Cahiers GUTenberg

LES FICHES CUISINE D'ONC POSTSCRIPT : 3. ÉCRIRE EN FRANÇAIS AVEC POSTSCRIPT

Bruno Borghi

Cahiers GUTenberg, nº 6 (1990), p. 60-62.

<http://cahiers.gutenberg.eu.org/fitem?id=CG_1990___6_60_0>

Les fiches cuisine d'Onc' PostScript Fiche n° 3: écrire en français avec PostScript

Bruno BORGHI

METASOFT, 13 rue Duhamel, 35000 Rennes

1. Introduction

Comme la plupart des langages informatiques, et bien qu'il soit conçu pour pouvoir imprimer tous les dialectes de l'univers, PostScript a des origines anglo-saxonnes suffisamment marquées pour que l'impression des lettres accentuées relève du casse-tête.

La raison en est simple : les polices standard de PostScript sont *encodées* conformément à la norme USASCII, qui, comme son nom l'indique, sert à écrire en américain.

Cette fiche cuisine vous donne une bonne recette pour éviter de faire l'impasse sur les signes diacritiques.

2. Changer le vecteur d'encodage

Chaque police de PostScript est munie d'un vecteur d'encodage : c'est tout simplement un tableau qui permet de faire la correspondance entre un code sur 8 bits et la description d'un caractère. Ce tableau n'est pas forcément complet : il est possible qu'un code ne corresponde à aucun caractère. Par contre, si l'on veut imprimer un caractère particulier, il est indispensable qu'un code lui ait été attribué au moyen du vecteur d'encodage.

Les polices standards utilisent le vecteur d'encodage standard `StandardEncoding`, qui ne contient pas les lettres accentuées. Pour écrire en français dans une police donnée, il faut donc en changer le vecteur d'encodage.

Vous avez pu découvrir la procédure de base dans le livre rouge à la section 5.7 : *User-defined fonts.* Lisez tout de même la suite : sa mise en œuvre est un peu délicate.

3. Au boulot

La pratique ordinaire de la programmation PostScript est, pour chaque problème, d'écrire une bonne fois pour toutes un prologue un peu général et donc réutilisable.

Avant d'écrire un prologue d'impression en français, il faut :

1. choisir un vecteur d'encodage ;
2. choisir un nom pour chaque police ainsi créée ;
3. choisir une méthode pour créer les polices réencodées.

3.1. Choisir un vecteur d'encodage

A priori, le vecteur d'encodage qu'il nous faut est à peu près quelconque. Il est tout de même préférable de garder les codes 0 à 127 tels qu'ils sont dans `StandardEncoding`, ce qui permet de rester compatible avec l'ASCII.

Les caractères « supplémentaires » vont donc se voir attribuer un code entre 128 et 255. Vous êtes tenté d'appliquer votre norme préférée (ISO, IBM PC, DEC, selon

les religions). *Sachez qu'en général cela ne marche pas.*

En effet, les signes diacritiques ont un code attribué dans `StandardEncoding` entre 193 et 200. Si on les supprime du vecteur ou si on les déplace, les caractères accentués ne sont pas imprimés correctement. Ne modifiez donc pas les codes entre 193 et 200. Cette contrainte est due à une bogue dans la définition récursive des caractères accentués. Elle sera (ou a été) peut-être corrigée un jour ; pour l'instant elle est encore présente dans la plupart des imprimantes PostScript.

Le fragment ci-dessous donne un exemple partiel de définition d'un vecteur d'encodage français.

```
[ StandardEncoding aload pop ]

dup 8#210 /Egrave put
dup 8#211 /Eacute put
dup 8#212 /Ecircumflex put
dup 8#213 /Edieresis put

dup 8#330 /egrave put
dup 8#331 /eacute put
dup 8#332 /ecircumflex put
dup 8#333 /edieresis put

/FrenchEncoding exch readonly def
```

Le vecteur d'encodage est construit à partir de `StandardEncoding`. Les codes correspondant aux caractères sont donnés en octal. Bien que cela semble bizarre, c'est en fait plus simple à cause de la syntaxe des chaînes de caractères PostScript : par exemple, Épée se note `(\211p\330e)`. Le vecteur d'encodage est défini en octal : plus tard, cela évitera le souci de convertir de tête des codes de décimal en octal.

Après la construction, on limite le droit d'accès de `FrenchEncoding` à la lecture seulement. Simple mesure de prudence qui permet de partager le même vecteur entre plusieurs polices sans risque de modifications ultérieures.

3.2. Choisir des noms de polices

Prenons par exemple la police `Courier`. La solution la plus immédiate est de nommer également `Courier` la version francisée de cette police.

C'est là aussi s'exposer à des problèmes : on perd en effet l'accès à la police de base, ce qui peut se révéler ennuyeux. De plus, et Onc'PostScript a pu lui-même le constater, l'interpréteur PostScript à bord de l'imprimante n'aime pas tellement cela et peut se bloquer dans certaines conditions.

Adoptons plutôt une politique de nommage homogène au moyen d'un préfixe ajouté au nom original. Ainsi, la version française de `Courier` sera `F-Courier`.

La correspondance entre les noms se fait au moyen d'un tableau bidimensionnel. Je n'en montre ici qu'une partie, le tableau complet pouvant reprendre toutes les polices de l'imprimante :

```
/FontList [
   [ /F-Courier /Courier ]
   [ /F-Courier-Bold /Courier-Bold ]
   [ /F-Courier-Oblique /Courier-Oblique ]
   [ /F-Courier-BoldOblique
      /Courier-BoldOblique ]
] def
```

3.3. Réencoder les polices

La procédure `ReencodFont` est tirée tout droit du livre rouge. Elle prend sur la pile le nom de la police originale et elle y met la police réencodée. La manip est relativement simple ; elle se résume en deux étapes : i/ on duplique la police ; ii/ on insère le vecteur d'encodage désiré. Ne pas oublier de *ne pas* dupliquer l'entrée magique `FID` qui caractérise de façon unique une police, et qui est

générée automatiquement par l'opérateur `definefont`.

```
/ReencodFont { % key ReencodFont font
   findfont dup length dict dup begin exch
      {
         1 index /FID ne
         {def} {pop pop} ifelse
      } forall
      /Encoding FrenchEncoding def
   end
} def
```

Lorsqu'on fait un prologue standard, on souhaite en fait que toutes les polices soient accessibles sous leur forme réencodée, sans demander explicitement ce réencodage. Cela permet en outre de ne pas réencoder trente-six fois la même police. La procédure `ReencodList` appliquée au tableau `FontList` de tout à l'heure remplit cet office.

```
/ReencodList { % fontlist ReencodList -
   { aload pop ReencodFont definefont pop
   } forall
} def
```

Cette solution convient dans certains cas. Cependant, lorsque l'on veut disposer de toutes les polices, elle s'avère chère. Il y a 33 polices à réencoder : cela prend un temps certain et dans la plupart des cas, c'est du temps perdu.

La bonne technique est alors de faire du *réencodage à la demande.* Le réencodage a lieu la première fois que l'on cherche cette police dans le dictionnaire des polices. Pour cela, on redéfinit ainsi l'opérateur standard `findfont` :

```
/std-findfont /findfont load def
/findfont {
   dup FontDirectory exch known not {
      FontList {
         dup 0 get 2 index eq {
            aload pop ReencodFont
            definefont pop
            exit
         } {pop} ifelse
      } forall
   } if
   std-findfont
} def
```

Lorsque `findfont` est appelé, il commence par examiner le dictionnaire `Font Directory` pour déterminer si la police demandée est connue. Si elle ne l'est pas, il effectue une recherche séquentielle dans le tableau `FontList`. Si le nom demandé est trouvé, la police correspondante est créée par réencodage. On termine par l'appel à l'opérateur `findfont` standard afin de faire le travail habituel : trouver le dictionnaire et, le cas échéant, générer une erreur lorsque le nom est invalide.

4. Conclusion

Vous pouvez maintenant, au moyen d'un prologue standard, imprimer en français sans acrobatie et sans dégradation de performances[1].

P.S. : Adobe vient d'annoncer la sortie de *PostScript Level 2*, la nouvelle génération du langage PostScript. De la couleur, une gestion mémoire améliorée et un nouveau mécanisme pour imprimer des formulaires. À suivre...

[1] Les fichiers utilisés par l'auteur sont disponibles en envoyant un mail à : `gut@irisa.fr`. *(ndlr)*