Dynamic fonts

Jacques André† and Bruno Borghi‡

† *INRIA/IRISA Campus de Beaulieu*
F-35042 Rennes, France
‡ *Metasoft*
13 Rue Duhamel
F-35000 Rennes, France

ABSTRACT : Dynamic fonts are fonts whose character shape is defined every time the corresponding character is printed rather than when the font is defined as a whole. Such fonts allow, for example, random characters (such as graffiti), context dependencies (as in logo design and calligraphy), or character extension (as in the justification of semitic text).

KEYWORDS: Dynamic fonts, PostScript, PUNK, print-time.

1 Introduction

Characters belong to one of the two following classes:

Static fonts The characters are designed, then cut or digitized and finallly used in a printing process. Almost all fonts belong to this class. Example: the Times Roman font (see figure 1, where all 'A's have the same shape).

AAAAA

Figure 1: Static font: Times-Roman – All 'A's are identical

Included in this class are random fonts such as PUNK designed by Donald Knuth [Knuth88]: A meta-font [Knuth82] is defined, the characters are digitized with a shape that depends on parameters defined at compile time, then they are used in a printing process.

Figures 2.left and 2.right show roman-PUNK and bold-PUNK fonts: In each font, the "A"s have the same shape, although since the two fonts were generated with random numbers they produce "A"s of different shapes.

Figure 2: Static random fonts: PUNK-Roman (left) and PUNK-Bold (right) fonts – Note that the "A"s are identical within each font but differ from one font to the other.

Dynamic fonts On the other hand, in this class of font the characters are redefined at each instantiation (every time they are printed) rather than when the font is defined as a whole. The "A"s of Figure 3 come from the same font: however, because each of them has been computed at print time with random numbers, they are all different.

Similarly, all handwritten "fonts" (handwritten alphabets, calligraphic letters, graffiti, and even hand-printed capitals) are dynamic because of individual variations.

Figure 3: Dynamic font: Although these characters were generated with the single PostScript "`(AAAAA) show`" instruction, they are all different.

2 Dynamic fonts and PostScript

METAFONT, being a batch font design system [Knuth85], does not allow PUNK to be a dynamic font. On the contrary, PostScript [Adobe85bgr] does support dynamic fonts. In fact, two operators control the PostScript font machinery: `setcachedevice` requests the machinery to transfer the bitmap of each character into cache memory (from where it will be retrieved by using the `show` operator) while `setcharwidth` states that the bitmaps are not to be placed into the font cache. Thus, each time a given character is to be printed using a `show` instruction its bitmap will be fully computed.

Dynamic fonts may be divided into three classes, according to the type of information exchanged:

- the fonts are self-sufficient, no information needs to be passed on. The dynamic aspects of such fonts are governed exclusively by random functions.
- the character shapes depend on a limited number of read-only parameters, such as width expansion.

- the character shapes depend on a large number of parameters, i.e. a context, and conversely their design may modify the context.

3 Random dynamic fonts

Random functions may be used to produce random lines or random curves when designing a character. Refer to figure 3 for an example.

Random functions may also be used to define geometrical transformations on a character. For example, figure 4 was generated with a character definition calling a function to rotate each character by θ degrees where θ is randomly defined.

Figure 4: Scrabble, another dynamic font

Besides graffiti and the like, such fonts may be used to produce characters for the testing of recognition systems for hand-printed characters [Suen80].

4 Associating parameters

Semitic languages do not justify text in the same way that European ones do. As a first approach, it can be said that instead of expanding spaces, some long character forms are extended. Figure 5 shows the two justification methods, discounting cultural considerations.

Extending tooth-letters is an easy process with PostScript: the coordinates of control points defining the Bézier splines that produce curved letters have to be modified depending on the expected justification. It works if the font is dynamic and the expansion value is conveyed to the font machinery, e.g. through the stack. Figure 6 shows such dynamic modifications on a letter looking like an arabic letter (similar to a *shin*). The PostScript call is "$\delta x\ \delta y$ `(s) show`".

the room
the room

Figure 5: Two methods of justification: the European method (top – spaces are expanded) versus the Semitic method (bottom – certain long forms are expanded, here the arches of the final "m".)

The aim of this example is not to offer new designs for character shapes, nor to modify existing ones, but rather to show that shapes can be defined at print time instead of offering a limited set of expanded typefaces. Today, it seems that no software is capable of composing Hebrew or Arabic texts properly. However, a few additions would make TEX (at least its bi-directional version [Knuth & MacKay87]) a candidate for such languages.

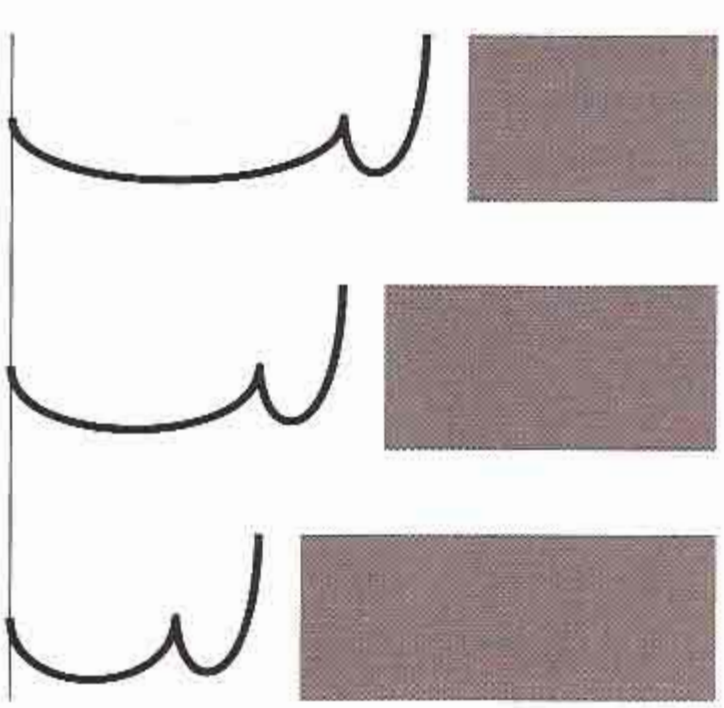
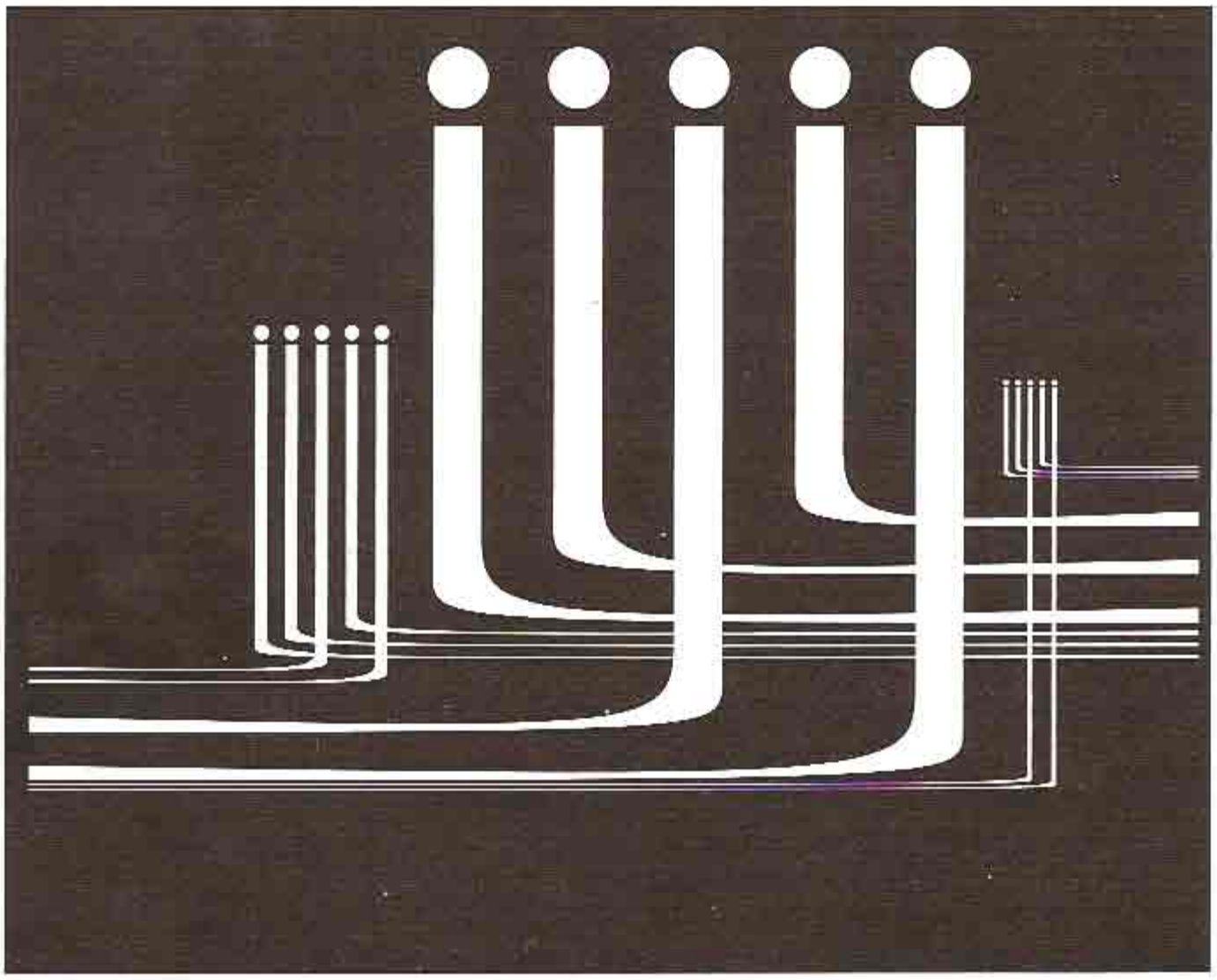


Figure 6: Left dynamic extension of an arabic-like tooth letter; grey part indicates the beginning of the line.

5 Context dependence

The third class of dynamic letters requires a two-way exchange of information between a context (i.e. a set of variables) and the characters. The PostScript dictionnary concept allows information to be sent to and from the font machinery. Figure 7 shows a piece of "calligraphy" printed in PostScript with a set of single "`(iijij) show`" instructions once the overall parameters (left and right limits, distance between horizontal lines etc.) have been passed to the font. The PostScript program is given in the Appendix.



6 Conclusion

Why such fonts? First to reproduce the complexity of the real world, which is non-deterministic (e.g. to simulate handwritten characters). Secondly, to revive the old tradition which sometimes allowed typesetters to use various (clearly discrete) letter widths (e.g. some types designed and cut by Rudolf Koch). And thirdly, to allow character designers to invent new signs (one dares not call them letters!) however much classically-minded designers and typographers dislike the idea [Laufer87].

References

[Adobe 85b] Adobe Systems Inc., *PostScript language tutorial and cookbook*. Reading, Mass.: Addison Wesley, 1985.

[Adobe 85r] Adobe Systems Inc., *PostScript language reference manual*. Reading, Mass.: Addison Wesley, 1985.

[Adobe 88g] Adobe Systems Inc., *PostScript language program design*. Reading, Mass.: Addison Wesley, 1988.

[Karow 87] Peter Karow, *Digital formats for typefaces*. Hamburg: URW Verlag, 1987.

[Knuth 82] Donald Knuth, "The concept of a meta-font", *Visible language*, vol. 16 no. 1, 1982, pp. 3–27.

[Knuth 88] Donald Knuth, "A punk meta-font", *TUGboat*, vol. 9 no. 2, August 1988, pp. 152–168.

[Knuth & MacKay 87] Donald Knuth & Pierre MacKay, "Mixing right-to-left texts with left-to-right texts", *TUGboat*, vol. 8 no. 1, April 1987, pp. 14–25.

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

[Laufer 87b] Roger Laufer, "Calligraphie synthétique animée", *Culture technique*, no. 17 (numéro spécial *Electricité, Electronique, Civilisation*), March 1987, pp. 273–275.

[Suen *et al.* 80] Ching Y. Suen, Marc Berthod & Shunji Mori, "Automatic recognition of handprinted characters: the state of the art", *Proc. IEEE*, vol. 68 no. 4, April 1980, pp. 469–487.

Appendix: A PostScript Program for Dynamic Fonts

```
% we follow the structure given in the example
% ''Building a new font'' in [Adobe85r] -
% obvious definitions have been skipped

%...
/FontBBox [ 0 0 0 0 ] def     % a dynamic font has no fixed BBox
%...

% the definition of the letter i in the CharProcs dictionary

% Dynamic data are recorded in the Context dictionary.
% These data are the Rx and Ry device space coordinates
% of the current lower right limit of dynamic expansion.

% Use of transform and itransform allows an absolute reference
% to be kept independently of the scale of the font.
% However, variation of the reference between instantiations
%  is proportional to the scale.
```

```
/i -
250 0 setcharwidth                          % NOT setcachedevice
75 0 translate                              % left sidebearing

Context begin                               % get context
Rx Ry itransform                            % Rx Ry in the device space
/y exch def /x exch def                     % x y in the character space

gsave
newpath 50 700 63 0 360 arc closepath       % the dot on the i
100 600 moveto 0 600 lineto                 % top of the body
0 y translate                               % translate to bottom
0 100 lineto                                % left side of the body
0 0 0 0 x 0 curveto                         % bottom of expansion
0 30 rlineto                                % right side of expansion
100 0 100 0 100 200 curveto                 % top of expansion
closepath                                   % right side of the body
fill
grestore

x y 100 add transform                       % update context (back
/Ry exch def /Rx exch def                   % i is stepping up
end      % Context
~ def
%...
/Dynamic newfont definefont pop

% Before the first use of the font, Context must be initialized

/Context 4 dict def                 % room for Rx, Ry, x, y
Context begin
        500 -100 transform
        /Ry exch def /Rx exch def   % lower right limit
                                    % of the dynamic expansion
       end      % Context
% Now, try it
/Dynamic findfont 50 scalefont setfont
100 0 moveto (iii) show
```