Towards the revival of traditional Arabic typography through TeX

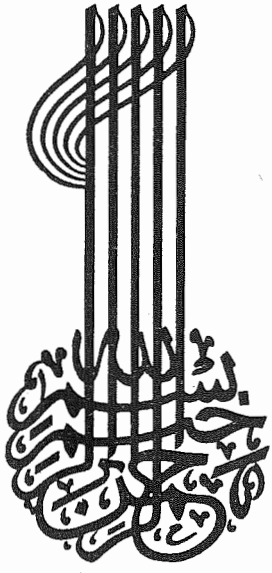
Yannis Haralambous

Towards the revival of traditional Arabic typography... through TEX

Yannis Haralambous

Abstract

The goal of this new Arabic TEX system is the revival of traditional Naskhi style typography, by keeping the most simple and natural input, and the most of compatibility with TEX macro packages. Arabic typesetting is done in two ways: (a) without pre-processor for non vocalized text; (b) by means of a pre-processor for vocalized text. Both ways can be mixed in the same text. The new font, "Al Amal", features more than a 1,000 ligatures as well as *all* special signs and symbols needed for typesetting the Quran.



Bismillahi-Rahmani aRahim
In the name of Allah, the Gracious, the Merciful.

1 Apologies

At the date this paper is written (August 11, 1992) the items presented are still under development. This is the reason why the reader will not find a complete list of characters of the Arabic font

"Al Amal", nor a text example. Only the main ideas and goals will be presented. The author hopes that by the time of the EuroTEX 92 Conference, he will be able to distribute a complete version of this paper and would like to apologize for the inconvenience... *ars longa, vita brevis!*

2 Main ideas

Before speaking of the "Al Amal" project (the word meaning *hope* in Arabic) let's see what the real problems of Arabic typesetting are. First of all we can divide these problems into two categories: problems related with the computer, and problems of traditional Arabic typography. Of course one may argue that the first kind of problems (like encoding, input etc.) has it's counterpart in traditional typography, but describing these analogies would lead us too far. Problems arising when typesetting Arabic by the means of a computer are the following:

1. direction of Arabic script;
2. contextual analysis;
3. input (encoding, transliteration).

On the other hand, problems already present in traditional Arabic typography are:

1. spacing (no hyphenation allowed);
2. rules concerning ligatures, special character forms and special Quranic symbols;
3. positionning of vowels and other diacritics.

2.1 Direction of Arabic script

The first and most trivial problem is the direction of Arabic script, which happens to be the opposite of the one you are using now to read this text: Arabic is written from right to left. And the problems really begin, when Arabic and Latin scripts are mixed: the reader can meditate on the possible embarrasing situations which may occur when Arabic sentences in the middle of Latin text have to be splitted, or the inverse.

Fortunately these problems have been solved in a very elegant way, by a special TEX version called `TeX-XeT` and written by

the Grand Wizard D. E. Knuth in person, with the help of Pierre MacKay and very recent changes by Peter Breitenlohner. Unfortunately many (uninformed) people consider this version of TEX to be specific to bidirectional text and do not realize that it is completely upwards compatible with TEX, and produces *regular* DVI output. The author would like it to be a point of honour for every implementor to include TeX-XeT in his standard TEX implementation.

TeX-XeT uses two primitive pairs: \beginL, \endL, and \beginR, \endR to establish the "mode" of a text, and of hierarchically ordered parts of this text. A publication in English for example will be in left-to-right mode (or "L-mode" in TeX-XeTs jargon). An Arabic (or Hebrew) quotation will be in right-to-left mode (or "R-mode") inside this text. A Latin word inside the quotation will again be in L-mode inside the R-mode, and so forth.The author has used the standard language-switching macro \AR to enter into R-mode, and added the switch to L-mode, to the usual standard two-letter ISO language switching control sequences (like \US, \CZ, etc).

2.2 Contextual analysis

The next problem faced when typesetting Arabic is contextual analysis. Arabic letters can appear in four forms: isolated, initial, medial and final. Some letters can have only two forms: isolated and final; one letter (the *hamza* without carrior) can only appear in an isolated form.

Here is the first main and innovative idea of the "Al Amal" system:

> **Idea 1.** Contextual analysis is entirely performed by TEX's "smart ligatures".

Two of these ligatures are very important in Arabic, while they are only rarely used in other cases: the begin-of-word and end-of-word ligatures. The former allows the font designer to modify a character when it happens to be at the beginning of a word; the latter allows modification "at the end of a word", that is in front of certain user-defined non-alhabetical characters[1].

[1] This feature has been used in Greek, Hebrew and Old German

By using smart ligatures to perform contextual analysis, the Al Amal font is completely independent of any preprocessor or macro package. Arabic can be included into any complicated TEX construction (mathematics, tables, footnotes etc) without any risk of conflict[2]. To write in Arabic or Persian, the user has only to use the standard control sequence `\AR`, and to input his text in a fixed transliteration; all the rest is done at the font level.

Here is an example of the use of smart ligatures for a small word: `vgGplR`, where `v`, `g`, `G`, `p`, `l`, `R` stand respectively for *va, dzim, gaf, pe, lam* and *ghayn*. The general rule is that characters in positions `v`, `g`, `G` etc are medial characters, or final if there is no medial form. Hence `v` produces a medial va. But since `v` is at the beginning of the word, a begin-of-word ligature replaces the medial va by an initial va. `g`, `G`, `p`, `l` produce medial letters, which are left unchanged. Finally `R` produces a medial ghayn which is at the end of the word: the end-of-word ligature converts it into a final ghayn. Here is the result in the "Al Amal" font. The reader may skip a few sections to see the same word fully ligatured.

2.3 Input

Another problem is the input of Arabic. Here comes the second main idea of "Al Amal":

Idea 2. Arabic text input is done in **two** ways:

[2] There is one exception though: the `\uppercase` macro has to be avoided since Arabic input uses differently uppercase and lowercase characters...

1. directly, using a fixed transliteration, for non-vocalized text;
2. using macros; the names of these macros being names of characters and diacritics.

To see the reason for this dichotomy, one may simply take a look into Arabic books, newspapers, journals etc. Whenever modern Arabic is used, vowels and diacritics appear very rarely, except a few standard ones like the shadda, madda and the nunation diacritics. In this case, TEX input should be as easy as possible; and should never interfere with other TEX macro packages, like LATEX, LAMSTEX, etc. One should be able to use Arabic words inside a LATEX table, or inside a math formula, as easy as Latin text. This is achieved thanks to TEX's smart ligatures described in previous section. Example:

```
    label{nominal}
\DE Danach sind \AR Al"talb yktb \DE und
\AR Al"talbT tfhm \DE {\it Nominals\"atze},
dagegen \AR yktb al"talb \DE und \AR tfhm
al"talbT \DE {\it Verbals\"atze}.
```

On the other hand, if one looks into scholarly publications or Quranic texts, one will discover that virtually every letter has at least one diacritic, which may be a vowel (fatha, kasra, damma), the absence of vowel (sukun), duplication (shadda), nunnation, or other diacritics (madda, wesla, etc.) It is impossible for TEX to perform contextual analysis when these diacritics are present. To solve this problem, the opposite solution has been chosen: everything is specified in form of macros with arguments. Each letter, or ligature, is one macro, whose first argument corresponds to the contextual form, and the subsequent arguments (one for a single letter, n for a n-letter ligature) specify the diacritics. Example:

```
%@ARABIC
\waw3{\fatha}
\kaf1{\fatha}\dhal3{\fatha}\smallalif0{}\lam1{\kasra}\kaf3{\fatha}
\nun1{\damma}\fa2{\fatha}\sad2{\shaddakasra}\lam3{\damma}
\alif3{\wesla}\lamalifhamza0{\sukun}{\fatha}\ya1{\fatha}%
\smallalif2{}\ta3{\kasra}
\waw3{\fatha}
\lam1{\fatha}\ayn2{\fatha}\lam2{\shaddafatha}\ha2{\damma}\min3{\sukun}
```

```
\ya1{\fatha}\ra3{\sukun}
\dzim1{\kasra}\ayin2{\damma}\waw3{}\nun0{\fatha}
%@LATIN
```

Pre-processors can be used to convert input from different sources into the "Al Amal macros" and back. These sources can be transliterated Arabic text using arbitrary transliteration, or direct Arabic input using an Arabic hardware system. These macros are written in 7-bit standard TEX (which makes them email-portable) and their expansion can be changed according to the used font. So for example if a font does not contain the "kaf-nun-alif" ligature (which is the general case, since "Al Amal" is the only font providing this rare and beautiful traditional Arabic ligature), then the macro

```
\kafnunalif3{damma}{fatha}{}
```

will expand into

```
\kaf2{damma}\nun2{fatha}\alif3{}.
```

The `%@ARABIC` and `%@LATIN` are comments for the pre-processor which will by this be able to go back and convert the TEX code between the comments into transliterated Arabic, or system-dependent Arabic etc.

2.3.1 A short comment

As the reader may have realized, up to now the "Al Amal" system together with TEX and the adequate pre-processors is *"doing all the work"*. Another way to express this, is to say that the user up to this moment is doing a mechanical job of inputting Arabic text (either on the keyboard, or by converting some pre-existing data) without any further effort. This work may be tedious, but is straightforward (and certainly leaves a lot of work to TEX and the pre-processor).

Now we can examine the kind of problems which traditional Arabic typographers of the last centuries had (and have) to face. It is time for the third main idea of "Al Amal":

> **Idea 3.** If we face a traditional typographical problem, then we shall make TEX follow the mental process of the traditional Arabic typographer to solve it.

This sounds like an extravagant abstraction (or a meaningless triviality), but we will see some concrete examples where this idea leads to efficient results.

2.4 Spacing

Arabic allows no hyphenation. The reader is invited to take some Latin-alphabet text and to typeset it while prohibiting hyphenation (this can be done very easily by changing `\language` to some value for which no hyphenation patterns have been loaded). A lot of underfulls and overfulls will fatally occur (especially if the text is in German or Turkish or any other agglomerating language).

To solve this problem, Arabic typographers are introducing line segments between attached letters. Traditionally these segments are placed before the last letter of a word (some other restrictions apply). TEX can do what the dream of any Arabic typographer is: it can adjust the sum of lengths of these segments to be exactly the amount of underfull, and then divide by the number of the segments so that they are all of equal length.

The reader may wonder how this magic operation can be performed; actually it is very simple (another proof of the power of TEX): each line segment is just the macro `\hrulefil` (the analogous of the TEXbook's `\hrulefill`, but with one 'l' like in `\hfil`) followed by the primitive `\noboundary`). The rule dimensions are computed by a font parameter which gives the width of a generic horizontal stroke of the Arabic font (all Arabic script is based on a generic horizontal stroke). Why one 'l'? Because at the end of a paragraph, when a line is not to be justified, one simply needs to compensate by an `\hfill` with two 'l's. The `\noboundary` primitive is necessary to continue the contextual analysis. Otherwise TEX will consider the line segment to be the end of the word and the forthcoming letter will be of isolated form.

Of course perfection is sometimes ... too perfect and one would like to correct it. This is possible by using a fixed-length line segment character included in the font. But this character has another very important function: sometimes, because of their design, letters come too close. In these cases one just need to insert a small line segment between them, to arrange the global

appearance of the word. This again is done automatically by the font, using TEX's smart ligatures: suppose you have a pair of letters $l_1 l_2$; you can define a ligature in the following way: "l_1 *followed by* l_2 *shall be replaced by* $l_1 l_3$ *followed by* l_2". The author strongly believes that we are only starting to exploit the amazing possibilities of TEX's smart ligatures.

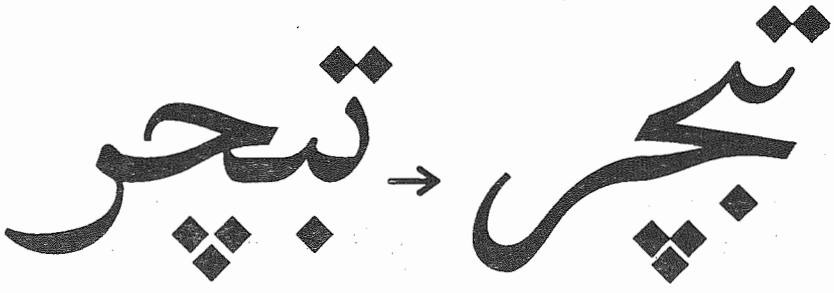
2.5 Rules concerning ligatures, special character forms and special Quranic symbols

There is a (not too large) amount of specialized books which list the complete set of rules of traditional Naskhi typography. These rules—like the shapes of typographic Naskhi characters—are inspired by calligraphy, but are both accurate and manifold enough to give a dynamic and easy readable Naskhi text. Unfortunately about 90% of these rules have been ignored by digital typography. One of D. E. Knuth's motivations while creating TEX was to ensure the revival of traditional mathematical typography through the computer; **the goal of "Al Amal" is the revival of traditional Arabic typography!**

Let's take a closer look to these rules. One can roughly subdivide them into two categories: ligatures and special forms. The ligatures can include as much as four letters (for example ba-ta-dzim-mim). Special forms of characters arise in certain situations: for example "each time there are three consecutive letters of the {ba, ta, tha, ya, pe} family, the middle one must have a higher form", or "when a final letter of the {ra, za, zhe} family is preceded by some letter of the set {hha, kha, dzim, tshim, tta, zza, ayn, ghayn, fa, qaf, va, kaf, gaf, ha} it takes a special calligraphic form" (see next illustration) etc. The author will publish these rules in the near future, illustrated by their concrete and automatic realization in the "Al Amal" project.

Here is a first example of the use of ligatures, in the (imaginary) Persian word of the previous illustration:

The reader can see the va-dzim ligature as well as the triple ligature gaf-pa-lam. In the next illustration the process of ligaturing is shown, as well as the special form of the ra letter, mentionned in the previous paragraph. Once again we have a triple ligature, ta-ba-tshim:



By counting all possible combinations of characters forming ligatures, one obtains more than a thousand of them. Fortunately there is no need to create four supplementary font tables to cover these; they can be formed by "constructions". A construction is a set of superposed characters, which all (with one exception) have zero width. These "characters" (in the computer sense: a "character" is an element of the font table) can be parts of a letter, simple connecting strokes or just points. This concept allows us to economize place in the font table, as well as in the size of PK files.

The reader may argue that TEX cannot (at least in its actual state) perform ligatures involving more than one font table. At this point it might be interesting to note that certain ligatures could not be performed by TEX anyway (even if all parts were

in the same font table). Let's take for example the case of the ba-ta-dzim-mim ligature. The problem is that there is no ba-ta ligature. We can of course use the common trick[3] of setting some digit (let's say 7) to be the result of an hypothetical ba-ta ligature, and then define a 7-dzim ligature as ba-ta-dzim (which fortunately does exist). But suppose the letter following 7 is not dzim, but for example kaf. Since there is no ba-ta-kaf ligature, we must forget all about the 7 and go back to the pair of letters ba,ta; which is unfortunately impossible[4]

The purpose of the last paragraph was to show why one is better off when letting the preprocessor perform all complicated ligatures, which is exactly what is proposed in the "Al Amal" project. Nevertheless all ligatures which *can* be handled automatically by TEX (which are already quite a few) are included in the main font table, and thus are accessible in the first level of Arabic input (without pre-processor). The remaining ligatures (which are accessed by macros, written by a quick pre-processor or a patient user) form a second font table, together with vowels, and some other symbols frequently used in the Quran, some of which will be described in the next paragraph.

First of all, discretionary signs, indicating where the reciter is allowed to pause; mandatory signs, where the reciter must stop or must continue; marginal marks which show the division of the Quran into surahs, ruku's, ayahs. All of these are small characters or group of characters placed over the regular line of text; the isolated ayin with three numbers indicates the serial number of the ruku in the surah, the number of ayahs in the ruku, and the number of the ruku in the juz: TEX can do this very easily using a macro `\margayn{}{}{}` where the three arguments correspond to the three numbers.

The most hidden special symbol is perhaps the "redundant stroke" (like a ba without the point) which is neither pronounced nor has any grammatical value or associated vowel. It simply is there; the reader can encounter it at locations 2.286, 3.151, 4.114, 6.162 and many more. This character is also provided in the font,

[3] Used for example in the AMS encoding of Cyrillic fonts.

[4] The author deep in his hard awaits the day when TEX will be compatible with UNICODE. A 65,000 character font should be sufficiently large to contain all possible ligatures and parts-of-ligatures, including vowels and diacritics...

in initial and medial form.

Finally a very common sign in comments is seen in the left of the following illustration, while on the right the title of the Holy Quran Book is displayed calligraphically. The latter character of the font can be seen also at the end of this article, in inverted white-on-black form.

The "Bismillahi-Rahmani aRahim" at the beginning of this paper is also a character of the font. It is the (scaled) copy of a 6 meter high statue in Djeddah, Saudi-Arabia[5].

These three calligraphic complexes have been first drawn in PostScript on a graphic program following a scanned sample; then the PostScript code has been converted into rudimentary Metafont code (filled outlines explicitly defined by their control points). This method could only be applied because there are no metaness requirements. A bolder version can eventually be obtained by changing all `fill` instructions into `filldraw`, and by using an appropriate `pencircle` of non-zero width.

2.6 Positionning of vowels and other diacritics

In the two last sections we have seen examples of problems which are solved by TEX alone, while the user just has to indicate the control sequence of the symbol he wishes to typeset. The author has kept the positionning of vowels as last, because it is an example of what TEX cannot do accurately from the very beginning, but must first "be trained" (the reader will soon see what is meant by that).

[5]How beautiful it is for a culture to make monuments out of words!...

Classical Arabic texts are vocalized, in the sense that to each letter are assigned a vowel and eventually a diacritical mark. Letters do not have the same width, not the same height and often the best place for a vowel is not on the middle axis of the letter. The situation gets even more complicated when ligatures are used. In this case letters are vertically ordered and vowels must be positionned in a way that there is no confusion on their correspondence to letters.

To solve this problem the following three-step solution is adopted:

1. there is a default position for vowels/accents: at a default height or depth, and on the middle axis of the letter. In this way, by writing `\ba0{\damma}` one will obtain a ba with a damma vowel at default height (one of the font parameters) and centered in the middle of the character box;
2. the default position can be adjusted for the particular character inside the accenting mechanism. The adjustment units are `ex` and `em`, the font x-height and em-width, so that this adjustment is independent of possible (horizontal or vertical) scaling of the font. In this way, on every occurence of an isolated ba the upper vowel will be adjusted;
3. the default position can also be (re-)adjusted on the fly, by introducing an adjustment macro inside the text: `\ba0{\damma\adjust{-0.1}{0}}` will be adjusted 0.1 em to the left. This can be useful for rare cases of letter/accent combinations, or for experimental reasons, before inserting this adjustment inside the accenting mechanism macro. The user can periodically record these changes and progressively improve the aspect of his vocalized Naskhi text.

The reader now understands the idea of "training TEX" to do correct accent positionning. The process is the same as for the traditional printer: at first he would choose a default position, then he would make temporary adjustments until he would be sure that they look good, and finally he would record these and repeat them automatically converging slowly to a mature style. Of course the computer has the big advantage that every such improvement is immediately at the disposal of every user and this transcedents in some sense the professionnal experience of the

traditional kraftmanship.

3 Conclusion

TEX can do more than just typeset good math. It is an open system which can be adapted to any need and face any challenge. One of these challenges was traditional Naskhi; the author really enjoyed bringing this project near to accomplishment. An even more tempting challenge would be to design a Nastaliq system, where every letter of a word is written on up to 10 different levels. *À suivre...*