Cahiers GUTenberg

PUNK : DE METAFONT À POSTSCRIPT

Jacques André, Victor Ostromoukhov

Cahiers GUTenberg, nº 4 (1989), p. 23-28.

<http://cahiers.gutenberg.eu.org/fitem?id=CG_1989___4_23_0>

© Association GUTenberg, 1989, tous droits réservés.

L'accès aux articles des *Cahiers GUTenberg* (http://cahiers.gutenberg.eu.org/), implique l'accord avec les conditions générales d'utilisation (http://cahiers.gutenberg.eu.org/legal.html). Toute utilisation commerciale ou impression systématique est constitutive d'une infraction pénale. Toute copie ou impression de ce fichier doit contenir la présente mention de copyright.

PUNK : De METAFONT à PostScript

Jacques ANDRÉ† et Victor OSTROMOUKHOV‡

† *IRISA, Campus de Beaulieu, F-35042 Rennes Cedex, France*
‡ *EPFL-INF/LSP, Ecublens, CH-1015 Lausanne, Suisse*

Résumé La police PUNK, définie par D. Knuth en METAFONT, a été traduite automatiquement en PostScript, ce qui lui confère des propriétés encore plus dynamiques grâce au mécanisme de création de fontes en PostScript.

Abstract *The PUNK font, initialy designed by D. Knuth for METAFONT, has been translated into PostScript. Thanks to the PostScript font machinery, this new font is more dynamic than the genuine one.*

Depuis 500 ans environ, on ne connaît plus que deux types de caractères : les caractères manuscrits qui ont encore gardé leur *liberté* (laquelle, lorsqu'elle est particulièrement mise en valeur, conduit à la calligraphie) et les caractères typographiques (en plomb, photocomposés ou numérisés) qui sont immuablement figés dans leurs dimensions et leur forme (malgré leur nom de caractères « mobiles » !).

C'est probablement D. Knuth qui, le premier, a proposé la notion de méta-fonte [Knuth 82] puis décrit ce que nous pensons être la première police de caractères dynamique : PUNK. Nous avons porté cette police en PostScript, ce qui permet de la rendre encore plus dynamique.

1. La police PUNK

La police PUNK a été conçue par Knuth un peu avant 1980 mais n'a été publiée que récemment [Knuth 88]. Il s'agit d'une police de caractères inspirée des graffiti « punks » que l'on pouvait trouver sur les murs des grandes villes à la fin des années 70, mais aussi dans certaines œuvres de Picasso ou de Dali. Il s'agit d'une écriture formée de bâtons aux extrémités desquels Knuth met un gros point (comme un gros coup de brosse ronde de peintre en bâtiment). La figure 1 donne quelques exemples de caractères PUNK. L'originalité

POLICE PUNK
DON KNUTH

Figure 1 : Exemples de caractères PUNK

de cette police vient d'une part de l'aspect irrégulier des tracés comme le fait quelqu'un peignant à la hâte ces caractères sur un mur (mais ceci existe déjà dans certains caractères, par exemple *revolution* de Lettraset) et d'autre part, et surtout, du fait que ces irrégularités sont produites par des fonctions aléatoires lors de la création d'une occurrence de police PUNK :

La capitale « A » est définie en METAFONT comme suit :

```
beginpunkchar("A", 13, 1, 2);
z1=pp(1.5u, 0); z2=(.5w, 1.1h);
z3=pp(w-1.5u, 0);
pd z1; pd z3;
draw z1--z2--z3;
            % left and right diagonals
z4=pp.3[z1,z2]; z5=pp.3[z3,z2];
pd z4; pd z5; draw z4--z5;   % crossbar
endchar;
```

où pp est définie par

```
vardef pp expr z=z+(hdev*normaldeviate,
            vdev*normaldeviate) enddef;
```

c'est-à-dire en utilisant la fonction aléatoire `normaldeviate`.

A moins d'utiliser toujours le même germe pour la génération des nombres aléatoires, chaque fois que l'on installe une police PUNK (pour un corps donné), on crée un nouveau jeu de caractères.

2. Pourquoi utiliser PostScript?

Puisque les fonctions aléatoires sont utilisées lors de la création de la police, tous les caractères restent, une fois créés, identiques. Ceci se vérifie aisément dans le texte de la figure 1 où les « P » de la première ligne sont semblables et où les « N » de la seconde aussi ; par contre le « U » de la première ligne diffère de celui de la seconde car il ne s'agit pas de la même police (la première est en romain normal, la seconde en gras). Grâce à la *font machinery* de PostScript [Adobe85, p. 85 sqq], on peut utiliser une fonction aléatoire non plus lorsque l'on construit globalement une police pour un corps et des orientations donnés (c'est-à-dire par les instructions `scalefont` ou `makefont` et `setfont`), mais lorsque l'on utilise un caractère (instructions `show`). En effet, cette machinerie dispose de deux modes : un mode accéléré (qui est le mode usuel) et un mode simple (mais pouvant être 1000 fois plus lent que le mode normal !) :

- En mode normal, le processus d'écriture peut être décrit comme suit :
  Au premier appel d'un caractère d'une police donnée, par exemple `(A) show`, on appelle la procédure `BuildChar` qui doit obligatoirement exister dans le dictionnaire décrivant la police en cours. Cette procédure passe les informations sur la métrique du caractère en utilisant la procédure `setcachedevice`. Puis `BuildChar` exécute l'algorithme décrivant le caractère, ce qui produit la matrice de points du caractère « A » pour la police en cours et pour l'imprimante utilisée. Cette matrice est alors mise dans la page en cours et (c'est le second rôle de `setcachedevice`) une copie est gardée en mémoire cache.
  Lors d'un nouvel appel `(A) show` (ou lors d'une seconde occurrence de ce même caractère « A » dans une chaîne, p.ex. `(AA) show`), on prend directement la matrice de points en mémoire cache pour la copier dans la page en cours.
- En mode simple :
  au premier appel de `(A) show`, on appelle `BuildChar` qui passe les informations métriques en utilisant la procédure `setcharwidth`. La matrice de points est alors calculée et mise dans la page courante. Mais elle n'est pas mise en mémoire cache.
  Lors d'un nouvel appel `(A) show`, on refait tous les calculs comme au premier appel.

Cette seconde méthode est évidemment beaucoup plus longue (les algorithmes de *scan conversion* sont très coûteux), mais elle permet d'appeler une fonction aléatoire à chaque occurrence du caractère « A » et non plus une fois pour toutes. Diverses polices dynamiques ont été produites ainsi [André & Borghi 89a,b, Packard 89].

3. De METAFONT à PostScript

Il était donc intéressant d'essayer de profiter du dynamisme de PostScript tout en gardant la description METAFONT de PUNK. Mais il faut souligner, tout d'abord, que la traduction METAFONT→PostScript n'est possible en principe que grâce à une « heureuse coïncidence » : les deux langages manipulent la même classe d'objets, effectuant sur eux les mêmes opérations [Ostromoukhov88]. Les tableaux 1 et 2 résument la situation.

Objets	METAFONT	PostScript
point	pair	moveto
ligne	z1- - z2	lineto
Courbe de Bézier	z1..z2	curveto
plume ronde	pencircle	setlinewidth
transformation	transform	CTM

Opérations	METAFONT	PostScript
Remplissage	fill	fill
tracé de plume	draw	stroke
effacement	erase	1 setgray fill
appliquer une transformation	transformed	translate, scale, rotate

Il existe néanmoins quelques différences : par exemple, METAFONT peut manipuler des plumes de forme quelconque, tandis que PostScript ne sait pas le faire. Une autre différence significative consiste en l'impossibilité d'effectuer en PostScript l'opération d'effacement lors de la construction d'un caractère, après la commande `setcachedevice`. La technique d'effacement est couramment utilisée par les programmes METAFONT. Ces différences restent tout de même non-significatives pour la conversion METAFONT→PostScript quand les sources METAFONT sont un peu arrangés.

4. Traduction de PUNK

Nous possédons donc une implémentation du programme METAFONT qui effectue une telle conversion METAFONT→PostScript automatiquement. Le processus de génération de la police PUNK dynamique est divisé en deux phases :

1. Dans une première phase, nous avons supprimé dans les sources METAFONT tout aspect aléatoire : la tâche de perturbation sera entièrement confiée à PostScript. Nous avons produit ainsi la version « régulière » de PUNK.
2. Dans une deuxième phase, nous avons remplacé les primitives `lineto` et `curveto` de PostScript par des expressions plus compliquées contenant un élément aléatoire et paramétré par les variables dédiées.

Exemple, pour la version régulière :

```
/L{lineto}def
/M{moveto}def
/C{curveto}def
......
11.71527 0 M
51 270.6015 L
35.43878 6.29214 32.93958 12.32655
                  28.4907 16.77544 C
......
```

et pour la version dynamique :

```
/HalfMax 1073741824 def % 2^30
/LinetoRatio   66000000 def
              % variable dediee
/SmallCurvetoRatio 500000000 def
              % variable dediee
/L{ exch rand HalfMax sub
    LinetoRatio div add
    exch rand HalfMax sub
    LinetoRatio div add lineto}def
/M{ exch rand HalfMax sub
    LinetoRatio div add
    exch rand HalfMax sub
    LinetoRatio div add moveto}def
/C{
    6 -1 roll rand HalfMax sub
    SmallCurvetoRatio div add 6 1 roll
    5 -1 roll rand HalfMax sub
    SmallCurvetoRatio div add 5 1 roll
    4 -1 roll rand HalfMax sub
    SmallCurvetoRatio div add 4 1 roll
    3 -1 roll rand HalfMax sub
    SmallCurvetoRatio div add 3 1 roll
    exch      rand HalfMax sub
    SmallCurvetoRatio div add exch
          rand HalfMax sub
    SmallCurvetoRatio div add
    curveto}def
......
11.71527 0 M
51 270.6015 L
35.43878 6.29214 32.93958 12.32655
                  28.4907 16.77544 C
......
```

Nous avons produit ainsi la version dynamique de la police PUNK : chaque lettre est différente des autres à chaque occurrence, puisque chaque caractère est recalculé à nouveau lors de chaque invocation, avec un élément aléatoire.

Les figures 2 et 3 présentent les mêmes caractères ; mais, dans le premier cas, il

ABCDABCDABCDABCDABCD

PORTAGE DE POLICE PUNK EN POSTSCRIPT

PORTAGE DE POLICE PUNK EN POSTSCRIPT

PORTAGE DE POLICE PUNK EN POSTSCRIPT

AAAAABBBBBCCCCCDDDDD

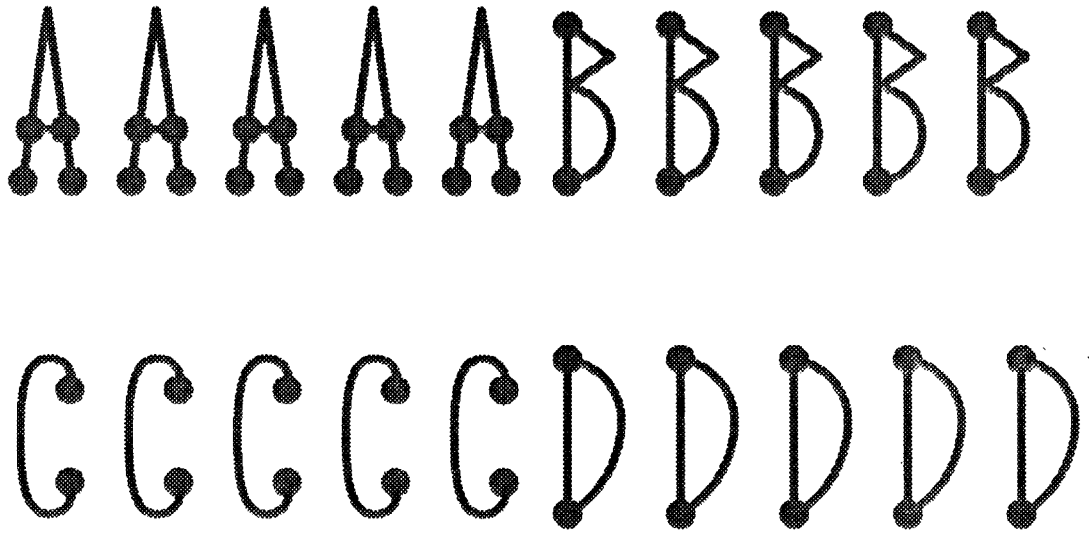

Figure 2 : Caractères PUNK vus par PostScript – version régulière.

ABCDABCDABCDABCDABCD

PORTAGE DE POLICE PUNK EN POSTSCRIPT

PORTAGE DE POLICE PUNK EN POSTSCRIPT

PORTAGE DE POLICE PUNK EN POSTSCRIPT

AAAAABBBBBCCCCCDDDDD

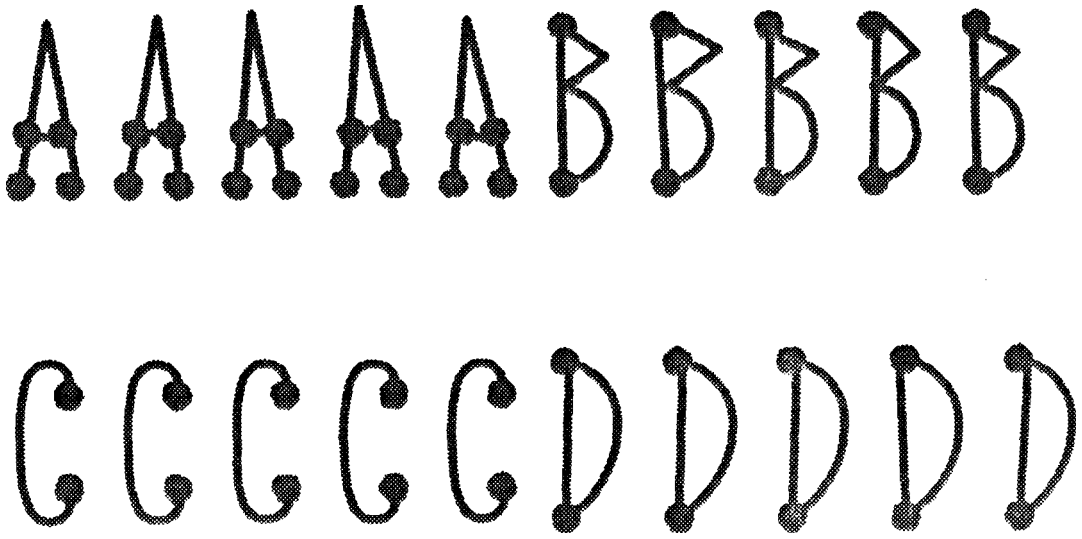

Figure 3 : Caractères PUNK vus par PostScript – version dynamique.
Il y a plus de pertubations sur, par exemple, `curveto`. Notez que les « A » sont tous différents les uns des autres.

s'agit de la version « régulière », tandis que, dans l'autre, il s'agit de la version dynamique.

5. Conclusion

Nous sommes conscients que les programmes PostScript générés par le procédé décrit ci-dessus sont assez naïfs. Nous laissons au lecteur le soin de rendre le code plus sophistiqué, s'il le juge nécessaire, notre but ici étant seulement de démontrer la faisabilité de l'opération.

On trouvera dans [André & Borghi 89a] d'autres applications de ce dynamisme. Mais nous sommes conscients aussi que l'intérêt n'est pas évident pour tout le monde. Certains typographes crient au scandale dès que l'on touche à l'immuabilité du caractère (voir par exemple les expériences de [Laufer 87a et b] à ce sujet). Pourtant le monde où nous vivons est déterministe ; il est donc normal d'accepter, et donc de simuler, sa complexité.

Références

[Adobe85] ADOBE SYSTEMS INC., *PostScript Language Reference Manual*, Addison Wesley, Reading, Mass., 1985.

[André Borghi 89a] , Jacques ANDRÉ and Bruno BORGHI, "Dynamic fonts", in *Raster Imaging and Digital Typography* (André & Hersch eds.), Cambridge University Press, 1989, 198-204.

[André Borghi 89b] , Jacques ANDRÉ and Bruno BORGHI, "Dynamic fonts", in *PostScript Language Journal*, vol.2, No.3, december 1989, 6–8.

[Knuth 82] D. KNUTH, "The concept of a metafont", *Visible Language*, vol. XVI, no 1, 1982, 3-27. Traduction française : « Le concept d'une meta-fonte », *Communications et langages*, 1984, 55–132.

[Knuth 88] , Donald E. KNUTH, "A PUNK MetaFont", *TugBoat*, vol. 9 (1988), No 2, 152-168.

[Laufer 87a] Roger LAUFER (textes réunis par), *Le texte en mouvement*, Presses Universitaires de Vincennes, 1987.

[Laufer 87b] Roger LAUFER, « Calligraphie synthétique animée », *culture technique* numéro 17, mars 1987, 273–275.

[Ostromoukhov 88] Victor OSTROMOUKHOV, "METAFONT vs. PostScript", *Proceedings of EuroTEX conference – Exeter '88* (à paraître).

[Packard 89] Ted PACKARD, "Ransom Font", *The PostScript Journal*, vol. 2, no 2, July 1989, 44–45.