-teintes par des moyens binaires. On sait que la télévision opère autrement, puisque la demi-teinte y est représentée par l'amplitude du signal. Or la carte VGA permet de retrouver l'avantage de la télévision ; on la qualifie d'*analogique*, alors qu'elle est toujours numérique, mais pas binaire. Dans cette carte, on trouve trois décodeurs (convertisseurs numérique-analogiques, un par couleur) permettant d'obtenir 256 niveaux (d'où par combinaison 16 millions de couleurs). En rendant égales les sorties sur les trois codeurs, le programmeur disposerait de 256 niveaux de gris directement, sans l'artifice du matriçage, pour chaque point du masque-écran. Malheureusement, en général, on ne peut pas programmer directement ces décodeurs, on doit passer par le registre de palette qui ne contient que 16 entrées programmables en même temps. Cette limitation provient de la trop faible capacité de la mémoire video. Les cartes SuperVGA, dotées d'au moins un mégaoctet de mémoire, autorisent bien (entre autres) 256 types de gris dans la même image-écran. Le Macintosh offre les mêmes possibilités.

---

5 - Voir par exemple *Fondamentals of interactive computing graphics* par J.D. Foley et A. Van Dam, (Addison-Wesley).

Chapitre 9

EXECUTION DU PROGRAMME POSTSCRIPT

1 - IMPRESSION DE LA PAGE

On a décrit la majeure partie des opérateurs de PS et la façon de les présenter pour que l'interpréteur sache les exécuter. Rappelons encore (cf. page 9) que la description d'une page doit se terminer par

showpage

opérateur sans argument qui provoque la copie de la page virtuelle (en mémoire) sur la page réelle et l'éjecte de l'imprimante. Cet opérateur efface ensuite la page-mémoire et réinitialise la CTM. Il existe un opérateur voisin :

copypage

Beaucoup moins employé, **copypage** agit comme **showpage**, mais n'efface pas la mémoire. Il peut servir à des impressions partielles au cours de mises au point. Pour imprimer plusieurs exemplaires d'une page, il faut utiliser **showpage** après avoir initialisé la variable **#copies** au nombre désiré *n* :

#copies *n* **def**

Ni **copypage** ni **showpage** ne doivent être utilisés dans un fichier exporté comme on l'expliquera p. 51.

2 - CONVENTIONS

Il est conseillé d'ajouter à un programme PS un certain nombre de lignes en commentaire, lignes commençant donc par le symbole **%** . De plus, certains logiciels n'exécutant pas PS, mais le transmettant ou l'important, doivent en obtenir des informations. Par convention, ces informations sont placées en tête du fichier et précédées du symbole **%%**, sauf la première qui identifie le fichier et la version PS :

%!PS–Adobe

C'est en particulier à la réception du double symbole **%!** que les imprimantes mixtes se commutent automatiquement en mode PostScript. Beaucoup d'autres commentaires peuvent suivre : comme dans les fichiers TIFF, on peut indiquer l'auteur, la date, le nombre de pages, leur numéro ... Tout cela est en général facultatif, sauf la pseudo-instruction :

%%BoundingBox: x_0 y_0 x_1 y_1

qui permet à un logiciel importateur de connaître les dimensions du dessin. Les **entiers** x_0 y_0 x_1 y_1 donnent **en points** les coordonnées des coins extrêmes du dessin (sa diagonale). Elles permettent à ce logiciel de redimensionner la figure avec un **scale** pour l'ajuster à la surface qui lui a été réservée dans le document récepteur. Ces informations, ou **DSC**, seront appelées dans ce livre **directives** (on trouvera plus de détails dans l'annexe 6).

3 - COMPATIBILITE

Seules en principe les machines bénéficiant de la licence d'Adobe sont garanties être conformes à PostScript. Cependant, d'autres imprimantes acceptent ce langage de manière plus ou moins correcte. Leurs polices toutefois sont légèrement différentes de celles d'Adobe, qui sont brevetées.

A part les polices, l'auteur n'a pas noté de différence entre les unes et les autres machines. Parfois les images sont imprimées avec plus ou moins de succès. Ce n'est pas l'opérateur **image** qui paraît en cause, mais la réception de ces fichiers un peu trop volumineux. C'est pourquoi nous avons donné, pages 46-47, plusieurs méthodes pour associer ces fichiers à l'opérateur **image**.

On peut également trouver, spécialement sur Internet, des logiciels qui non seulement interprètent PostScript en représentant le dessin à l'écran, mais également l'adaptent à des imprimantes non PostScript (HPLaserjet ou à jet d'encre par exemple). Certains de ces logiciels sont de surcroît gratuits.

4 - TRAITEMENT DIRECT PAR L'IMPRIMANTE

La destinée première d'un fichier PS est de parvenir à une imprimante, une photocomposeuse, ou toute machine d'imprimerie acceptant ce langage. Cette machine se distingue des autres imprimantes par la présence d'un interpréteur de langage de description de page appelé RIP.

Un RIP *(raster image processor)* est une unité de calcul décodant les instructions du type tracé vectoriel pour en faire un dessin inscrit dans les cellules d'une mémoire, donc de type *bitmap*. L'imprimante étant matricielle, cet échantillonnage *(rastering)* est nécessaire, mais il est très important qu'il se fasse le plus tard possible, donc juste avant l'impression, si l'on veut conserver au dessin toutes ses qualités. Les dessins bitmap sont très dépendants des machines, difficiles à conserver ou à exporter, parce que malaisés à dilater ou à réduire alors que le dessin vectoriel joue avec les changements d'échelle sans problèmes ni perte de définition.

Le RIP crée dans une mémoire une image pixellisée de la page à imprimer. La taille de cette mémoire est très importante et intervient dans le prix du RIP. A raison d'un bit par point, de 600 points par pouce, l'image N&B d'une feuille de A4 exige plus de 4 mégaoctet (imprimante de bureau). Une photocomposeuse préparant 4 pages à la fois à raison de 1 200 points par pouce exigera plus de 64 mégaoctets en noir et blanc et 4 fois plus en quadrichromie. Sa mémoire est alors celle d'un disque dur et la durée de fabrication de la page (du cliché) peut être longue.

Le RIP est généralement constitué d'un processeur rapide, souvent de la série 68000 de Motorola, le logiciel interpréteur étant contenu dans des mémoires mortes. Ce processeur, les circuits annexes, les mémoires mortes et vives forment alors une unité de traitement – une carte – consacrée uniquement à PS. On trouve également des RIP purement logiciels exploitant les moyens de calcul d'un ordinateur de bureau. A l'origine, ces RIP étaient beaucoup moins rapides que ceux de la première formule **(2)**.

Le fichier PostScript peut être envoyé en bloc à l'imprimante PS, après passage, en général, par une file d'attente. Sous DOS, cet envoi se fait avec l'ordre **print** *fichier*. Sur Mac, on utilisera l'un des utilitaires cités dans la section suivante. Ou bien on l'envoie via Ghostscript sur une imprimante quelconque.

5 - TRAITEMENT INTERACTIF

Si, pour mettre au point un programme PS, on ne dispose pas d'un interpréteur comme Ghostscript ou Gsview, on peut l'envoyer sur une imprimante compatible PS et analyser le résultat. La consommation de papier, d'encre et l'usure du tambour peuvent alors devenir prohibitives. Aussi est-il préférable d'utiliser les propriétés interactives de l'interpréteur pour débusquer les erreurs **(1)**.

On supprimera alors l'opérateur **showpage** (pour raison d'économie) et on reliera imprimante et calculateur par un câble duplex. Sous Macintosh, la liaison AppleTalk convient et permet le dialogue. Sous DOS, il faut éviter la liaison Centronix, unidirectionnelle, et utiliser une liaison RS232, plus difficile à configurer et moins rapide, mais bidirectionnelle.

On enverra les instructions si possible une à une avec un logiciel approprié et on attendra la réponse de l'imprimante. En annexe, on donne un programme permettant une telle communication. Mais des logiciels commerciaux peuvent également être utilisés. Les réponses de l'interpréteur sont de types divers. Tout d'abord, dès la première instruction, l'interpréteur décline en général son identité et sa version. Puis, si l'on envoie l'instruction

true echo

la machine retournera au calculateur la copie de chaque instruction reçue. Ensuite, on pourra envoyer des instructions du type **(2)**

variable **==** ou **pstack**

qui provoquent, la première, le retour au calculateur de la valeur de la *variable* désignée et, la deuxième, le retour du contenu de la pile (sans l'effacer). Ce sont de précieux moyens de diagnostic. Enfin, en cas d'erreur, la machine envoie un message spécial répertorié dans la section 10. Les mêmes diagnostics sont fournis par Ghostscript.

Sur Macintosh, il faut soigneusement éviter l'encapsulation automatique par le *system* des fichiers envoyés, sinon on n'imprimera que le texte même du programme PS. Pour qu'il soit exécuté et non imprimé, il faut utiliser un *utilitaire* comme *SendPS* qui évite cette encapsulation. Les messages de l'interpréteur sont alors affichés à l'écran, mais ils sont plutôt fugitifs. L'utilitaire *Laser Writer Font Utility* (menu **Utilities**, puis **Download PS Files**), plus récent, est bien préférable : il range les messages d'erreurs dans un fichier (**PSLog**) que l'on pourra ensuite consulter.

---

1 - Voir également à ce sujet le dernier alinéa de la page suivante (§6).

2 - Ce n'est plus le cas actuellement (1998).

6 - IMPORTATION

Si l'interprétation directe, par imprimante interactive ou mieux par Ghostscript **(3)**, est toujours nécessaire pour la mise au point du dessin, la plupart des petits programmes PS sont plutôt destinés à être inclus dans un document plus général réalisé avec un logiciel de traitement de texte ou de mise en page.

Dans ce cas, il est nécessaire de vérifier que ledit logiciel *importe* bien les fichiers PS. Il vaut mieux également qu'il les *exporte*, c'est-à-dire qu'il sache créer des fichiers PS pour une imprimante de ce type. Sinon, il se contenterait de traduire le programme importé en image de points *(bitmap)*, ce qui dégrade généralement la qualité du dessin.

Lorsqu'un programme PS doit être importé dans un document, il faut prendre quelques **précautions** : en particulier, on ne doit pas déclencher l'impression prématurée d'une page non terminée (donc ni de **showpage** ni de **copypage**), ni réinitialiser la matrice CTM ou l'état graphique (opérateurs **initmatrix** et **initgraphics**), car on perturberait gravement le reste de la page. On proscrira de même tout opérateur faisant appel au matériel (incluant le terme **device**). Enfin, on laissera la pile vide et l'état graphique dans son état initial, en englobant tout le programme à importer entre **gsave** et **grestore**. Il est préférable également de placer le tout dans un dictionnaire privé.

Cependant, la plupart des logiciels importateurs se méfient des programmeurs et insèrent avant importation des instructions inhibant les opérateurs dangereux mentionnés ci-dessus (par exemple en n'autorisant pas au programme importé l'accès au **systemdict**, mais seulement à une copie réduite). De plus, ces logiciels placent avant importation une marque (**mark**) sur la pile et, au retour à la page normale, nettoient la pile jusqu'à ladite marque inclusivement (opérateur **cleartomark**).

On rappelle que les logiciels importateurs recherchent dans le programme importé certaines informations, lesquelles sont précédées du symbole **%%**. En particulier, l'importateur utilisera les paramètres de la directive **BoundingBox:** pour changer l'échelle du dessin de façon à ce que sa plus grande dimension s'ajuste dans le cadre qui lui a été réservé. Tout cela fait partie des spécifications DSC (annexe 6).

7 - EXPORTATION

On dit qu'un logiciel *exporte* PS s'il est conçu pour *produire* des fichiers décrivant des images de page en PostScript. Les meilleurs traitements de texte offrent cette possibilité, ainsi que les logiciels de CAO et presque tous ceux de PAO.

On pourra demander à ces logiciels d'*imprimer en différé*, c'est-à-dire d'écrire leurs résultats dans un *fichier* PostScript. On transmettra ensuite ce fichier par disquette ou par réseau à une imprimante compatible (sa *portabilité* est élevée). Il est inutile dans ce cas de travailler en mode interactif.

Le programme PS produit par ces logiciels commerciaux est souvent condensé ; les opérateurs y sont redéfinis (par ex. **S** pour **show**, **R** pour **rotate** ...) et ils utilisent un grand nombre de procédures spécifiques. La définition de ces procédures, des variables, des polices spécifiques ... est contenue dans un *fichier-prologue*, que le logiciel envoie à l'imprimante avant la première description de page. Il ne renouvelle pas cet envoi à chaque page, les définitions restant en mémoire virtuelle. Toute impression d'un fichier PS produite par un logiciel de ce genre doit être précédée de l'envoi à l'imprimante du prologue associé (par ex. le fichier *LaserPrep* sous Macintosh).

Noter un emploi inattendu de PS : la transmission par réseaux de documents prêts à être imprimés, voire de livres. De tels documents incluent des caractères de mise en page, non strictement ASCII. Leur transition par les nœuds du réseau peut gravement les altérer. Mais c'est surtout la réception qui déforme le plus ces textes, les imprimantes n'ayant pas forcément les mêmes tables de codage que l'émetteur. L'envoi de tels documents après encapsulation dans un fichier PostScript est une méthode garantissant leur transmission sans défaut, à condition qu'on ait respecté les normes PS, en particulier qu'on se soit strictement limité aux caractères ASCII 32...127.

8 - SUPERPOSITION

On peut avoir affaire à un logiciel d'édition qui exporte PS sans l'importer, ou bien qui l'importe mal. Voici deux méthodes *artisanales* qui permettent malgré tout d'importer un fichier – disons un dessin – dans le document principal.

D'abord, le fichier-dessin exigera une attention soutenue : suppression des opérateurs dangereux (cf. § 6), encadrement par le couple **gsave ... grestore** et mise en place du dessin en coordonnées absolues. Le rectangle réservé pour lui dans le document ayant pour origine x_0 , y_0 (mesurée par rapport au coin bas gauche du papier), pour largeur l et pour hauteur h, on fera suivre le **gsave** initial des instructions

---

3 - *Ghostscript* d'Aladdin Entreprises, *Ghostview* et *Gsview* sont des outils extrêment précieux pour la mise au point d'un programme PS. Se reporter à l'annexe 8.

x_0 y_0 **translate** a b **scale**

avec $a = l/L$ et $b = h/H$, L et H étant les dimensions du dessin dans le fichier initial. On tiendra compte des unités. Si tout est mesuré en millimètres, on n'oubliera pas de multiplier l'échelle par 2.835 en tête du fichier.

Double passage : avec une imprimante de bureau, on peut imprimer le texte sur une page vierge, avec les réservations adéquates pour l'illustration, puis on réalimentera l'imprimante avec la même feuille de papier pour y superposer le dessin. Les deux fichiers comporteront chacun un **showpage.** Ce procédé n'est pas toujours possible (surtout dans le cas de clichés).

Concaténation : on pourra également fusionner (concaténer) les deux fichiers PS, l'un produisant du texte, l'autre du dessin, par exemple, sous DOS, avec l'ordre " **copy** *fic1 + fic2 fic3* ". S'il n'imprime qu'une seule page, le fichier de texte précédera celui de dessin, mais ne comportera pas de **showpage**. A sa suite, on pourra inclure les instructions suivantes :

grestoreall clear initmatrix

On supprimera ce qui paraît inutile en tête du dessin, comme les directives commençant par **%!** et **%%.** Si le texte est multipage, le fichier dessin doit être inclus à la bonne page, mais il faut l'englober non seulement entre **gsave** et **grestore** mais aussi dans un dictionnaire privé, pour éviter un conflit entre des définitions. Ce sont des procédés d'emploi délicat (ne pas oublier l'adjonction du prologue).

9 - REACTION AUX ERREURS DE PROGRAMME

On a vu comment l'utilisation interactive d'une imprimante permet d'accéder aux erreurs de programmation (liste ci-après). Ghostscript fournit les mêmes diagnostics.

Quand le fichier comporte une erreur, les interpréteurs réagissent de différentes façons. Certains n'impriment rien sur la page, ou bien un message d'erreur. Dans le meilleur des cas, ils impriment tout le dessin précédant l'instruction erronée et tentent d'interpréter la suite du programme. Cette partie du dessin sera en général incorrecte et l'opérateur localisera ainsi rapidement l'instruction erronée.

10 - LISTE DES ERREURS D'EXECUTION

Voici la liste des messages d'erreur émis par l'interpréteur.

dictfull	plus de place dans le dictionnaire (placer sur la pile un dictionnaire de taille suffisante, cf. page 34).
dictstackoverflow	pile des dictionnaires saturée, trop de dictionnaires en service (nombreux **begin** sans **end**).
dictstackunderflow	trop de dictionnaires, trop de **end**.
execstackoverflow	pile d'exécution saturée (trop de boucles imbriquées).
interrupt	interruption externe (appui sur une touche d'arrêt).
invalidacess	tentative de violer une interdiction d'accès (fichier, tableau, dictionnaire).
invalidexit	**exit** hors d'une boucle.
invalidfileacess	accès à un fichier non autorisé.
invalidfont	nom de police ou dictionnaire invalide.
invalidrestore	**restore** invalide.
ioerror	erreur d'entrée-sortie.
limitcheck	dépassement d'une limite allouée dans l'installation.
nocurrentpoint	aucun point courant n'est défini.
rangecheck	un opérande est hors des limites autorisées.
stackoverflow	débordement de la pile.
stackunderflow	manque d'opérandes dans la pile.
syntaxerror	erreur de syntaxe dans les instructions **(4)**.
timeout	dépassement du temps alloué.
typecheck	opérande d'un type incorrect.
undefined	nom inconnu.
undefinedfilename	nom de fichier non défini.
undefinedresult	résultat sans signification.
unmatchedmark	marque **[** absente de la pile.
unregistered	erreur interne.
VMerror	mémoire virtuelle saturée.

Erreurs courantes dont il faut connaître la signification : **nocurrentpoint, undefined, stackunderflow, rangecheck, typecheck, syntaxerror**.

4 - Si, pour cette erreur, l'interpréteur indique "**Offending command : file**", vérifier si le fichier est bien constitué : accolades, crochets, parenthèses appariés, **ifelse** équilibrés...

Chapitre 10

LISTE DES MOTS CLES

Dans ce chapitre, on reprendra la majorité des opérateurs de PostScript pour en donner un résumé. Quand on indiquera "*dépile un objet*", il faudra comprendre "*retire et détruit l'objet situé en haut de la pile*". La flèche → sépare les états de la pile avant et après l'action de l'opérateur. Les pointillés symbolisent un contenu quelconque. Si ces pointillés subsistent après opération, c'est que l'opérateur n'a pas modifié ce contenu.

abs n_1 **abs** → $|n_1|$

dépile le nombre n_1, empile sa valeur absolue **(1)**.

add n_1 n_2 **add** → n_3

dépile les nombres n_1 et n_2, empile leur somme **(1)**.

aload *tab* **aload** → a_0 a_1 a_2 .... a_{n-1} *tab*

dépile le tableau *tab*, place sur la pile les éléments le constituant (indice supérieur en haut de pile), puis empile le tableau *tab* par dessus (page 26). Généralement, on enlève le dernier argument avec **pop**.

anchorsearch

ch témoin **anchorsearch** → *ch2 ch1* **true**
→ *ch* **false**

recherche si la chaîne *témoin* constitue le début de la chaîne *ch*. Si oui, coupe *ch* en deux, empile ch_2, partie finale de *ch*, puis la partie initiale ch_1 identique à *témoin*, et enfin **true.** Sinon, empile *ch*, puis **false** (page 25).

and $bool_1$ $bool_2$ **and** → $bool_3$
n_1 n_2 **and** → n_3

remplace les deux opérandes par leur intersection logique. Si ce sont des entiers, le résultat est l'image de leur intersection bit à bit (pages 23 et 31).

arc ... *x y r* ang_1 ang_2 **arc** → ...

ajoute au chemin d'abord un segment de droite, puis un arc de cercle centré en *x,y*, de rayon *r*, compris entre les angles ang_1 et ang_2. Le cercle est toujours décrit dans le sens direct à partir de ang_1 (si $ang_2 < ang_1$, PS ajoute 360° à ang_2, page 10).

arcn ... *x y r* ang_1 ang_2 **arcn** → ...

comme **arc**, mais l'arc de cercle est décrit dans le sens rétrograde, ou sens des horloges (page 10).

arcto
x_1 y_1 x_2 y_2 *r* **arcto** → xt_1 yt_1 xt_2 yt_2

ajoute au chemin éventuellement un segment, puis un arc de cercle de rayon *r*. Place sur la pile les coordonnées des deux points de tangence (page 11).

array *n* **array** → *tab*

crée et empile un tableau de *n* éléments, initialisés avec l'objet **null** (page 26).

ashow ... *l h ch* **ashow** → ...

imprime la chaîne *ch* comme **show**, mais ajoute un décalage supplémentaire entre caractères de *l* unités en largeur et *h* en hauteur (page 38).

astore a_0 a_1 ... a_{n-1} *tab* **astore** → *tab*
... /*tab* a_0 a_1 ... a_{n-1} *n* **array astore def**→ ... **(2)**

place *n* objets a_i dans le tableau *tab* de longueur *n*. La seconde instruction crée et initialise le tableau *tab* (page 26).

atan *num dénom* **atan** → *angle*

retire deux nombres, les remplace par la valeur en degrés (0 à 360) de l'angle dont le rapport *num/dénom* est la tangente, dans le quadrant correct.

awidthshow
... *l' h' car l h ch* **awidthshow** → ...

imprime la chaîne *ch* comme **ashow**, mais ajoute un décalage supplémentaire de *l* ' en largeur et de *h*' en hauteur après chaque caractère *car* (page 38).

begin ... *dico* **begin** → ...

place *dico* sur la pile des dictionnaires et en fait le dictionnaire courant (page 34).

bind *proc* **bind** → *proc*

remplace la procédure *proc* par une traduction binaire (compilation). Attention, les variables **internes** à *proc* peuvent garder comme valeur celle prise au moment de l'exécution de **bind** (page 32).

---

1 - Si les opérandes sont de type entier, le résultat est entier (sauf s'il en dépasse les limites). Sinon, il est réel.

2 - la deuxième forme est identique à :
/*tab* [a_0, a_1, ... a_{n-1}] **def**

bitshift *n nb* **bitshift** → *n'*

décale l'entier *n* de *nb* bits vers la gauche si *n*>0, vers la droite si *n*<0 (page 31).

bytesavailable *fichier* **bytesavailable** → *n*

donne le nombre d'octets non encore lus dans le fichier *(n*=−1 si la fin du fichier est atteinte)

ceiling *r* **ceiling** → *n*

remplace le nombre *r* par l'entier *n* égal ou immédiatement supérieur (page 29) **(1)**.

charpath ... *ch booléen* **charpath** → ...

ajoute au chemin courant le contour de la chaîne *ch* (page 38).

clear ... **clear** → ∅

vide entièrement la pile (page 33).

cleartomark ... **[** a_1 ... a_n **cleartomark** → ...

vide la pile à partir du haut jusqu'à la marque comprise (pages 33 et 51).

clip ... **clip** → ...

détermine un nouveau pochoir, égal à l'intersection entre l'ancien et la surface délimitée par le chemin courant (règle *des entrelacs*, pages 14-15).

clippath ... **clippath** → ...

établit comme chemin courant la frontière du pochoir en cours (page 15).

closepath ... **closepath** → ...

ferme le chemin courant avec un segment rejoignant le point initial (page 12).

concat ... *matrice* **concat** → ...

modifie la CTM en la multipliant matriciellement avec *matrice* (page 20).

copy a_1 ... a_n *n* **copy** → a_1 ... *an* a_1 ... a_n

ajoute sur la pile la copie de ses *n* premiers objets (les plus hauts, page 33).

tab_1 tab_2 **copy** → tab_3

tab_1 , tab_2 étant des tableaux, chaînes ou dictionnaires, substitue les *n* éléments de tab_1 aux *n* premiers éléments de tab_2 (si la longueur de tab_2 *est* suffisante), empile tab_3, la partie de tab_2 qui sera modifiée par la copie. L'objet composite tab_2 conserve sa taille initiale (pages 24-26).

copypage ... **copypage** → ...

provoque l'impression de la page-mémoire sur la page-papier, éjecte celle-ci, mais n'efface pas la page-mémoire et ne modifie pas les piles (page 49).

cos *angle* **cos** → *réel*

remplace l'*angle*, exprimé en degrés, par son cosinus.

count a_1 ... a_n **count** → a_1 ... a_n *n*

compte le nombre d'éléments de la pile sans les retirer et empile ce nombre (page 33).

counttomark

... **[** a_1 ... a_n **counttomark**→... **[** a_1 ... a_n *n*

compte sans les retirer le nombre d'éléments dans la pile au-dessus de la marque et empile ce nombre.

current*variable*

... **current***variable* → ... obj_1 ... obj_n

différents opérateurs permettent d'empiler et de connaître la valeur en cours de certains paramètres, comme **currentdash** (type de pointillé), **currentdevice** (imprimante), **currentdict** (dictionnaire en cours), **currentfile** (fichier, p 34, 46), **currentflat** (approximation polygonale), **currentfont** (police, p. 42), **currentgray** (valeur de gris fixée par **setgray**), **currenthsbcolor** (couleurs, 3 composantes HSB), **currentlinecap** (extrémités des lignes), **currentlinejoin** (forme des coudes), **currentlinewidth** (épaisseur du trait), **currentmatrix** (matrice courante), **currentmiterlimit** (limitation des coudes effilés), **currentpoint** (point courant, p. 17), **currentrgbcolor** (couleur RVB en cours), **currentscreen** (paramètres de tramage), **currenttransfer** (donne la loi de grisé, p. 48).

curveto ... x_1 y_1 x_2 y_2 x_3 y_3 **curveto** → ...

ajoute au chemin une courbe de Bézier commençant au point courant, guidée par les points { x_1 , y_1}, { x_2 , y_2} et se terminant en { x_3 , y_3} qui devient point courant (page 12).

cvi ch_1 *(ou n')* **cvi** → *n*

convertit la chaîne *ch* ou le nombre *n'* en un entier *n* (avec troncature) (page 30).

cvr *ch* (ou *n)* **cvr** → *r*

convertit la chaîne *ch* ou l'entier *n* en un réel *r* (p. 30).

cvrs *n base ch* **cvrs** → ch_1

convertit le nombre *n* en une chaîne le représentant en base *base*. La chaîne *ch* doit être suffisamment longue pour contenir ch_1 (page 30).

cvs *objet ch* **cvs** → *ssch*

convertit un *objet* quelconque, place sa représentation dans *ch* et empile à leur place la sous-chaîne *ssch* représentant l'objet (page 30).

def ... *lnom objet* **def** → ...

définit *nom* comme une nouvelle variable, si elle n'est pas déjà connue, et lui affecte l'*objet* cité (nombre, tableau, chaîne, procédure, pages 24, 30, 31).

definefont *lnom dico* **definefont** → *dico*

définit le dictionnaire *dico* comme une police et lui donne le *nom* cité (page 41).

dict *n* **dict** → dictionnaire

crée un dictionnaire vide de capacité *n* et le place sur la pile (page 34).

div n_1 n_2 **div** → *réel*

dépile les nombres n_1 et n_2 , les remplace par leur quotient *réel*.

dtransform *dx dy* [*matrice*] **dtransform** → *dx' dy'*

donne les longueurs *dx', dy'* égales en unités machine aux longueurs *dx, dy* (les translations sont ignorées). Si la *matrice* est omise, la CTM la remplace.

dup *objet* **dup** → *objet objet*

duplique le premier *objet* de la pile (page 33).

echo ... *booléen* **echo** → ...

si *booléen* = **true**, provoque le renvoi par l'imprimante du texte de toutes les instructions reçues ultérieurement en communication interactive. Si *booléen* = **false** , met fin à ce comportement (p. 50)

end ... **end** → ...

retire de la pile le dictionnaire courant (annule un **begin**, page 34).

eoclip ... **eoclip** → ...

comme **clip**, mais avec la règle des frontières paires-impaires (page 15).

eofill ... **eofill** → ...

remplit la surface contenue sous le chemin courant par le grisé ou la couleur en cours, en utilisant la règle pair-impair (page 14).

eq obj_1 obj_2 **eq** → *booléen*

dépile deux objets, les compare et les remplace par **true** ou **false** selon que leurs valeurs sont identiques ou non (page 23).

erasepage ... **erasepage** → ...

efface la page-mémoire en cours et la remet à la couleur du fond de page. N'affecte pas l'état graphique. Exécuté automatiquement par **showpage**.

exch obj_1 obj_2 **exch** → obj_2 obj_1

permute les deux objets du haut de pile (page 33).

exec { *proc* } **exec** → résultat de *proc*

exécute la procédure placée en argument (page 32).

exit ... **exit** → ...

quitte une boucle, passe à la boucle plus externe (page 32).

exp *y x* **exp** → *réel*

calcule *réel* = y^x , *x* et *y* entiers ou réels, mais *x* entier si $y < 0$ (page 29).

false ... **false** → ... **false**

empile le booléen **false** .

file *nom code* **file** → *fichier*

crée un fichier dont la chaîne *nom* donne le nom et dont l'accès n'est autorisé qu'en écriture si *code* = **(w)** , ou en lecture si *code* = **(r)** (page 34).

fill ... **fill** → ...

remplit la surface sous le chemin en cours par la couleur ou le grisé en cours (règle des *entrelacs* non-nuls). Détruit le chemin en cours (pages 13-14).

findfont *nom* **findfont** → *police*

place un dictionnaire-police sur la pile (page 37).

flattenpath ... **flattenpath** → ...

remplace les courbes du chemin courant par son tracé polygonal (page 39).

floor *n'* **floor** → *n*

remplace le nombre *n'*, réel ou entier, par l'entier égal ou juste inférieur (page 29).

flush ... **flush** → ...

provoque la sortie immédiate des valeurs produites par l'interpréteur (vidage des tampons). L'opérateur **flushfile** fait de même avec les autres fichiers (p. 32).

for i_0 *pas* i_n { *proc* } **for** → *i* (*n* fois)

pour chaque valeur d'un indice *i* variant de i_0 à i_n inclus (entiers ou réels) par pas égaux à *pas* , place *i* sur la pile, puis exécute la procédure *proc* (page 26).

forall ... *composite* { *proc* } **forall** → *élément (n fois)*

empile successivement chacun des éléments du *composite* (tableau, dictionnaire, chaîne), en commençant à l'indice 0, puis exécute à chaque fois *proc*. Si le *composite* est une chaîne, les éléments empilés seront les numéros des caractères (0 .. 255). Si le composite est vide, pas d'exécution (page 27).

ge obj_1 obj_2 **ge** → *booléen*

dépile deux objets (tous deux nombres ou tous deux chaînes), les remplace par **true** si $obj_1 \geq obj_2$, sinon par **false** (page 23).

get *composite indice* **get** → *élément*

copie sur la pile l'élément du *composite* (tableau ou chaîne) dont l'*indice* est cité (les indices commencent à zéro). Agit de même avec un dictionnaire si, au lieu de l'*indice*, on donne la *clé* (pages 25-26).

getinterval *composite ind nb* **getinterval** → *sous-composite*

crée un nouveau tableau ou une nouvelle chaîne contenant les éléments du *composite* dont l'indice varie de *ind* à *ind* + *nb* – 1 (pages 24-26).

grestore ... **grestore** → ...

efface l'état graphique en cours et le remplace par celui sauvegardé par le **gsave** précédent (page 21).

grestoreall ... **grestoreall** → ...

retire de la pile des états tous les états graphiques sauvegardés depuis le début de la session ou depuis le dernier **save** (page 52).

gsave ... **gsave** → ...

sauvegarde l'état graphique en cours dans la pile des états graphiques (page 21).

gt obj_1 obj_2 **gt** → *booléen*

dépile deux objets, les compare et les remplace par **true** si $obj_1 > obj_2$ ou, dans le cas contraire, par **false** (page 23).

idiv n_1 n_2 **idiv** → n_3

division entière, avec troncature du résultat (n_1 , n_2 , n_3 sont des entiers, page 29).

if ... *booléen* **{** *proc* **}** **if** → ...

dépile les deux opérandes, exécute *proc* si le *booléen* est égal à **true** (page 28).

ifelse
... *booléen* **{** $proc_1$ **}** **{** $proc_2$ **}** **ifelse** → ...

dépile 3 opérandes, exécute $proc_1$ si le *booléen* est **true**, sinon $proc_2$ (page 28).

image ... *l h nb* **[***matrice***]** **{***proc***}** **image** → ...

imprime une image formée de *l*×*h* pixels, codés chacun sur *nb* bits, ces bits étant extraits des octets générés par la procédure *proc*. Cette procédure peut se réduire à une chaîne hexadécimale. La matrice transforme l'image décrite par *proc* en une image de taille 1×1 unité (pages 46-47).

imagemask
... *l h bool* **[** *matrice* **]** **{** *proc* **}** **imagemask** → ...

comme **image**, mais l'image ne peut être que binaire *(nb*=1). Si *bool*=**false**, les bits 0 donnent des pixels imprimés (en noir) et les bits 1 ne donnent lieu à aucune impression (fond de page préservé). C'est l'inverse si *bool*=**true** (page 47).

index a_n ... a_0 *n* **index** → a_n ... a_0 a_n

dépile l'indice *n*, cherche le (*n*+1)ième objet sur la pile et le duplique. Attention : l'objet en haut de la pile a pour rang 0 (page 33).

initclip ... **initclip** → ...

retour au pochoir initial (celui du système, page 15).

initgraphics ... **initgraphics** → ...

retour à l'état graphique du système. A éviter, surtout dans fichier importé.

itransform *x y* **itransform** → *X Y*

transforme les coordonnées-machine *x* et *y* en leurs valeurs *X* et *Y* dans le repère en cours (page 21).

known *dico nom* **known** → *booléen*

empile **true** (ou **false**) selon que *nom* est (ou n'est pas) une entrée du dictionnaire *dico*.

kshow **{** *proc* **}** *chaîne* **kshow** → ...

imprime une *chaîne* comme **show**, mais exécute *proc* entre chaque caractère (en particulier, possibilité de *crénage*, page 38).

le obj_1 obj_2 **le** → *booléen*

retire deux objets de la pile, les compare et les remplace par **true** si $obj_1 \leq obj_2$ ou, dans le cas contraire, par **false** (page 23).

length *composite* **length** → *n*

dépile le *composite* (tableau, chaîne, dictionnaire), le remplace par *n*, nombre d'éléments actuellement contenus (pages 24, 26, 34).

lineto ... *x y* **lineto** → ...

ajoute au chemin un segment allant au point *x*, *y*, qui devient point courant (page 10).

ln *n* **ln** → *réel*

remplace le nombre *n* (entier ou réel) par son logarithme népérien (page 29).

load **/***nom* **load** → *valeur*

cherche *nom* dans les dictionnaires, met sa *valeur* sur la pile si ce *nom* existe, sinon provoque l'erreur **undefined** (page 30).

log *n* **log** → *réel*

remplace *n* *(n* > 0, entier ou réel) par son logarithme décimal *(réel)* (page 29).

loop ... **{** *proc* **}** **loop** → ...

répète indéfiniment la procédure *proc*. Sortie avec *bool* **{exit}** **if** (page 26).

lt obj_1 obj_2 **lt** → *booléen*

dépile deux objets, les compare et les remplace par **true** si $obj_1 < obj_2$ ou, dans le cas contraire, par **false** .

mark ... **mark** → ... **[**

place une marque en haut de la pile (page 33).

maxlength *dico* **maxlength** → *n*

donne la taille totale du dictionnaire *dico* , définie au moment de sa création (page 34).

mod n_1 n_2 **mod** → n_3

donne la valeur de n_1 modulo n_2 (reste de la division entière de n_1 par n_2, page 29).

moveto ... *x y* **moveto** → ...

commence un (sous-)chemin en { *x* , *y* }, qui devient point courant (page 10).

mul n_1 n_2 **mul** → n_3

dépile n_1 et n_2 , les remplace par leur produit n_3 .

ne obj_1 obj_2 **ne** → *booléen*

dépile deux objets, les compare et les remplace par **true** si ces deux objets sont différents ou, s'ils sont égaux, par **false** (page 23).

neg n_1 **neg** → n_2

dépile le nombre n_1 et le remplace par son opposé $n_2 = -n_1$ (page 29)

newpath ... **newpath** → ...

détruit le chemin courant (page 17, en note).

not n_1 (ou *booléen*) **not** → n_2 (ou $bool_2$)

dépile l'argument, le remplace par son complément binaire (pages 23 et 31).

null ... **null** → ... **null**

place l'objet **null** sur la pile (page 4).

or obj_1 obj_2 **or** → *booléen*

dépile deux objets, les compare et les remplace par leur réunion logique (si objets booléens) ou binaire (si objets entiers) (pages 23 et 31).

pathbox ... **pathbox** → ... x_0 y_0 x_1 y_1

empile les coordonnées de la *boîte* contenant le chemin en cours (page 37).

pathforall

... **{***promv***}** **{***proli***}** **{***procb***}** **{***proclo***}** **pathforall** → ...

retire les 4 procédures opérandes, empile les coordonnées successives des points du chemin en cours, puis, pour chacun d'eux, exécute l'une des procédures, celle en rapport avec l'opérateur l'ayant tracé (page 39).

pop ... *objet* **pop** → ...

dépile le premier *objet* et le rejette (page 33).

print ... *chaîne* **print** → ...

envoie la *chaîne* dans le fichier de sortie (et non sur l'imprimante).

pstack obj_1 ... obj_n **pstack** → obj_1 ... obj_n

copie dans le fichier de sortie tous les objets de la pile, sans effacer celle-ci (pages 32, 50).

put *composite ind objet* **put** → ...

remplace dans le *composite* (chaîne, tableau, dictionnaire) l'élément d'indice *ind* par l'*objet* placé en argument. S'il s'agit d'un dictionnaire, l'indice doit être remplacé par le nom de la clé souhaitée (pp. 25-26).

putinterval

composite ind séquence **putinterval** → *sous-composite*

remplace dans le *composite* (tableau ou chaîne seulement) la suite d'éléments commençant à l'indice *ind* par la *séquenc*e (sous-chaîne, sous-tableau) en argument. Les tailles doivent être compatibles (page 24).

quit ... **quit** → ...

quitte l'interpréteur (fin de la session de travail, p. 32).

rand ... **rand** → ... *n*

fournit un nombre entier *n* au hasard, $0 \le n \le 2^{31}-1$. La séquence produite peut être rendue reproductible par **srand** (page 29).

rcurveto

... dx_1 dy_1 dx_2 dx_3 dy_3 **rcurveto** → ...

ajoute au chemin une courbe de Bézier commençant au point courant, comme **curveto**, mais avec des arguments relatifs au point courant M_0 (page 12).

read *fichier* **read** → *numcar booléen*

lit le premier octet disponible dans le *fichier* cité, empile sa valeur numérique, puis **true**. Si la fin du fichier est atteinte, empile seulement **false** (page 34).

readhexstring

fichier ch **readhexstring** → *ssch booléen*

lit des couples de caractères dans *fichier*. Ne doit rencontrer que les caractères **0..9**, **A..F** ou **a..f**, en fait des nombres hexadécimaux dont il remplit la chaîne *ch* , puis l'empile et empile **true**. Si la fin de *fichier* est atteinte avant la fin de *ch*, retourne la sous-chaîne ainsi remplie *ssch*, puis **false** (page 34).

readline *fichier ch* **readline** → *ssch booléen*

lit *fichier*, remplit la chaîne *ch* jusqu'à rencontre du caractère *fin-de-ligne*. Empile *ssch*, sous-chaîne remplie de caractères, puis **true**. Empile **false** si on rencontre la fin du fichier avant une *fin-de-ligne*. Erreur **rangecheck** si *ch* est trop petite (page 34).

readstring

fichier ch **readstring** → *ssch booléen*

comme **readhexstring**, mais les caractères sont formés d'un nombre 0..255 ; peut être utilisé avec l'opérateur **image** (page 34).

repeat ... *n* **{** *proc* **}** **repeat** → ...

dépile les deux opérandes et exécute *n* fois la procédure *proc* (page 26).

rlineto ... *dx dy* **rlineto** → ...

dépile deux opérandes, ajoute au chemin un segment dont la longueur a pour projections *dx* et *dy* (page 10).

rmoveto ... *dx dy* **rmoveto** → ...

initialise un nouveau (sous-)chemin au point situé à {*dx, dy*} du point courant (page 10).

roll

$a_n \ldots a_1\ n\ j$ **roll** → $a_j \ldots a_1\ a_n \ldots a_{j+1}$

prend les *j* premiers octets (modulo *n*) de la pile et les place derrière le *n*ième (page 33).

rotate ... α **rotate** → ...

fait tourner le système de coordonnées de α degrés dans le sens direct (page 19).

round *m* **round** → *n*

dépile le nombre *m* et le remplace par l'entier *n* le plus proche (page 29).

run ... *variable-fichier* **run** → ...
... **(***nom-de-fichier***)** **run** → ...

provoque l'exécution du fichier cité en opérande (emploi peu courant, sauf avec les interpréteurs-écran).

scale ... r_x r_y **scale** → ...

multiplie les échelles par r_x en *x* et r_y en *y* (pages 19-20).

scalefont *police n* **scalefont** → *police'*

dépile la *police*-argument, lui donne le corps *n* (réel) et la remet sur la pile (pages 37, 41, 42).

search

ch témoin **search** → ch_2 *témoin* ch_1 **true**

Recherche la sous-chaîne *témoin* dans la chaîne *ch*. Si trouvée, divise *ch* en trois. Empile le segment de *ch* qui suit *témoin*, puis *témoin*, puis le segment précédant *témoin* et enfin **true**. Si insuccès, empile *ch*, puis **false** (page 25).

setdash ... *tab dec* **setdash** → ...

met en service le motif du pointillé décrit par le tableau *tab* et décalé de *dec* (page 18).

setflat ... *n* **setflat** → ...

une courbe étant assimilée à une suite de segments (polygone), fixe la tolérance maximale entre la position des points dans l'une et l'autre description.

setfont ... *police* **setfont** → ...

rend active la *police* en argument (page 37).

setgray ... *n* **setgray** → ...

fixe le niveau de gris *n* , $0 \le n \le 1$, $1 \Leftrightarrow$ blanc $\Leftrightarrow$ fond de page (page 19).

sethsbcolor ... *h s b* **sethsbcolor** → ...

fixe les paramètres de couleur pour les opérateurs utilisant le modèle HSB (teinte, pureté, intensité, $0 \le (h, s, b) \le 1$, page 19).

setlinecap ... *n* **setlinecap** → ...

définit la forme des extrémités de traits: 0 = abrupte, 1 = arrondie, 2 = allongée (page 18).

setlinejoin ... *n* **setlinejoin** → ...

définit la forme des jonctions entre traits. *n* = 0, 1 ou 2 (2: chanfrein, page 18).

setlinewidth ... *n* **setlinewidth** → ...

fixe l'épaisseur *n* (réel) des traits en unités courantes (page 18).

setmiterlimit ... *n* **setmiterlimit** → ...

impose une limite au développement des arêtes (page 18).

setrgbcolor ... *r v b* **setrgbcolor** → ...

fixe la proportion *r, v, b* (0...1) de rouge, vert, bleu dans la couleur en cours (page 19).

setscreen ... *lin incl* **{** *proc* **}** **setscreen** → ...

fixe les paramètres du rendu des demi-teintes par tramé : linéature, inclinaison en degrés et procédure donnant l'ordre dans lequel sont imprimés les points-machine pour former un pixel d'une teinte donnée (emploi délicat, pages 21 et 48).

settransfer ... **{** *proc* **}** **settransfer** → ...

modifie la loi de progression des gris dans **setgray** (page 48) ou celle de l'intensité des couleurs.

show ... *chaîne* **show** → ...

imprime la *chaîne* à partir du point courant, qu'il déplace de la longueur imprimée (page 38).

showpage ... **showpage** → ...

transfère sur le papier la page-mémoire. Ejecte la feuille de papier, efface la page-mémoire (pp. 9, 49).

sin *angle* **sin** → *r*

dépile *angle*, le remplace par son sinus *r* (page 29).

sqrt *n* **sqrt** → *r*

dépile le nombre *n* (positif ou nul), le remplace par sa racine carrée (page 29).

srand ... *n* **srand** → ...

réinitialise le générateur de nombres aléatoires *(n* entier). Permet d'obtenir par **rand** des séquences reproductibles avec des valeurs *n* identiques (p. 29).

StandardEncoding

... **StandardEncoding** → *tableau de codage*

empile la table de codage normale (page 42).

store ... **/***nom valeur* **store** → ...

crée la variable *nom* si elle n'existe pas et lui affecte la *valeur* donnée en argument (page 30).

string *n* **string** → *chaîne*

crée et empile une chaîne de longueur *n*. En général, il faut lui donner un nom avec **def** (page 24).

stringwidth *ch* **stringwidth** → *l h*

empile l'encombrement de la chaîne *ch* en longueur *(l)* et en hauteur *(h)* (page 37) .

stroke ... **stroke** → ...

encre le chemin courant avec les caractéristiques graphiques de l'état graphique en cours, puis détruit le chemin courant (pages 9, 13).

sub n_1 n_2 **sub** → n_3

dépile deux nombres, les remplace par leur différence algébrique.

transform ... *x y* **translate** → ... *x' y'*

transforme les coordonnées *x y* (utilisateur) en coordonnées machine (*x' y'*) (page 21).

translate ... *x y* **translate** → ...

dépile les deux opérandes, modifie la CTM. Fixe l'origine du nouveau repère des coordonnées au point *x, y* ; (le point de coordonnées {*x*,*y*} dans l'ancien système devient point {0,0} dans le nouveau (page 19).

true ... **true** → ... **true**

empile le booléen **true**.

truncate n_1 **truncate** → n_2

réduit le nombre n_1 à l'entier inférieur le plus proche n_2 (page 29 ; cf. note 1).

userdict **userdict** → *dictionnaire*

empile le dictionnaire **userdict** (sur la pile opérationnelle, page 34).

version **version** → *chaîne*

donne la version du langage et celle de l'interpréteur.

widthshow

... *dx dy numcar ch* **widthshow** → ...

imprime la chaîne *ch* comme **show**, mais en ajoutant les décalages *dx* et *dy* après chaque caractère de numéro *numcar* (page 38).

write ... *fichier numcar* **write** → ...

écrit le caractère de numéro *numcar* dans le *fichier*. Il existe deux autres opérateurs pour écrire dans un fichier, **writehexstring** et **writestring** (page 34).

xor $bool_1$ $bool_2$ **xor** → $bool_3$

n_1 n_2 **xor** → n_3

dépile deux opérandes, les remplace par leur réunion exclusive. Voir page 23 si les opérandes sont booléens, page 31 s'ils sont entiers.

[... **[** → ... **[**

place sur la pile l'objet appelé *marque* (page 33).

]... ... **[** a_0 ... a_n **]** → ... *tableau*

dépile tous les objets précédant la marque **[**, puis cette marque elle-même et fait des objets un tableau qu'il empile (page 26).

= ou **==** ... *objet* **==** → ...

dépile *objet*, envoie dans le fichier de sortie une chaîne représentant sa valeur (page 50).

Opérateurs PS (niveau 1) non étudiés :

banddevice	makefont	setcharwidth
cachestatus	matrix	setpacking
closefile	noaccess	stack
concatmatrix	nulldevice	start
countdictstack	packedarray	status
countexecstack	prompt	stop
cvlit	rcheck	stopped
defaultmatrix	readonly	strokepath
dictstack	renderbands	token
errordict	resetfile	type
execstack	restore	usertime
executeonly	reversepath	version
FontDirectory	rrand	vmstatus
framedevice	save	wcheck
identmatrix	setcachedevice	where
idtransform	setcachelimit	xcheck
invertmatrix	setcacheparams	

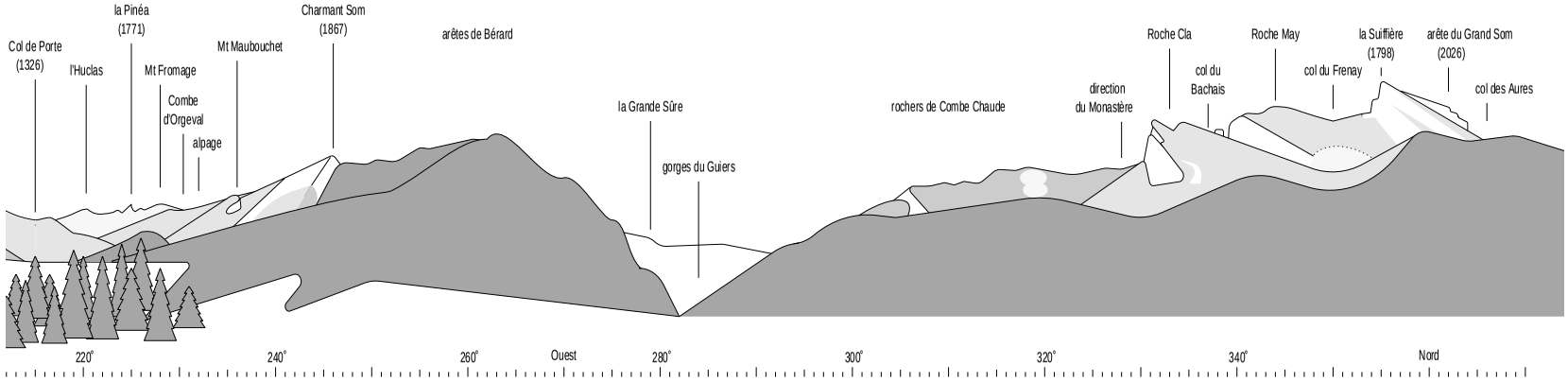
Parmi les opérateurs non étudiés, notons

- **type** : précédé du nom d'une variable, empile son type, ce qui en permettra le traitement judicieux ;

- **closefile** : ferme le fichier dont le pointeur est donné en argument ;

- **usertime** : empile le temps mis par l'interpréteur pour exécuter le programme ;

- **version** : empile le numéro de la version PS de l'interpréteur.



Un exemple de dessin PostScript : panorama partiel de St Pierre de Chartreuse.

ANNEXES

Annexe 1

PROGRAMME D'IMPRESSION INTERACTIVE

Le programme suivant, en Basic, permet d'envoyer à partir d'un PC un fichier-programme PostScript à une imprimante compatible. Il envoie les lignes une à une et lit les messages d'erreur de la machine à chaque instruction. Seuls les gros fichiers d'image ne sont pas acceptés par ce programme à cause de la taille réduite des tampons de communication faisant appel au BIOS. Ne pas oublier de sélectionner l'entrée RS232 sur l'imprimante et d'en régler les paramètres en accord avec ceux du programme (ligne 15).

```
1 '      Programme de communication interactive avec imprimante  PS
2 '
10  DIM ch$(256), rep$(120), fich$(10), nom$(80), reper$(60)
15  CLS : OPEN "COM2: 4800,N,8,1, RB32767" FOR RANDOM AS #1
20  INPUT "Nom complet du répertoire des fichiers PS ? : ",  reper$
22                                          '   Reprise ici en fin de fichier
25  INPUT "Nom (+ suffixe) du fichier PS ? : ", fich$
30  nom$ = reper$ + "\" + fich$                          ' arrêt par RC ou Esc
35  IF LEN(fich$) < 2 THEN STOP
40  OPEN nom$ FOR INPUT AS #2
45  PRINT #1, "clear  true echo"  :   GOSUB 500' nettoyage pile, demande d'écho
50  '
55  WHILE NOT EOF(2)                                      ' boucle principale
60  LINE INPUT #2, ch$  :          GOSUB 500
65  INPUT ; "", a                                        ' attente frappe touche
70  WEND
75  CLOSE 2 :     GOTO 25                                ' fin d'envoi du fichier
500  PRINT #1, ch$ :        PRINT ch$                      ' Traitement chaîne
510  GOSUB 800 :            PRINT "-> "; rep$ ;
520  IF INSTR(rep$, "Error") = 0 THEN RETURN     ' si pas message d'erreur
530  PRINT #1, " pstack" :   GOSUB 800                    ' si message d'erreur
540  PRINT " Pile = " ;  RIGHT$(rep$, n - 7) ;
550  RETURN
600 '                                              Réception
800    FOR a = 0 TO 6 STEP .002: x = 25 * SIN(a) : NEXT a   ' boucle d'attente
810    n = LOC(1) :  IF n > 1 THEN rep$ = INPUT$(n, #1)     ' acquisition réponse
820    FOR i = 1 TO n - 5                                  ' nettoyage de la réponse
830    IF ASC(MID$(rep$, i, 1)) < 32 THEN MID$(rep$, i, 1) = " "
840    IF (MID$(rep$, i, 3) = "%%[") THEN MID$(rep$, i) = CHR$(13)' RC suppl. si erreur
850    NEXT i
860  RETURN
```

Bien sûr, on pourra récrire ce programme en C ou en Pascal, si l'on dispose du compilateur correspondant. Les réponses de l'imprimante pourront alors être mieux analysées et mieux présentées à l'écran, le Basic offrant peu de souplesse sur ce point. Rien ne s'oppose également à ce qu'on perfectionne ce programme, par exemple pour :

- passer du mode fichier au mode console et envoyer à l'imprimante soit des instructions directement frappées au clavier, soit les lignes du fichier ;

- corriger dans le fichier les instructions erronées.

99 F

Annexe 2

EXEMPLES DE PROGRAMMES PS

Impression d'un code-barre EAN 13

```
%!PS-Adobe-2.0
%%BoundingBox:  0  0  190  90
%%DocumentFonts: Courier
/Bbox { /y2  exch  def  /x2  exch  def  /y1  exch  def    /x1  exch  def  x1  y1  moveto
      x1  y2  lineto  x2  y2  lineto  x2  y1  lineto  closepath  stroke }  def
%  0  0  190  90  Bbox                          %  Ne rendre active cette instruction qu'en essai
2.8346  dup  scale     0  setlinecap            %  toutes coordonnées en mm
/codbar  13  string  def

%  les 3 lignes ci-dessous sont modifiables par l'utilisateur, pour donner la position
%  x,y  de l'origine, le grandissement horizontal g , la hauteur h des barres et le code.
%  On n'est pas obligé de donner le chiffre de contrôle (dernier chiffre).

/x  20 def   /y 10  def   /g 1 def   /h  20  def
/codebarre  (312345600789)  def    /prix  (99 F)  def          %  valeur du code (12 ou 13 chiffres) et du  texte associé

                    %   NE RIEN MODIFIER APRES CETTE LIGNE
/ch  59  string  def
/codc  (3211) (2221) (2122) (1411) (1132) (1231) (1114) (1312)  (1213) (3112)
     10  array  astore  def                                    % codage des 10 chiffres en code C
/codab  0  11  13  14  17  25  29  21  22  26    10  array  astore  def    % position des codes B dans la partie AB
/d  0.33  g  mul  def                                          %  période des barres
codebarre  codbar  copy  pop                                  % si le caractère de contrôle n'a pas été donné,
codbar  dup  12  get  48  lt                                  % calcul & ajout de ce caractère.
  { 0  1  11  { codbar exch get 48 sub } for   0  6  { exch  3 mul add exch  add }  repeat
   10 mod neg  58 add  12 exch put }   if
% obtention via codab du code (codex) du caractère externe  (1er chiffre), puis injection des 3 séparateurs dans ch.

/codex  codab  codbar  0  get  48  sub  get  store
0 (999) 27 (99999) 56 (999)    3 { ch 3 1 roll putinterval }  repeat
                                                  % D'abord, tout est mis en code C
1 1 12 { /i exch store codbar i get 48 sub codc exch get ch i 4 mul 1 sub
   i 6 gt {5 add } if   3 2 roll putinterval }  for          %  puis on effectue la symétrisation pour les codes B selon codex
1 1 6 { /i exch store  /j  i 4 mul 1 sub  store   codex i 6 sub bitshift 1 and
 1 eq { 0  1  3  { j add ch exch get } for  0 1 3 { j add exch ch  3  1  roll put } for      } if    for

/c 0 def          x  y  moveto                                  %  impression des barres
ch { 48 sub dup 9 eq { pop 1 /s true def } { /s false def } ifelse    d mul  dup
setlinewidth 2 div dup 0 rmoveto currentpoint  /c c dup setgray 1 xor store  gsave
s { 0 -3 rmoveto 0  3  h  add  rlineto }  { 0 h rlineto } ifelse
  stroke grestore  moveto 0 rmoveto }  forall
/Courier  findfont  5 scalefont  setfont                      %  impression des chiffres en clair
/i -1 def        /u x d 4.3 mul sub def          /v  y 5 g mul sub def     %  position de départ
codbar {/i i 1 add store u i 0 eq {4 d mul sub} if i 6 gt {3.75 d mul add} if    %  pos. individ.
7 d mul i mul add v moveto   48 sub 10 ( ) cvrs show} forall   %  chiffres clairs

u d 99 mul g mul add 20 add y 8 add moveto  prix  show         %  impression texte associé
showpage

% on a créé une chaîne ch de 58 caractères représentant chacun l'épaisseur de 2 barres  alternativement noire
% et blanche. Dans les parties A & C, on écrit d'abord du code C, puis on le symétrise pour les codes B.
% Les codes A restent tels quels, mais ils seront inversés à l'impression. Pour plus de détails, cf. "Initiation à
% l'informatique", annexe 3, du même auteur.   Résultat ci-contre (page 62).
```

Programme de tracé graphique

On suppose qu'un programme en C (ou en un autre langage) a ouvert un fichier contenant des nombres, en a saisi le contenu, extrait les bornes de variation, etc. Ce programme envoie ensuite à l'imprimante PS des chaînes telles que celles-ci :

```
fic = freopen ("PRN", "wt", stdout) ;                                   /*  redirection des sorties  */
printf ("%%!PS-Adobe-2.0 \n") ;                                      /*  pseudo-instruction normalisée  */

/*  Les 20 lignes suivantes constituent un prologue fixe, contenant des définitions  */

printf ("/mm  {2.8345 mul} def      /trl {translate}  def      /deb  {gsave} def \n") ;
printf ("/mv {moveto} def           /rm {rmoveto} def          /rl {rlineto} def \n") ;
printf ("/ferm {closepath} def      /ret {grestore} def        /trc {stroke} def \n") ;
printf ("/ep {setlinewidth} def     /cp {currentpoint} def     /li {lineto} def \n") ;
printf ("/gris   {deb setgray fill ret} def \n ") ;
printf ("/point  { 0 0 mv 0 0 li ep setgray trc } def \n ") ;
printf ("/cerc0  {deb trl 0 s point ret } def \n ") ;                   /* symboles :  cercle noir  */
printf ("/cerc1  {deb trl 1 s point ret } def \n ") ;                          /*  cercle blanc  */
printf ("/cerc2  {deb trl 1 s point 0 s 5 div point ret }  def \n ") ;         /*  cercle pointé  */
printf ("/trian  {dup scale -.5 -.2887 mv 1 0 rl -.5 1 rl ferm 0 gris trc } def \n ") ;
printf ("/tri0   { deb trl 0 s trian ret } def \n ") ;                         /*  triangle noir  */
printf ("/carre  { dup scale .5 .5 mv 1 0 li 0 1 li -1 0 li ferm gris trc } def \n") ;
printf ("/car0   { deb trl 0 s carre ret } def \n") ;                          /*  carré noir  */
printf ("/plus  {deb trl s s scale 0 .5 mv 0 1 li  .5 0 mv 1 0 li trc ret} def \n");   /*  signe +  */
printf ("/crux  {deb trl s s scale .5 .5 mv 1 1 li  -.5 .5 mv 1 -1 li trc ret} def \n");   /*  croix  */
printf ("/ch1 1 string def          /recul {stringwidth pop neg 2 div 0 rm } def \n") ;
printf ("/ecrit {dup dup show length 1 sub get ch1 0 3 2 roll put ch1 recul } def \n") ;
printf ("/acc {ch1 0 3 2 roll put ch1 recul ch1 show } def \n") ;              /*  lettres accentuées  */
printf ("/aig {ecrit 194 acc} def         /gra {ecrit 193 acc} def \n") ;
printf ("1 mm 1 mm  scale     1 setlinejoin   1  setlinecap     clear     0.15 ep \n") ;

/* constantes du graphique: uc=60, vc=38 (position x & y en mm du coin bas gauche du cadre); lc=100, hc=70
(longueur et hauteur du cadre en mm); lm=2 (longueur en mm des grandes  marques sur les axes); npm=5 (nombre
de petites marques entre les grandes); x0=-15, xm=48, y0=0.092, ym=0.13 (bornes du graphique en unités utilisa-
teur); dx=10, dy=0.01 (intervalles entre grandes marques en x et y), pa=5  (pas du tableau en x, mêmes unités)  */

printf ("/uc %d def     /vc %d def    /lc %d def    /hc %d def \n ", uc, vc, lc, hc) ;
printf ("/x0  %f def    /xm %f def    /y0 %f def    /ym %f def \n ", x0, xm, y0, ym) ;
printf ("/dx %f def     /dy %f def    /lm %f def    /npm %d def \n ", dx, dy, lm, npm) ;

    /*  coefficients pour le changement d'échelle :  u et v étant les coordonnées d'un point en mm, x et y celles de
                                                  l'utilisateur, on a :  u = a*x + b   et   v = c*y + d  */
printf ("/a lc xm x0 sub div def     /c hc ym y0 sub div def \n ") ;
printf ("/b uc a  x0 mul sub def     /d vc c y0 mul sub def    /pa %f def \n ", pa) ;
printf ("/dpm dx npm div def \n ") ;                                  /*  longueur entre petites marques en x  */
printf ("/dpy dy npm div def \n ") ;                                  /*  longueur entre petites marques en y  */
printf ("/x1 x0 dx div 1 add floor dx mul def \n ") ;                       /*  première grande marque en x  */
printf ("/y1 y0 dy div 1 add floor dy mul def \n ") ;                       /*  première grande marque en y  */
printf ("/u1  x1 a mul b add def \n ") ;                                    /*  position 1re gde marque en x  */
printf ("/v1  y1 c mul d add def \n ") ;                                    /*  position 1re gde marque en y  */

printf ("/Times-Italic  findfont %f scalefont  setfont \n ", corps) ;                /*  police pour légendes  */
printf ("uc lc add vc mv (%s) stringwidth pop neg -10 rm \n", legendx) ;
printf (" (tempe) aig (rature  \\(degre) aig (s C\\))  show ") ;                          /*  légende axe x  */
printf ("/ch (%s) def \n", legendy) ;                                       /*  légende sur axe y  */
printf ("deb uc 12 sub vc hc add ch stringwidth pop sub trl 0 0 mv 90 rotate  ch show ret \n") ;

printf ("/Times-Roman  findfont %f scalefont  setfont \n", cor2);                    /*  pour valeurs sur axes  */
                                                          /*  procédure pour les marques sur l'axe x */
printf ("/marqx {/x exch def x vc mv 0 l rl trc x vc hc add mv 0 l neg rl trc } def \n ") ;
printf ("/l lm def          u1 dx a mul uc lc add  {marqx} for\n") ;                 /*  grandes marques en x  */
printf ("/l lm 2 div  def  u1 dpm a mul uc lc add  {marqx} for ");                    /*  petites marques en x  */
printf ("    u1 dpm neg a mul uc {marqx} for \n ");                          /*  premières petites marques x  */
printf ("x1 dx xm {dup  a mul b add vc lm 2 add sub mv (     ) cvs dup stringwidth pop ") ;
```

```
printf ("   -2 div 0 rm show }  for \n") ;                          /*  valeurs sur l'axe x  */
printf ("/marqy {/y exch def  uc y mv l 0 rl trc }  def \n ") ;     /*  proc. pour marques y  */
printf ("/l lm 2 div def    v1 dpy c mul vc hc add {marqy} for  ") ;  /*  petites marques y */
printf ("           v1 dpy neg c mul vc  {marqy} for\n ") ;        /*  premières petites marques y */
printf ("/l lm def          v1 dy c mul vc hc add  {marqy}  for\n ") ;  /*  gdes marques en y */
printf ("y1 dy ym 1.01 mul {dup c mul d add uc exch mv (     ) cvs dup ") ;
printf ("   stringwidth pop neg 1 sub -1 rm show }  for\n ") ;      /*  valeurs sur axe y  */

printf ("deb   uc vc mv  lc 0 rl  0 hc rl  lc neg 0 rl ferm clip trc\n ") ;   /*  tracé du cadre  */
```

/ envoi des tableaux de valeurs en y. On suppose une progression régulière en x */*

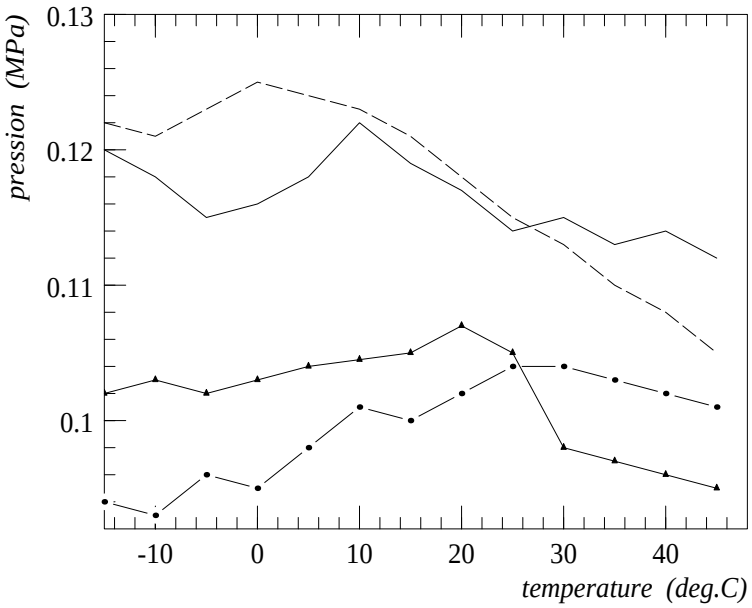
```
printf ("/ty1 [.095 .094 .096 .095 .098 .101 .1 .102 .104 .104 .103 .102 .101] def \n") ;
printf ("/ty2 [.102 .103 .102 .103 .104 .1045 .105 .107 .105 .098 .097 .096 .095] def \n") ;
printf ("/ty3 [.12 .118 .115 .116 .118 .122 .119 .117 .114 .115 .113 .114 .112] def \n") ;
printf ("/ty4 [.122 .121 .123  .125 .124 .123 .121 .118 .115 .113 .11 .108 .105] def \n") ;
```

/ définition d'une procédure traçant une courbe continue */*

```
printf ("/courb { {c mul d add x a mul b add exch 3 2 roll {li} {mv} ifelse  /x  x  pa add  def") ;
printf ("  true } forall trc pop }  def     /s %f def \n ", s ) ;          /* taille des symboles */

printf ("/x  x0 def false %s courb \n", "ty1") ;          /*  tracé de ty1 en continu, puis ... */
printf ("/x  x0 def ty1 {c mul d add x a mul b add exch 2 copy  /s 3 def cerc1 ") ;
printf (" cerc0 /x x pa add def } forall \n") ;          /*  superposition de points détachés  */
                                                         /*  s est la taille des symboles en mm  */
printf ("/x  x0 def false %s courb \n", "ty2") ;          /*  courbe continue pour ty2, puis ... */
printf ("/x  x0 def ty2 {c mul d add x a mul b add exch 2 copy /s 1 def ") ;
printf ("    tri0  /x x pa add def }  forall \n") ;       /*  superposition du symbole triangle  */
printf ("/x  x0 def false %s courb \n", "ty3") ;          /*  ty3 en courbe simple  */
printf ("deb [%d 1] 0 setdash  /x  x0 def false %s courb  ret \n", 3, "ty4") ;
                                                         /*  courbe ty4 en pointillé  */
printf ("ret     showpage \n") ;                         /*  fin du graphique  */
fclose (fic) ;                                           /*  résultat ci-dessous  */
```

En remplaçant dans la première instruction le nom symbolique PRN par celui d'un fichier, on envoie les instructions dans un fichier PS importable.



On peut réaliser ce travail uniquement en PS, en faisant suivre le programme de tracé par la liste des nombres à représenter. Ils seront considérés comme contenus dans le fichier **currentfile** *qu'on lit selon les instructions p. 34 (chap. 6). Noter qu'il est impossible en opérant ainsi de composer un fichier PS de plusieurs pages satisfaisant aux normes DSC.*

Annexe 3

DESCRIPTEURS TIFF

La table suivante donne de manière non exhaustive : le numéro du descripteur en décimal, puis en hexadécimal, puis tel qu'il est écrit dans les fichiers II (Intel). Elle donne ensuite le type des valeurs contenues (Entier, Long, Ascii, Fraction) et leur nombre dans le champ (NR signifie: *descripteur de type ancien, non recommandé)*.

254	FE (254,0)	L	1	Type d'image : 0 = normale (défaut), 1 ou 3 = réduction, 2 ou 3 = masque (262→4)
255	FF (255,0)	E	1	Type d'image : 1=fichier normal (NR).
256	100 (0,1)	E|L	1	Largeur image. Nb de pixels par ligne.
257	101 (1,1)	E|L	1	Hauteur image. Nb de lignes.
258	102 (2,1)	E	*Nc*	*Nb* = Nb de bits par pixel et par composante.
259	103 (3,1)	E	1	Type compress. : 1=néant, 2,3,4=CCITT, 5=LZW, 32773 (5,128)=Packbit. Déf.: 1
262	106 (6,1)	E	1	Photométrie : 0=N-B (0=Blanc), 1=N-B (1=B), 2=RGB, 3=palette, 4=masque semi-transparent.
263	107 (7,1)	E	1	Type de traduction d'image demi-teinte en N-B.
264	108 (8,1)	E	1	Largeur matrice de grisé en bits (NR).
265	109 (9,1)	E	1	Hauteur matrice de grisé en bits (263 doit être à 2), (NR).
266	10A (10,1)	E	1	Ordre des bits dans les octets-pixel : 1=fort-faible (défaut), 2 = faible-fort (NR).
269	10D (13,1)	Asc		Nom du document (pour fax).
270	10E (14,1)	Asc		Commentaire ou nom de l'image.
271	10F (15,1)	Asc		Fabricant appareil de saisie.
272	110 (16,1)	Asc		Modèle d'appareil.
273	111 (17,1)	E|L	*Ns*	Adresses du début des *Ns* bandes-images.
274	112 (18,1)	E	1	Origine image : 1=haut gauche, 2=haut droit, 4=bas gauche , ... (NR).
277	115 (21,1)	E	1	***Nc*** = nb de composantes couleur par pixel : 1 pour N-B et palette, 3 pour RVB.
278	116 (22,1)	E|L	1	Nb de lignes par bande.
279	117 (23,1)	E|L	*Ns*	Nombre d'octets par bande.
280	118 (24,1)	E	*Nc*	Val. mini du pixel (ou d'une composante) (NR).
281	119 (25,1)	E	*Nc*	Val. maxi du pixel (ou d'une composante) (NR)
282	11A (26,1)	Frac	1	Linéature, nb. de pixels par u. de longueur en *x*.
283	11B (27,1)	Frac	1	Résolution en *y*, nb. de pixels par unité de longueur.
284	11C (28,1)	E	1	Disposition des plans couleur : 1=1 seul plan, (pixels contigus), 2=plans séparés.
285	11D (29,1)	Asc		Nom de la page (pour fax).
286	11E (30,1)	Frac	1	Position de l'image en *x* (pour fax).
287	11F (31,1)	Frac	1	idem en *y* .
288	120 (32,1)	L		Adresse de chaque zone libre (NR).
289	121 (33,1)	L		Nb d'octets dans chaque zone libre (NR).
290	122 (34,1)	E	1	Exposant *n* du diviseur de 291.
291	123 (35,1)	E	2^{Nb}	Unités de gris (valeurs de 0 à 2), à diviser par 10^n *(n* donné par 290).
292	124 (36,1)	L	1	Option pour fax, si compression CCITT3.
293	125 (37,1)	L	1	... idem si CCITT4.
296	128 (40,1)	E	1	Unité de longueur pour 282 : 1=néant, 2=inch (défaut), 3=cm.
297	129 (41,1)	E	2	Numéro de page et nb. total de pages (1ère=0).
301	12D (41,1)	E	$3*2^{Nb}$	3 courbes de réponses RVB (défaut, courbe NTSC, γ=2,2).
305	131 (45,1)	Asc		Nom et version du logiciel générateur.
306	132 (50,1)	Asc	20	Groupe date-heure numérique.
315	13B (59,1)	Asc		Auteur.
316	13C (60,1)	Asc		Calculateur.
317	13D (61,1)	E	1	Prédicteur pour compression LZW (1 : néant).
318	13E (62,1)	Frac	2	Coordonnées du blanc de référence dans le graphe chromatique CIE-1931.
319	13F (63,1)	Frac	6	Chromaticités primaires (coordonnées dans le graphe ci-dessus).
320	140 (64,1)	E	$3*2^{Nb}$	Table de couleurs (palette RVB, noir= (0,0,0), blanc=3*65535).
au-delà de 32768				descripteurs privés, propres à une société.

Pour plus d'information : *Le format TIFF et ses modes de compression,* Christian Monteix (Eyrolles).

Annexe 4 - POLICES PS NORMALES

octal	*..0*	*..1*	*..2*	*..3*	*..4*	*..5*	*..6*	*..7*
\00..								
\01..								
\02..								
\03..								
\04..		!	"	#	$	%	&	’
\05..	(	)	*	+	,	-	.	/
\06..	0	1	2	3	4	5	6	7
\07..	8	9	:	;	<	=	>	?
\10..	@	A	B	C	D	E	F	G
\11..	H	I	J	K	L	M	N	O
\12..	P	Q	R	S	T	U	V	W
\13..	X	Y	Z	[	\	]	^	_
\14..	‘	a	b	c	d	e	f	g
\15..	h	i	j	k	l	m	n	o
\16..	p	q	r	s	t	u	v	w
\17..	x	y	z	{	|	}	~	
\20..								
\21..								
\22..								
\23..								
\24..		¡	¢	£	⁄	¥	*f*	§
\25..	¤	'	“	«	‹	›	fi	fl
\26..	°	–	†	‡	·	µ	¶	•
\27..	‚	„	”	»	…	‰	¾	¿
\30..	À	`	´	ˆ	˜	¯	˘	˙
\31..	¨	É	°	¸	Ì	˝	˛	ˇ
\32..	—	Ñ	Ò	Ó	Ô	Õ	Ö	×
\33..	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
\34..	à	Æ	â	ª	ä	å	æ	ç
\35..	Ł	Ø	Œ	º	ì	í	î	ï
\36..	ð	æ	ò	ó	ô	ı	ö	÷
\37..	ł	ø	œ	ß	ü	ý	þ	ÿ

Les zones en gris correspondent aux codes de commande ASCII (selon Adobe)

Annexe 5 - POLICE PS SYMBOL

octal	*..0*	*..1*	*..2*	*..3*	*..4*	*..5*	*..6*	*..7*
\00..								
\01..								
\02..								
\03..								
\04..		!	∀	#	∃	%	&	∍
\05..	(	)	*	+	,	−	.	/
\06..	0	1	2	3	4	5	6	7
\07..	8	9	:	;	<	=	>	?
\10..	≅	Α	Β	Χ	Δ	Ε	Φ	Γ
\11..	Η	Ι	ϑ	Κ	Λ	Μ	Ν	Ο
\12..	Π	Θ	Ρ	Σ	Τ	Υ	ς	Ω
\13..	Ξ	Ψ	Ζ	[	∴	]	⊥	_
\14..		α	β	χ	δ	ε	ϕ	γ
\15..	η	ι	φ	κ	λ	μ	ν	ο
\16..	π	θ	ρ	σ	τ	υ	ϖ	ω
\17..	ξ	ψ	ζ	{	|	}	~	
\20..								
\21..								
\22..								
\23..								
\24..	€	ϒ	′	≤	⁄	∞	*f*	♣
\25..	♦	♥	♠	↔	←	↑	→	↓
\26..	°	±	″	≥	×	∝	∂	•
\27..	÷	≠	≡	≈	…	|	—	↵
\30..	ℵ	ℑ	ℜ	℘	⊗	⊕	∅	∩
\31..	∪	⊃	⊇	⊄	⊂	⊆	∈	∉
\32..	∠	∇	®	©	™	∏	√	·
\33..	¬	∧	∨	⇔	⇐	⇑	⇒	⇓
\34..	◊	〈	®	©	™	∑	⎛	⎡
\35..	⎝	⎢	⎜	⎣	⎜	⎧	⎩	⎜
\36..		〉	⌠	⎞	⎟	⎭	⎫	⎥
\37..	⎭	⎤	⎥	⎦	⎤	⎨	⎭	

Annexe 6

POSTSCRIPT NIVEAU 2

Le langage PostScript décrit dans ce livre est celui de niveau 1. Le niveau 2, apparu peu après la première rédaction, ne modifie pas les bases du langage, mais l'unifie et l'élargit, tout en maintenant une compatibilité correcte avec le niveau 1 (PS de niveau 1 est accepté par toutes les imprimantes de niveau 2).

Unification : PS intégrera désormais dans le produit de base des adjonctions qui permettaient à certains utilisateurs d'utiliser le modèle de couleurs CMJK (cyan, magenta, jaune, noir), modèle complémentaire du RVB, indispensable pour réaliser des négatifs en photogravure. De même, le modèle HSB de PS devient normalisé en se ralliant aux définitions de la CIE (compagnie internationale de l'éclairage, 1931). Les normes émises par cette association servent de référence à nombre de "produits" couleurs et, en particulier, aux fichiers TIFF. De même, PS englobera désormais les jeux de caractères Kanji.

Elargissement : ajout de possibilités concernant la sauvegarde des formes (objets, dessins partiels, procédures...) dans une antémémoire (*cache* en anglais), amélioration des procédés de tramage couleurs, **décodage direct des fichiers images comprimés**, emploi de hachures pour remplir des figures, etc.

Toutefois, dans le niveau 2, Adobe a précisé des notations importantes comme la structuration par les **DSC** et ajouté un certain nombre d'instructions, qui méritent que nous en parlions un peu.

L'instruction **setpagedevice** donne à l'interpréteur des informations très complètes sur le **papier** : dimensions exactes, numéro du bac d'alimentation, façon dont les pages doivent être classées, pliées et coupées. Ces paramètres concernent plutôt des imprimantes haut de gamme. L'instruction **setpagedevice** modifie en réalité un dictionnaire clé par clé. Nous ne citerons que deux de ces clés, **PageSize** qui donne la largeur *l* et la hauteur *h* du papier et la clé **Duplex** qui, mise à la valeur **true**, provoque l'impression **recto-verso** (si la machine le permet bien sûr) :

```
<< /PageSize [l h] /Duplex true >> setpagedevice
```

Noter les nouveaux opérateurs **<< >>** pour ouvrir et fermer un dictionnaire. Nous n'étudierons pas plus cette puissante instruction (voir le livre d'Adobe). L'amateur sera davantage concerné par les précisions apportées sur la structuration des documents par les DSC. Ont été précisées des notions importantes, comme celles de **fichier EPSF** *(encapsulated PS file)* et celle de **document multipage**. La clarification de ces notions n'a pas exigé des modifications de langage, mais seulement des compléments sur les *DSC*.

Les **DSC** *(document structuring conventions)* sont en quelque sorte des **directives** ; non destinées à l'interpréteur PS, elles donnent des informations indispensables au logiciel hôte (celui qui va transmettre ou incorporer le fichier PS). Ces directives s'écrivent chacune sur une ligne entière, laquelle commence par le double symbole **%%.**

Un **fichier EPSF** est destiné à être incorporé dans un autre document ; il décrit une page au plus. C'est souvent une illustration. Il doit comporter au moins la *directive* **%%BoundingBox:**, mais peut donner titre (**%%Title:**) ou auteur (**%%Creator**) qui seront affichés par l'hôte s'il n'exécute pas le PS. L'hôte donnera un aperçu du contenu de l'EPSF s'il est doté d'une vignette **(%%BeginPreview:)** chaîne dessinant l'image pixellisée *(bitmap)* à basse résolution. Ce fichier devrait commencer par **%!PS-Adobe-3.0-EPSF**. Enfin, comme tout fichier importé, il ne doit pas comporter certaines instructions (**showpage**,... cf. page 51, §6).

Un **document multipage** doit être maintenant mieux structuré. Il commencera toujours par la ligne **%!PS-Adobe**, texte qui devrait être suivi de **–3.0** (ce qui indique sa conformité aux DSC). Toutes les DSC (sauf celles qui débutent les pages et celles du *final*) doivent suivre immédiatement cette première ligne, mais sans ordre imposé. On doit donner le nombre *n* de pages dans la directive **%%Pages:** *n* (*n* entier). On peut en donner l'orientation ; c'est très utile (**%%orientation: Landscape** ou **Portrait**). Après les DSC, viendra le **prologue**, qui ne doit comporter que des définitions. Puis vient la page 1, précédée de la directive **%%Page: 1** suivie d'un autre entier, donnant la façon dont ce numéro est imprimé dans le document (chiffre romain par exemple). Chaque page est précédée de la directive **%%Page:** suivie de deux entiers comme ci-dessus. Un final précédé de la directive **%%Trailer** peut terminer le document. A la suite de **%%Trailer**, des directives peuvent figurer, comme **%%EOF** (tout à la fin) ou **%%Pages:** *n*, avec *n* pour nombre de pages. Dans ce cas, la directive correspondante dans l'en-tête doit s'écrire **%%Pages: (atend)**.

Les logiciels comme **Gsview**, capables d'interpréter des documents PS multipage, ne peuvent en afficher correctement le contenu que si ces DSC sont respectées. Pour ce faire, il est indispensable que les pages soient indépendantes les unes des autres (mais elles peuvent utiliser des définitions communes contenues dans le prologue avant la première page). Il semble que, bien que l'état graphique soit en général le même de page à page, il ne faille pas tenter de le sauver en fin de page pour le récupérer à la page suivante. Il vaut mieux le définir dans le prologue et

l'appeler au début de chaque page. Bien sûr, chaque page débutera par un **gsave** et finira par un **grestore**.

On peut résumer ces spécifications par le schéma suivant auquel devrait se conformer tout fichier PS.

!PS-Adobe-3.0
%%BoundingBox: x_0 y_0 x_1 y_1
(seulement pour fichier EPSF)
%%Tittle: *titre*
%%Pages: *n* *ou bien* **%%Pages: (atend)**
autres directives
Prologue (instructions PS de définitions communes)
/etagr **{***..toutes les instructions relatives à l'état graphique.***} bind def**
%%Page: 1 1 *(ou 1 I, 1 A, ...)*
gsave *etagr .....*
instructions pour la page 1
grestore
%%Page: 2 2
gsave *etagr .....*
idem
.....
.....
autres pages....
.....
grestore
%%Trailer
%%Pages: *n (si en en-tête figure* **%%Pages: (atend)***)*
%%EOF

Enfin, PS niveau 2 accepte les images comprimées si on utilise les *filtres* (décodeurs) appropriés. Citons le cas important des **images JPEG** (photo couleur). Si on en connaît les caractéristiques, nombre de lignes *nl*, nombre de pixels par ligne *np*, nombre de bits par pixel *nb* (généralement 8), on peut écrire :

/fich **(***nom complet du fichier JPEG***) (r) file def**
x y **translate** *(donner les valeurs numériques n, x, y, r)*
/np n **def** */nh n* **def** */nc n* **def**
/DeviceRGB setcolorspace *np nh* **scale** *r* **dup scale**
<< /DataSource *fich* **/DCTDecode filter /ImageType 1**
/Width *np* **/Height** *nh* **/ImageMatrix [np 0 0 nh neg 0 nh]**
/BitsPerComponent *nc* **/Decode [0 1 0 1 0 1] >> image**

Si on ne connaît pas *np*, *nh* et *nc*, il faut les extraire du fichier JPEG en première lecture, puis revenir en début de fichier avec les instructions ci-après à insérer avant les 4 dernières lignes ci-dessus (**setfileposition** est parfois refusé par l'imprimante).

/*fich* **(***nom complet du fichier JPEG***) (r) file def**
/*k* **-1 def /***cp* **0 def**
{/*c fich* **read {def} {exit} ifelse**
k **0 ge {***k* **3 eq {/***nb c* **def} if**
k **5 eq {/***nh c cp* **256 mul add def} if**
k **7 eq {/***nl c cp* **256 mul add def} if**
k **8 eq {/***nc c* **def exit} if} /***k k* **1 add def} if**
c **16#C0 eq {***cp* **16#FF eq {/***k* **1 def} if } {/***cp c* **def} ifelse**
} loop *fich* **0 setfileposition**
/DeviceRGB setcolorspace ... [0 1 0 1 0 1] >> image

ACROBAT

Les logiciels de la série *Acrobat* créent ou utilisent des fichiers ***.pdf** *(portable document files)*. Avec ces fichiers, Adobe a voulu créer un format de document universel, capable de contenir tout produit de bureautique, traitement de texte, tableur, dessin, plan, image, etc, et ceci, pour n'importe quel type de machine. Le langage Acrobat (non lisible avec un éditeur ASCII) est une forme dérivée de PS, mais en plus simple.

On peut créer un fichier *pdf* avec certains logiciels bureautiques récents (comme Word). On peut également le faire avec le logiciel *Acrobat Distiller* (commercial) qui transforme en *pdf* les fichiers PostScript. Ce logiciel semble fournir de meilleures résultats que ceux de bureautique (en 1998 du moins). Un autre logiciel commercial, *Exchange*, permet de procéder à quelques modifications (par exemple ajouter des liens de type hypertexte) dans un fichier *pdf*.

Une fois créé, le document *pdf* est très facile à lire et imprimer, puisque, pour presque tous les systèmes d'exploitation de type graphique (donc pour Windows, mais pas pour MS-DOS), Adobe, relayé par les serveurs Internet, distribue gratuitement le logiciel **Acrobat Reader** ou **Acroread** ; ce dernier permet

- d'imprimer tout fichier *pdf* sur toute imprimante reconnue par le système d'exploitation ;

- de lire à l'écran ce même type de document avec une grande aisance : possibilité d'accès aux pages dans n'importe quel ordre et agrandissement de toute page ou fraction de page à n'importe quelle échelle. La plus haute qualité est toujours présente, le langage étant vectoriel.

Le format *pdf* conserve mise en page, enrichissements, couleurs, illustrations ... C'est un outil de communication d'une très grande qualité. En principe, il ne permet pas la modification du document communiqué (ce qui contribue au respect du droit d'auteur), mais il autorise son analyse avec des fonctions de recherche sur les mots ou sur les phrases.

Annexe 7

LISTE DES CARACTERES PS PREDEFINIS

Les numéros, en octal, sont égaux à la position dans la table de codage

	space	040	@	at	100	«	guillemotleft	253	˙	dotaccent	307
!	exclam	041	A-Z	A-Z	101-132	‹	guilsingleleft	254	¨	dieresis	310
"	quotedbl	042	[	bracketleft	133	›	guilsingleright	255	°	ring	312
#	numbersign	043	\	backslash	134	fi	fi	256	¸	cedilla	313
$	dollar	044	]	bracketright	135	fl	fl	257	˝	hungarumlaut	315
%	percent	045	^	asciicircum	136	–	endash	261	˛	ogonek	316
&	ampersand	046	_	underscore	137	†	dagger	262	ˇ	caron	317
´	quoteright	047	`	quoteleft	140	‡	daggerdbl	263	—	emdash	320
(	parenleft	050	a-z	a-z	141-172	.	periocentered	264	Æ	AE	341
)	parenright	051	{	braceleft	173	§	paragraph	266	ª	ordfeminine	343
*	asterisk	052	|	bar	174	•	bullet	267	Ł	Lslash	350
+	plus	053	}	braceright	175	,	quotesinglebase	270	Ø	Oslash	351
,	comma	054	~	ascitilde	176	„	quotedblbase	271	Œ	OE	352
-	hyphen	055	¡	exclamdown	241	»	guillemotright	273	º	ordmasculine	353
.	period	056	¢	cent	242	…	ellipsis	274	æ	ae	361
/	slash	057	£	sterling	243	‰	perthousand	275	ı	dotlessi	365
0-9	zero-nine	060-071	/	fraction	244	¿	questiondown	277	ł	lslash	370
:	colon	072	¥	yen	245	`	grave	301	ø	oslash	371
;	semicolon	073	ƒ	florin	246	´	acute	302	œ	oe	372
<	less	074	§	section	247	ˆ	circumflex	303	ß	germandbls	373
=	equal	075	¤	currency	250	˜	tilde	304			
>	greater	076	'	quotesingle	251	¯	macron	305			
?	question	077	“	quotedbleft	252	˘	breve	306			

Caractères supplémentaires codés en ISOLatin1

Á	Aacute	301	Ó	Oacute	323	ç	ccedilla	347	ó	oacute	363
Â	Acircumflex	302	Ô	Ocircumflex	324	©	copyrigth	251	ô	ocircumflex	364
Ä	Adieresis	304	Ö	Odieresis	326	°	degree	260	ö	odieresis	366
À	Agrave	300	Ò	Ograve	322	÷	divide	367	ò	ograve	362
Å	Aring	305	Õ	Otilde	325	é	eacute	351	½	onehalf	275
Ã	Atilde	303	Ú	Uacute	332	ê	ecircumflex	352	¼	onequarter	274
Ç	Ccedilla	307	Û	Ucircumflex	333	ë	edieresis	353	õ	otilde	365
É	Eacute	311	Ü	Udieresis	334	è	egrave	350	±	plusminus	261
Ê	Ecircumflex	312	Ù	Ugrave	331	í	iacute	355	®	registered	256
Ë	Edieresis	313	á	aacute	341	î	icircumflex	356	¾	threequarters	276
È	Egrave	310	â	acircumflex	342	ï	idieresis	357	ú	uacute	372
Í	Iacute	315	ä	adieresis	344	ì	igrave	354	û	ucircumflex	373
Î	Icircumflex	316	à	agrave	340	¬	logicalnot	254	ü	udieresis	374
Ï	Idieresis	317	å	aring	345	µ	mu	265	ù	ugrave	371
Ì	Igrave	314	ã	atilde	343	×	multiply	327	ÿ	ydieresis	377
Ñ	Ntilde	321	¦	brokenbar	246	ñ	ntilde	361			

Ces caractères sont disponibles en les écrivant \nnn (code octal)
à condition que la police en cours soit codée en ISOLatin2.

Jeu de caractères pour la police Symbol

	space	040	ς	sigma1	126	∞	infinity	245	®	registerserif	322
!	exclam	041	Ω	Omega	127	*f*	florin	246	©	copyrightserif	323
∀	universal	042	Ξ	Xi	130	♣	club	247	™	trademarkserif	324
#	numbersign	043	Ψ	Psi	131	♦	diamond	250	∏	product	325
∃	existential	044	Z	Zeta	132	♥	heart	251	√	radical	326
%	percent	045	[	bracketleft	133	♠	spade	252	·	dotmath	327
&	ampersand	046	∴	therefore	134	↔	arrowboth	253	¬	logicalnot	330
∍	suchthat	047	]	bracketright	135	←	arrowleft	254	∧	logicaland	331
(	parenleft	050	⊥	perpendicular	136	↑	arrowup	255	∨	logicalor	332
)	parenright	051	_	underscore	137	→	arrowright	256	⇔	arrowdblboth	333
*	asteriskmath	052		radicalex	140	↓	arrowdown	257	⇐	arrowdblleft	334
+	plus	053	α	alpha	141	°	degree	260	⇑	arrowdblup	335
,	comma	054	β	beta	142	±	plusminus	261	⇒	arrowdblright	336
–	minus	055	χ	chi	143	″	second	262	⇓	arrowdbldown	337
.	period	056	δ	delta	144	≥	greaterequal	263	◊	lozenge	340
/	slash	057	ε	epsilon	145	×	multiply	264	〈	angleleft	341
0-9	zero-nine	60-71	ϕ	phi	146	∝	proportional	265	®	registersans	342
:	colon	072	γ	gamma	147	∂	partialdiff	266	©	copyrightsans	343
;	semicolon	073	η	eta	150	•	bullet	267	™	trademarksans	344
<	less	074	ι	iota	151	÷	divide	270	∑	summation	345
=	equal	075	φ	phi1	152	≠	notequal	271	⎛	parenlefttp	346
>	greater	076	κ	kappa	153	≡	equivalence	272	⎜	parenleftex	347
?	question	077	λ	lambda	154	≈	approxequal	273	⎝	parenleftbt	350
≅	congruent	100	μ	mu	155	…	ellipsis	274	⎡	bracketlefttp	351
A	Alpha	101	ν	nu	156	⏐	arrowvertex	275	⎢	bracketleftex	352
B	Beta	102	ο	omicron	157	—	arrowhorizex	276	⎣	bracketleftbt	353
X	Chi	103	π	pi	160	↵	carriagereturn	277	⎧	bracelefttp	354
Δ	Delta	104	θ	theta	161	ℵ	aleph	300	⎨	braceleftmid	355
E	Epsilon	105	ρ	rho	162	ℑ	Ifraktur	301	⎩	braceleftbt	356
Φ	Phi	106	σ	sigma	163	ℜ	Rfractur	302	⎪	braceex	357
Γ	Gamma	107	τ	tau	164	℘	weierstrass	303	〉	angleright	361
H	Eta	110	υ	upsilon	165	⊗	circlemultiply	304	⌠	integral	362
I	Iota	111	ϖ	omega1	166	⊕	circleplus	305	⌡	integraltp	363
ϑ	theta1	112	ω	omega	167	∅	emptyset	306	⎮	integralex	364
K	Kappa	113	ξ	xi	170	∩	intersection	307	⌡	integralbt	365
Λ	Lambda	114	ψ	psi	171	∪	union	310	⎞	parenrighttp	366
M	Mu	115	ζ	zeta	172	⊃	propersuperset	311	⎟	parenrightex	367
N	Nu	116	{	braceleft	173	⊇	reflexsuperset	312	⎠	parenrightbt	370
O	Omicron	117	|	bar	174	⊄	notsubset	313	⎤	bracketrighttp	371
Π	Pi	120	}	braceright	175	⊂	propersubset	314	⎥	bracketrightex	372
Θ	Theta	121	~	similar	176	⊆	reflexsubset	315	⎦	bracketrightbt	373
P	Rho	122	ϒ	Upsilon1	241	∈	element	316	⎫	bracerighttp	374
Σ	Sigma	123	′	minute	242	∉	notelement	317	⎬	bracerightmid	375
T	Tau	124	≤	lessequal	243	∠	angle	320	⎭	bracerightbt	376
Y	Upsilon	125	⁄	fraction	244	∇	gradient	321			

Annexe 8

GHOSTSCRIPT et GSVIEW

Ghostscript (GS) est un interpréteur PostScript de haute qualité élaboré par la firme *Aladdin Enterprises* et spécialement par L. Peter Deutsch, son président. Ce logiciel est distribué gratuitement sur *Internet* par l'association *Free Software Foundation* (distribution GNU ou Aladdin, serveurs ftp.cs.wisc.edu, ftp.inria.fr, ftp.jussieu.fr ...). Aladdin Enterprises fait sienne la position de la *League for Programming Freedom*, selon laquelle *software patents and broad ("look and feel") copyrights are injurious to the software industry and to the public good*.

Le télé-chargement se fait sous *ftp anonymous*. Si on n'a pas accès à Internet, on peut s'adresser à un ami (la copie est autorisée) ou à Russel Lang (Australie) qui enverra à tout demandeur un disque (optique) contenant GhostScript, moyennant le prépaiement des frais d'expédition. GS peut fonctionner sous Unix, DOS, DOS-Windows, Macintosh, ...

Le logiciel reçu est comprimé par **pkzip** ; après décompression par **gunzip** par exemple, GhostScript se décompose en 5 parties :

- des fichiers d'informations (README, USE.doc, MAN.doc, ...), ainsi que le texte de la licence d'exploitation (fichier général COPYING ; fichier COMMPROD.doc en cas d'usage commercial),

- éventuellement les programmes-source en C, ainsi que les fichiers d'en-tête associés (pour les versions récentes ces fichiers sont distribués à part),

- les programmes exécutables permettant d'interpréter à l'écran des fichiers PostScript et leurs fichiers prologues (initialisations),

- les pilotes *(drivers)* permettant d'imprimer l'interprétation des fichiers PS sur de nombreuses imprimantes laser, à jet d'encre, à bulles ou à aiguilles,

- des jeux de polices de caractères, conformes à PS, mais non identiques à celles d'Adobe (si on le désire, on peut remplacer ces polices par celles d'Adobe, à condition de respecter les droits de cette firme).

Le logiciel complet est volumineux (plusieurs méga-octets de disque). Il n'est pas nécessaire d'installer les fichiers sources (***.C**, ***.H**, ***src.zip**). Les versions anciennes pouvaient être ainsi ramenées à 2 Mo environ (en supprimant aussi les polices peu courantes) ; les plus récentes exigent plutôt 5 à 6 Mo. Actuellement (1998), 3 fichiers comprimés de moins de 1,4 Mo chacun sont nécessaires : **gs***nnn***ini.zip**, **gs***nnnsys***.zip** et **gs***nnn***fn1.zip** (remplacer *nnn* par le symbole de la version et *sys* par celui du système, **w32** sous Windows 95 par exemple. Le dernier fichier contient les polices de caractères).

La vitesse d'exécution de GS et sa précision sont remarquables. Dessins et images apparaissent à l'écran bien plus vite que sur une imprimante. GS reconnaît automatiquement la carte graphique et fonctionne bien avec EGA, VGA et diverses Super-VGA (liste dans **DEVS.mak**). Il détecte bien les erreurs, mais paraît plus tolérant que beaucoup d'imprimantes. Il est facile de faire interpréter à GS avant le fichier principal un fichier prologue (écrire un fichier ***.bat** ou donner successivement le nom des deux fichiers ; c'est moins simple avec Gsview). Seules ombres au tableau, les opérateurs **image** et, en niveau 2, **colorimage** ne semblent pas exploiter toute la dynamique couleur des cartes SVGA, du moins jusqu'à la version 5.01 ; de même, les hautes résolutions des imprimantes ne sont pas toujours prises en compte.

Son aptitude à imprimer des fichiers PS sur des imprimantes ordinaires est du plus haut intérêt. Pour ce faire, il suffit par exemple (mais d'autres moyens sont disponibles) d'ajouter dans le programme la ligne

(*nom* **) selectdevice**

où *nom*, égal par exemple à **laserjet**, est un code d'imprimante figurant dans la liste du fichier **DEVS.mak**. Bien sûr, GS est limité par la résolution (et la mémoire) de la machine. A résolution égale, les **dessins** produits sous GS par une HPLaserjet sont identiques à ceux produits par une imprimante PS. Par contre, les **caractères** fournis avec GS n'atteignent pas toujours la qualité de ceux d'Adobe **(1)**. Cette imperfection est peu visible à l'écran (sauf en grande taille), mais elle se manifeste beaucoup plus vite sur le papier.

Quoiqu'il en soit, la firme Aladdin a mis à la disposition des informaticiens un outil remarquable qui s'améliore d'ailleurs de version en version. Il permet à peu de frais de s'initier à PostScript et de mettre au point les dessins et copies d'images les plus complexes beaucoup plus vite qu'avec une imprimante. Rares sont les cas où l'imprimante PostScript reste nécessaire (résolution meilleure, polices plus variées, couleurs plus fidèles ...).

1 - La qualité des polices s'améliore à chaque version.

GSVIEW

Ghostscript a été conçu plutôt pour représenter des fichiers d'une seule page, tels les **EPSF**. On peut toutefois afficher successivement toutes les pages d'un document en passant à la suivante avec la touche RC *(Enter)*, mais sans pouvoir revenir en arrière.

Ghostview a été écrit par Tim Theisen pour Unix et **Gsview** par Russel Lang pour OS2 et Windows. Ces logiciels, également gratuits, sont distribués largement par Internet en même temps que GhostScript (environ 1 Mo) ; ils affichent à la perfection les documents PS **multipages** pourvu qu'ils soient conformes aux DSC. On peut afficher les pages dans l'ordre croissant (touche +), décroissant (touche –) ou dans un ordre quelconque. On peut les agrandir, les faire pivoter de 90° et enfin imprimer n'importe quelle page – comme avec Ghostscript – sur toute imprimante reconnue par le système d'exploitation, à condition qu'un pilote spécifique ait été écrit pour cette imprimante **(2)**. Tout ceci se commande par *clics* dans une fenêtre *à la Windows*.

Gsview utilise Ghostscript pour interpréter les pages PostScript. Chaque version de Gsview exige une version déterminée de GhostScript. Il existe deux versions "courantes" : l'une est accessible à tous (distribution **gnu**), l'autre plus récente est restreinte aux utilisations non commerciales (distribution **Aladdin**). L'importation de l'une de ces versions (avec les fichiers de police compatibles et obligatoires) demande quelque attention. Une page Internet explique comment faire en fonction du système d'exploitation **(3)** : sous Windows, quatre fichiers comprimés sont nécessaires, trois pour GhostScript, dont un de polices, et un pour Gsview (**gsv***nnnsys*.**zip**, cf p. 72).

On décomprime le fichier **gsv***nnnsys*.**zip** : il fournira des fichiers d'explications (**README**, ...) ainsi qu'un exécutable **install.exe** qui va à peu près tout installer. A sa première exécution, Gsview demandera à préciser la configuration (nom du répertoire contenant Ghostscript et de celui des polices ...). Sous Windows, si l'on veut utiliser Ghostscript directement sans passer par Gsview, il faudra initialiser la variable d'environnement **GS_LIB** dans le fichier **autoexec.bat** :

set GS_LIB = *répertoire du fichier* **GS_INI.PS ;** *répertoire des polices* GhostScript (séparer ces noms par un point-virgule).

Avec Gsview (ou Ghostview) on peut afficher à l'écran deux fenêtres réduites, l'une avec un éditeur (ex. Notepad) dans laquelle on écrit ou on modifie le texte du fichier PostScript, l'autre dans laquelle Gsview (après activation et demande de régénération) affiche le dessin programmé. Toutefois, il semble que les diagnostics d'erreur soient plus précis avec Ghostscript qu'avec Gsview. C'est pourquoi il paraît utile de pouvoir parfois recourir directement à lui (sous Windows, associer les fichiers PS à Gsview, mais installer une icône pour Ghostscript).

Gsview est capable d'afficher également les fichiers **pdf**. D'autres interpréteurs existent en libre service sur Internet pour représenter des documents PostScript (liste disponible sur Internet) **(4)** ; en général payants, ce sont soit des *partagiciels (freeware)* comme RoPS (30$) ou Printfile ou des produits commerciaux comme SuperPrint. Ils sont beaucoup moins volumineux et moins élaborés que GhostScript (ils utilisent en général les polices de Windows).

Le site *primaire* relatif à Ghostscript est le serveur **www.cs.wisc.edu/ghost**, mais de nombreux autres serveurs plus proches de nous rediffusent son contenu dans leurs répertoires (sites *miroirs*, comme celui de Paris-Jussieu).

Une abondante "littérature" existe sur Internet au sujet de Gsview, Ghostscript ou PostScript ; elle témoigne de la vitalité du langage et de l'intérêt de ses interpréteurs. On pourra consulter :

- *Introduction to Ghostscript* (www.cs.wisc.edu/ghost/intro) ;
- *GSview help* (www.cs.wisc.edu/ghost/gsview/winhelp) ;
- *GhostScript user manuel*, de Thomas Mertz, 24 pages (sites GS) ;
- *Ghostscript, Ghostview and GSview* (www.cs.wisc.edu/ghost) ;
- *PostScript sins*, par Kevin Thompson (www.byte.com/art/9508/sec13/art3).
- *Gripes*, par Russel Lang (www.cs.wisc.edu/ghost/gripes) ;
- *What is PostScript* ? (www.gkss.de/W3/PS)
- *Frequently asqued questions* (FAQ, www.lib.ox.ac.uk/internet/news/faq/archive/postscript.faq)

---

2 - liste sur Internet dans www.cs.wisc.edu/~ghost/printer
3 - http:\\www.cs.wisc.edu/~ghost/aladdin/get510 (en 1998).
4 - www.amtec.com/notes/psconv.

Annexe 9 - TABLE ASCII

hex	*0..*	*1..*	*2..*	*3..*	*4..*	*5..*	*6..*	*7..*
..0	*Nul* 0	*Dle* ▶ 16	32	0 48	@ 64	P 80	‘ 96	p 112
..1	*Soh* ☺ 1	*Xon* ◀ 17	! 33	1 49	A 65	Q 81	a 97	q 113
..2	*Stx* ☻ 2	*DC2* ↕ 18	" 34	2 50	B 66	R 82	b 98	r 114
..3	*Etx* ♥ 3	*Xof* ‼ 19	# 35	3 51	C 67	S 83	c 99	s 115
..4	*Eot* ♦ 4	*DC4* ¶ 20	$ 36	4 52	D 68	T 84	d 100	t 116
..5	*Enq* ♣ 5	*Nak* § 21	% 37	5 53	E 69	U 85	e 101	u 117
..6	*Ack* ♠ 6	*Syn* ▬ 22	& 38	6 54	F 70	V 86	f 102	v 118
..7	*Bel* • 7	*Etb* ↨ 23	’ 39	7 55	G 71	W 87	g 103	w 119
..8	*BS* ◘ 8	*Can* ↑ 24	(40	8 56	H 72	X 88	h 104	x 120
..9	*HT* ○ 9	*EM* ↓ 25	) 41	9 57	I 73	Y 89	i 105	y 121
..A	*LF* ◙ 10	*Sub* → 26	* 42	: 58	J 74	Z 90	j 106	z 122
..B	*VT* ♂ 11	*Esc* ← 27	+ 43	; 59	K 75	[91	k 107	{ 123
..C	*FF* ♀ 12	*FS* ∟ 28	, 44	< 60	L 76	\ 92	l 108	| 124
..D	*RC* ♪ 13	*GS* ↔ 29	- 45	= 61	M 77	] 93	m 109	} 125
..E	*SO* ♫ 14	*RS* ▲ 30	. 46	> 62	N 78	^ 94	n 110	~ 126
..F	*SI* ☼ 15	*US* ▼ 31	/ 47	? 63	O 79	_ 95	o 111	127

TABLE IBM 437

Annexe 10

TABLE MACINTOSH

hex	*8..*	*9..*	*A..*	*B..*	*C..*	*D..*	*E..*	*F..*
..0	Ç 128	É 144	á 160	▒ 176	└ 192	╨ 208	α 224	≡ 240
..1	ü 129	æ 145	í 161	▓ 177	┴ 193	╤ 209	β 225	± 241
..2	é 130	Æ 146	ó 162	▓ 178	┬ 194	╥ 210	Γ 226	≥ 242
..3	â 131	ô 147	ú 163	│ 179	├ 195	╙ 211	π 227	≤ 243
..4	ä 132	ö 148	ñ 164	┤ 180	─ 196	╘ 212	Σ 228	⌠ 244
..5	à 133	ò 149	Ñ 165	181	┼ 197	╒ 213	σ 229	⌡ 245
..6	å 134	û 150	ª 166	╢ 182	╞ 198	╓ 214	µ 230	÷ 246
..7	ç 135	ù 151	º 167	╖ 183	╟ 199	╫ 215	γ 231	≈ 247
..8	ê 136	ÿ 152	¿ 168	╕ 184	╚ 200	╪ 216	Φ 232	° 248
..9	ë 137	Ö 153	⌐ 169	╣ 185	╔ 201	┘ 217	Θ 233	● 249
..A	è 138	Ü 154	¬ 170	║ 186	╩ 202	┌ 218	Ω 234	• 250
..B	ï 139	¢ 155	½ 171	╗ 187	╦ 203	▉ 219	δ 235	√ 251
..C	î 140	£ 156	¼ 172	╝ 188	╠ 204	▄ 220	∞ 236	n 252
..D	ì 141	¥ 157	¡ 173	╜ 189	═ 205	▌ 221	∅ 237	2 253
..E	Ä 142	Pt 158	« 174	╛ 190	╬ 206	▐ 222	ε 238	■ 254
..F	Å 143	*f* 159	» 175	┐ 191	╧ 207	▀ 223	∩ 239	255

Les 128 premiers caractères sont ceux de l'ASCII

hex	*8..*	*9..*	*A..*	*B..*	*C..*	*D..*	*E..*	*F..*
..0	Ä 128	ê 144	† 160	♧ 176	¿ 192	– 208	‡ 224	240
..1	Å 129	ë 145	° 161	± 177	¡ 193	— 209	· 225	Ò 241
..2	Ç 130	í 146	¢ 162	≤ 178	¬ 194	“ 210	‚ 226	Ú 242
..3	É 131	ì 147	£ 163	≥ 179	√ 195	” 211	„ 227	Û 243
..4	Ñ 132	î 148	§ 164	¥ 180	*ƒ* 196	‘ 212	‰ 228	Ù 244
..5	Ö 133	ï 149	• 165	₩ 181	≈ 197	’ 213	Â 229	ı 245
..6	Ü 134	ñ 150	¶ 166	δ 182	Δ 198	÷ 214	Ê 230	ˆ 246
..7	á 135	ó 151	ß 167	Σ 183	» 199	◊ 215	Á 231	˜ 247
..8	à 136	ò 152	® 168	Π 184	« 200	ÿ 216	Ë 232	¯ 248
..9	â 137	ô 153	© 169	π 185	… 201	Ÿ 217	È 233	˘ 249
..A	ä 138	ö 154	™ 170	∫ 186	202	⁄ 218	Í 234	˙ 250
..B	ã 139	õ 155	´ 171	ª 187	À 203	¤ 219	Î 235	° 251
..C	å 140	ú 156	¨ 172	º 188	Ã 204	‹ 220	Ï 236	¸ 252
..D	ç 141	ù 157	≠ 173	Ω 189	Õ 205	› 221	Ì 237	˝ 253
..E	é 142	û 158	Æ 174	æ 190	Œ 206	fi 222	Ó 238	˛ 254
..F	è 143	ü 159	Ø 175	ø 191	œ 207	fl 223	Ô 239	ˇ 255

CODE WINDOWS

Annexe 11

CODE ISOLATIN1

hex	*8..*	*9..*
..0	128	144
..1	129	´ 145
..2	‚ 130	´ 146
..3	*f* 131	” 147
..4	„ 132	” 148
..5	… 133	• 149
..6	† 134	– 150
..7	‡ 135	— 151
..8	ˆ 136	˜ 152
..9	‰ 137	™ 153
..A	Š 138	š 154
..B	‹ 139	› 155
..C	Œ 140	œ 156
..D	141	157
..E	142	158
..F	143	Ÿ 159

Les caractères 160 ... 255 de Windows sont ceux de l'ISOLatin1

Les 128 premiers caractères des deux tables sont ceux de l'ASCII

hex	*8..*	*9..*	*A..*	*B..*	*C..*	*D..*	*E..*	*F..*
..0	128	144	160	° 176	À 192	Ð 208	à 224	ð 240
..1	129	145	¡ 161	± 177	Á 193	Ñ 209	á 225	ñ 241
..2	130	146	¢ 162	² 178	Â 194	Ò 210	â 226	ò 242
..3	131	147	£ 163	³ 179	Ã 195	Ó 211	ã 227	ó 243
..4	132	148	¤ 164	´ 180	Ä 196	Ô 212	ä 228	ô 244
..5	133	149	¥ 165	µ 181	Å 197	Õ 213	å 229	õ 245
..6	134	150	¦ 166	¶ 182	Æ 198	Ö 214	æ 230	ö 246
..7	135	151	§ 167	· 183	Ç 199	× 215	ç 231	÷ 247
..8	136	152	¨ 168	¸ 184	È 200	Ø 216	è 232	ø 248
..9	137	153	© 169	¹ 185	É 201	Ù 217	é 233	ù 249
..A	138	154	ª 170	º 186	Ê 202	Ú 218	ê 234	ú 250
..B	139	155	« 171	» 187	Ë 203	Û 219	ë 235	û 251
..C	140	156	¬ 172	¼ 188	Ì 204	Ü 220	ì 236	ü 252
..D	141	157	- 173	½ 189	Í 205	Ý 221	í 237	ý 253
..E	142	158	® 174	¾ 190	Î 206	Þ 222	î 238	þ 254
..F	143	159	¯ 175	¿ 191	Ï 207	ß 223	ï 239	ÿ 255

INDEX

Les mots clés PostScript, déjà catalogués dans le chapitre 10, ne sont pas repris dans cet index.

PostScript a longtemps été considéré comme un langage de très grande qualité, mais tellement ardu que sa manipulation devait être réservée aux robots assurant la communication entre logiciels de bureautique et imprimantes haut de gamme.

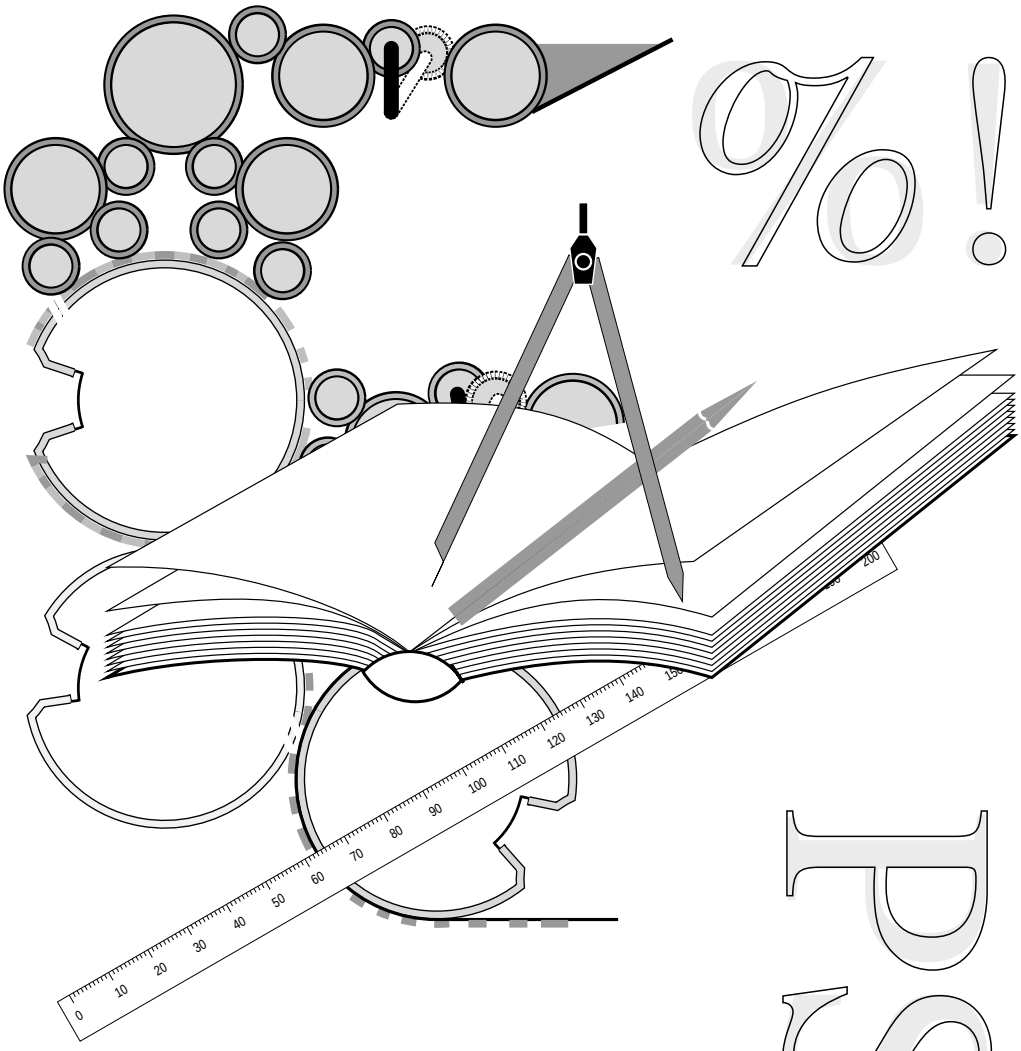
Ce langage n'est pourtant pas difficile. C'est même, parmi les langages graphiques, celui qui permet d'obtenir le plus rapidement les meilleurs résultats. Il faut simplement en comprendre toute la logique. C'est le but de ce livre.

La démarche peut intéresser tout possesseur d'un ordinateur qui désire réaliser à peu de frais du dessin de qualité. Ayant appris PostScript de manière rationnelle et progressive, il écrira ses programmes sous un éditeur banal et les soumettra au logiciel gratuit Ghostscript qui, en moins d'une seconde, lui affichera le résultat. En outre, PostScript est indépendant du système d'exploitation.

L'ouvrage rendra également service aux informaticiens désireux, non pas tant de programmer directement en PostScript, mais de comprendre et de corriger les produits des logiciels l'utilisant. Cette tâche est nettement plus ardue, car beaucoup de ces produits comportent des instructions dont l'étude dépasse le cadre de cet ouvrage ou bien n'est abordée qu'à la fin. Il lui faudra probablement continuer sa formation à l'aide de l'ouvrage de référence sur le sujet, ouvrage difficile certes, mais qui deviendra beaucoup plus compréhensible après l'apprentissage de base que propose ce livre.

PostScript

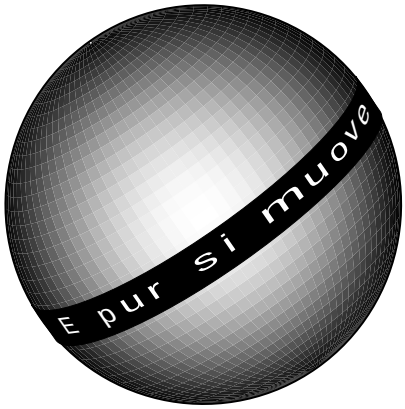
l'essentiel



%!PS

Pierre Blanc

TABLE DES MATIERES



Première édition : 1993
Seconde édition : 1998

AVANT - PROPOS

Le langage PostScript (PS) s'est imposé comme langage de composition de page pour les développeurs de logiciels d'impression. Il permet de coucher sur le papier à peu près tout ce que la typographie moderne est capable de faire. De plus, il autorise des dessins d'excellente qualité et, sur ce point, sert de référence aux meilleurs logiciels graphiques.

A l'origine, PostScript a été conçu pour des développeurs très spécialisés. Mais nombre d'informaticiens ont été séduits par les étonnantes qualités de ce langage et n'ont pas craint d'en explorer les méandres, malgré son abord un peu rébarbatif. On peut prévoir un engouement de plus en plus marqué pour PostScript, car le besoin d'agrémenter les textes par du dessin se fait de plus en plus sentir. Or on assiste depuis peu à deux phénomènes favorables : une baisse continue du prix des imprimantes et surtout la libre disposition d'un excellent interpréteur pour ce langage, à savoir **Ghostscript** associé en général à **GSview.** La programmation directe en PS est devenue tout à fait abordable et d'un coût en tout cas bien inférieur à celui d'un bon logiciel graphique commercial.

Après cette acquisition, le candidat PostScript devra parvenir à la maîtrise du langage, maîtrise qui lui donnera accès à une création de qualité "professionnelle". C'est pour l'aider dans cette quête que ce livre a été écrit. L'auteur a hésité entre une démarche pédagogique qui permettrait de progresser pas à pas vers la connaissance globale et l'écriture d'un ouvrage de référence qui fournirait d'emblée à son utilisateur toutes les ressources disponibles sur chaque aspect de la matière étudiée.

Pour utiliser une image bien connue du lecteur, un ouvrage pédagogique est *à accès série*, alors qu'un livre de référence est à *accès direct (aléatoire* comme disent les professionnels). L'auteur a voulu essayer de concilier l'un et l'autre. Sa démarche sera donc progressive, chaque chapitre nécessitant l'étude des précédents. Cependant, dès le début, seront citées des notions et donnés des exemples de haut niveau, dont la compréhension ne sera totale qu'après initiation. Ces passages seront signalés par une marque spéciale, le trèfle, symbole de la richesse – en PostScript s'entend . Il est donc conseillé d'ignorer en première lecture les passages marqués du symbole ♣ .

A part cette réserve, l'ouvrage ne suppose pas de connaissances spécialisées. L'habitude de la programmation des calculettes, surtout de celles en notation polonaise inverse, des notions de dessin vectoriel en HPGL seront des atouts précieux pour l'étude de PS. Mais rien de tout cela n'est nécessaire. Seuls les éléments de base de l'informatique seront supposés connus. L'auteur ne saurait trop conseiller à un lecteur néophyte de bien vouloir d'abord acquérir ces notions, par exemple dans son manuel *Initiation à l'Informatique* **(1)**. Ces deux livres sont d'ailleurs écrits dans le même style et le même esprit : favoriser l'accès du plus grand nombre aux techniques nouvelles.

Il va sans dire que l'aspect même de cet ouvrage pourra donner une idée sur les capacités de PostScript. Le texte a été saisi avec un logiciel de traitement de texte à sortie optionnelle PostScript (Word-5 sous DOS). Toutes les figures sont générées par des programmes PostScript importés dans des emplacements prévus à cet effet au sein du texte (des *réserves)*. Aucun travail de composition ou de mise en page n'a été nécessaire avant impression. C'est ça PostScript.

Quelques remarques sur le style des caractères employés : **l'écriture en caractères gras** signale bien sûr un passage important. Celle *en italique* attirera l'attention du lecteur sur une expression dont la signification n'est pas courante, mot étranger, mot propre à l'informatique ou à PostScript, ou bien variable mathématique. Les **textes en Helvetica** constituent des exemples d'instructions conformes à PostScript ; si, dans un tel texte, figure un mot en italique, c'est qu'il peut être écrit différemment : il s'agit d'une variable interchangeable avec un autre nom ou un autre objet permis par le langage.

La référence *Adobe* maintes fois citée renverra, pour plus de détails, au livre de base du PostScript **(2)** écrit par la firme Adobe elle-même.

L'auteur tient à remercier les personnes qui l'ont aidé dans sa collecte d'informations sur PostScript, l'imprimerie et l'imagerie informatiques. Ses remerciements s'adressent en particulier à M. Paul Vay, professeur à l'Ecole Supérieure de Papeterie et d'Industries Graphiques, à M. Jean-Marie Crochet, professeur au lycée technique André-Argouges, section imprimerie, à M. Alain Filhol, informaticien à l'Institut Laue-Langevin. Il exprime également sa gratitude à son épouse, Lisette Blanc, qui a bien voulu participer aux illustrations, ainsi qu'à tous ceux qui, d'une manière ou d'une autre, ont apporté leur concours à l'élaboration du présent ouvrage.

---

1 - *Initiation à l'Informatique*, P. Blanc, diffusion Internet.
2 - *PostScript Language Reference Manual*,
Adobe Systems Inc., Addison-Wesley, 1985.