ELECTRONIC PUBLISHING, VOL. ?(?), 1 (??? 19??)

Typesetting Khmer

YANNIS HARALAMBOUS

Institute of Oriental Languages and Civilizations of Paris
Private address: 187, rue Nationale, 59800 Lille, France
Fax (33) 20.40.28.64, Internet `Yannis.Haralambous@univ-lille1.fr`

SUMMARY
Because of the complexity of Khmer script, up to now there has been neither typesetting system nor standard encoding for the Khmer language. In this paper are presented: (a) a complete typesetting system for Khmer based on TEX, METAFONT and an ANSI C preprocessor, as well as (b) a proposal of 8-bit encoding table for Khmer information interchange. Problems of phonic input, subscript and superscript positioning, collating order, spelling reforms and hyphenation are solved, and their solutions described. Finally an alternative solution using 16-bit output font tables is briefly sketched.

KEY WORDS Khmer TEX METAFONT Computer Typesetting

1 INTRODUCTION

Certain languages use characters or character combinations which change according to the context. A common example in English (but not in Portuguese and Turkish!) are the 'fi' and 'fl' ligatures. Everytime he/she encounters the combination of letters 'f', 'i', the typesetter has to replace it by the ligature 'fi'. This practice, while remaining exceptional for the Latin script, becomes very important for certain Oriental scripts like Arabic (see [1], [2]), Indic scripts, Korean or Khmer.

Because of the repetition and transformation of the various shapes involved in this process, the best way of creating a font with strong contextual properties is to use a programming language, like METAFONT. Part of the contextual analysis can be done by TEX (in simple cases, such as Latin or modern Arabic), otherwise one has to use an independent preprocessor.

In this paper we present a typesetting system for one of the most complicated scripts: Khmer. In this case, the TEX/METAFONT/preprocessor approach is essential. Since there has been no standardization for Khmer information interchange yet, we also present a proposal for a Khmer 128-character table. This table has been submitted to ISO 10646 WG-2 for acceptance. Finally, the solutions to other typesetting problems such as hyphenation are also presented.

2 THE KHMER SCRIPT

The Khmer script is used to write Khmer, which is the official language of the Cambodian Republic and belongs to the Mon-Khmer group of Austroasiatic languages. It is a very old

0894–3982/??/??0001–1$?.??

and beautiful script, and from the typesetters point of view, one of the most challenging and exciting scripts in the world.

To understand the complexity of Khmer typesetting, we will start with a quick overview of the Khmer writing system. Khmer is written from left to right; the Khmer alphabet has 32 *consonants*, the following:

ក ខ គ ឃ ង ច ឆ ជ ឈ ញ ដ ឋ ឌ ឍ ណ ត ថ ទ ធ ន ប ផ ព ភ ម យ រ ល
វ ស ហ ឡ

The character អ denotes the absence of consonant. From the typesetters point of view and with respect to collating order, it might as well be considered as a consonant. We will use a box □ to denote an arbitrary consonant.

These 33 “consonants” (except ឡ) can appear as *subscript consonants:*

□្ក □្ខ □្គ □្ឃ □្ង □្ច □្ឆ □្ជ □្ឈ □្ញ □្ដ □្ឋ □្ឌ □្ឍ □្ណ □្ត □្ថ □្ទ
□្ធ □្ន □្ប □្ផ □្ព □្ភ □្ម □្យ □្រ □្ល □្វ □្ស □្ហ □្អ

A subscript consonant is pronounced after the “primary” consonant. Nevertheless, as the reader has certainly noticed, the subscript consonant □្រ is written on the left of the primary consonant.

It is also possible to have *two* subscript consonants carried by the same primary consonant. In that case, the second subscript consonant has to be □្រ. Examples: ផ្ក្រិ, ស្គ្រ.

A consonant, consonant + subscript or consonant + double subscript combination can carry a *vowel*. There are 28 vowels:

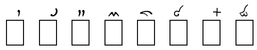
□ា □ិ □ិះ □ី □ឹ □ឹះ □ឺ □ុ □ុះ □ូ □ួ □ើ □ើះ □ឿ □ៀ
□េ □េះ □ែ □ែះ □ៃ □ោ □ោះ □ៅ □ុំ □ំ □ាំ □ះ □ៈ

Although vowels are always pronounced *after* consonants, their graphical representation literally surrounds the consonant/subscript combination: they can appear above, beneath, on the right or on the left of consonants. Often a vowel’s glyph has two or three non-connected parts.

When combining vowels with subscript consonants, the following graphical rules are followed:

- if the subscript has a right protruding stem then the vowel □ា connects to the subscript and not to the consonant: □្ឃ + □ា = □្ឃា etc.
- if the consonant carries both a subscript □្រ and a vowel with left branch, then the latter is placed on the left of the former: □្រ + □េ = □្រេ etc.
- if the consonant carries both a subscript consonant and a subscript vowel, then the latter is placed underneath the former: □្គ + □ូ = □្គូ, □្រ + □ូ = □្រូ etc.

Finally, a group of characters as described above can carry a *diacritical mark*. These are always placed above the character:



We will call the combination of consonant and possible subscript consonant, second subscript consonant, vowel and diacritical mark, a *consonantal cluster*. Theoretically there can be 535,060 different consonantal clusters, but in practice less than 1% of them are really used. An analytic decomposition of A. Daniel's Khmer-French dictionary [3] has provided no more than 2,821 different consonantal clusters out of 25,000 entries; colloquial Khmer may require even less clusters.

Besides consonantal clusters there are also 14 "stand-alone" characters in the Khmer alphabet:

These carry neither subscript consonants, nor vowels, nor accents. They cannot be found in subscript form. Orthographical reforms of Khmer have in some cases replaced them by "regular" consonantal clusters.

Inside a sentence, Khmer words are *not* separated by blank space. A blank space denotes the end of a sentence (or of part of a sentence: it acts like the period or the semicolon in Latin script).

Hyphenation occurs between *syllables:* a syllable consists of one or two consonantal clusters with the sole restriction that the second cannot have a vowel. When a word is hyphenated, a hyphen is used. Sentences are "hyphenated" into words, but in that case, no hyphen is used. So from the typesetters point of view, between two clusters hyphenation can be

1. forbidden (when the two clusters belong to the same syllable);
2. allowed and produce a hyphen (when the two clusters belong to the same word);
3. allowed without producing a hyphen (when the two clusters belong to different words in the same sentence).

This quick overview of the Khmer script has shown some of its particularities (see also [4], [5], [6]). To conclude, the author would like to underline the fact that the main difficulty in Khmer typesetting is the divergence between phonic and graphical representation of consonantal clusters (see fig. 1).

This paper is divided into five sections:

1. the definition and discussion of an 8-bit encoding table for information interchange and storage in the Khmer script. Consonantal clusters are encoded according to their phonic representation;
2. the presentation of three Khmer font families, designed in the METAFONT language. These fonts correspond to the three main styles of Khmer type and provide sufficient metaness[1] to perform optical scaling, continuous interpolation from light to extra-bold weight and strong raster optimization;
3. the description of the process according to which the graphical representation of consonantal clusters is derived from the phonic one (this process being implemented in an ANSI C preprocessor);

[1] In METAFONT lingo, *metaness* is the possibility of parametrized variation of the characters shape, weight and style.

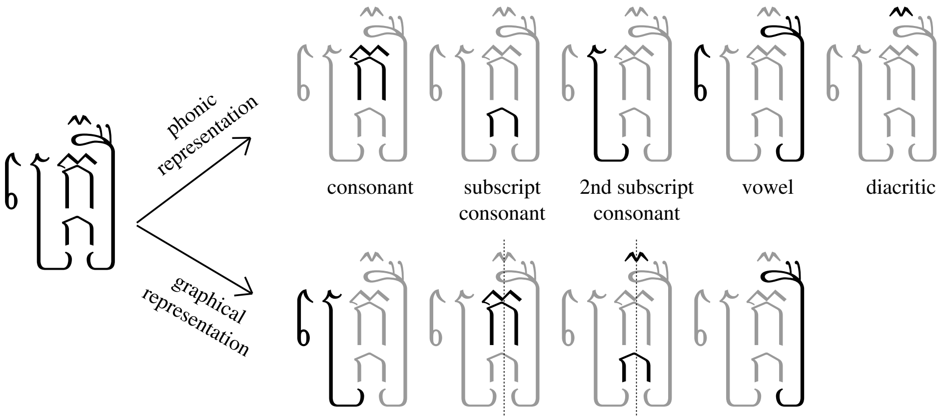


Figure 1. Decomposition of a Khmer consonantal cluster

4. an overview of hyphenation and spelling reform rules and their realization in the preprocessor;
5. shortcomings of the Khmer typesetting system and plans for future developments.

The author would like to thank Prof. Alain Daniel (Institute of Oriental Languages and Civilizations, Paris) for his continuous support and encouragement and the Imprimerie Louis-Jean (Gap) in the person of Maurice Laugier, for having financed this project.

3 AN 8-BIT ENCODING TABLE FOR THE KHMER SCRIPT

3.1 Discussion

As mentionned in the introduction, Khmer language is written using consonantal clusters and stand-alone special characters. The collating order of consonantal clusters is given lexicographically according to the cluster components:[2]

> Let $C_1 = c_1 s_1 s'_1 v_1 d_1$ and $C_2 = c_2 s_2 s'_2 v_2 d_2$ be two consonantal clusters, where $c_1, c_2 \in \{\text{consonants}\}$, $s_1, s_2 \in \emptyset \cup \{\text{subscript consonants}\}$, $s'_1, s'_2 = \emptyset$ or ្រ, $v_1, v_2 \in \emptyset \cup \{\text{vowels}\}$ and $d_1, d_2 \in \emptyset \cup \{\text{diacritics}\}$. Then
>
> 1. $c_1 \succ c_2 \Rightarrow C_1 \succ C_2$;
> 2. if $c_1 = c_2$ then $s_1 \succ s_2 \Rightarrow C_1 \succ C_2$ (where $\emptyset$ precedes any other element);
> 3. if $c_1 = c_2$ and $s_1 = s_2$ then $s'_1 \succ s'_2 \Rightarrow C_1 \succ C_2$;
> 4. if $c_1 = c_2, s_1 = s_2$ and $s'_1 = s'_2$ then $v_1 \succ v_2 \Rightarrow C_1 \succ C_2$;
> 5. if $c_1 = c_2, s_1 = s_2, s'_1 = s'_2$ and $v_1 = v_2$ then $d_1 \succ d_2 \Rightarrow C_1 \succ C_2$.

The table of 128 codes for Khmer characters presented below respects the collating order. Besides consonantal clusters and special characters, the following signs have been included in the 8-bit encoding:

1. digits: ០, ១, ២, ៣, ៤, ៥, ៦, ៧, ៨, ៩;

[2] The symbol $\emptyset$ denotes an empty set.

2. punctuation marks other than the ones borrowed from Latin script: ៗ (leikto) a variant form of the digit ២, indicating that the previous word is repeated (similar to Latin *bis*), ។ (khan) and ៕ (bariyatosan), equivalent to a full stop, ៖ (camnocpikuh) a graphical variant of the Latin colon, and the French *guillemets* « , »;
3. the currency symbol ៛ (rial);
4. the invisible code `WBK` (word-break) to indicate the word limits inside a sentence.

Have *not* been included in the table:

- the archaic characters ឝ and ឞ which were abolished about a century ago;
- the punctuation marks ៚ (cow's urine) and ៙ (cock's eye), used in poetry, divination and classical texts;
- the variant forms □ិ, □ី of □ិ, □ឹ, used in [7].

These characters are nevertheless included in the TEX output fonts and can be accessed via macros.

3.2 The Table

On fig. 2 the reader can see the table of codes 128–255 of the proposed 8-bit encoding for Khmer information interchange and storage. The 7-bit part of the table conforms to ISO 646 (standard 7-bit ASCII). Positions `0xCF` and `0xDF` are empty.

Codes `0x80`–`0x9F` and `0xC0` represent consonants; the corresponding subscript consonants are offset by 32 positions: they are represented by codes `0xA0` – `0xBE` and `0xE0`. The consonant `0x9F` does not have a corresponding subscript consonant. The practice of having subscripts 32 positions apart from primary consonants is similar to the 32-position offset of uppercase and lowercase letters in ISO 646 (7-bit ASCII).

Codes `0xC0`–`0xCE` represent special characters. Digits have been placed in positions `0xD0`–`0xD9`, vowels in `0xE1`–`0xF5` and diacritics in `0xF8`–`0xFF`. Finally, `0xFA` is the currency symbol, `0xDB`–`0xDE` are punctuation marks and `0xBF` is the word-break code `WBK`.

Because of the 128-character limitation, the following characters have not been included in the table: □ិះ, □ឹះ, □ុះ, េ□ឹះ, េ□ះ, ែ□ះ, េ□ាះ

They have to be represented by the following code pairs:

□ិះ	=	`0xE2 0xF4`	□ឹះ	=	`0xE4 0xF4`	□ុះ	=	`0xE6 0xF4`
េ□ឹះ	=	`0xE9 0xF4`	េ□ះ	=	`0xEC 0xF4`	ែ□ះ	=	`0xED 0xF4`
			េ□ាះ	=	`0xEF 0xF4`			

3.3 Requirements for Khmer script software

As in the case of Arabic and Hindi, software displaying Khmer text has to provide context-analytic algorithms. Below is an exhaustive list of the necessary context-dependent transformations:

1. when code `0xBA` follows a code in the range `0x80`–`0x9E`, `0xC0` then their glyphs must be permuted, for ex. ក + ្រ□ → ក្រ.

Hexa	80	90	A0	B0	C0	D0	E0	F0
0	ក	ថ	◌្ក	◌្ថ	អ	០	◌្អ	◌ៅ
1	ខ	ទ	◌្ខ	◌្ទ	ឥ	១	◌ា	◌ុំ
2	គ	ធ	◌្គ	◌្ធ	ឦ	២	◌ិ	◌ំ
3	ឃ	ន	◌្ឃ	◌្ន	ឧ	៣	◌ី	◌ាំ
4	ង	ប	◌្ង	◌្ប	ឪ	៤	◌ឹ	◌ះ
5	ច	ផ	◌្ច	◌្ផ	ឩ	៥	◌ឺ	◌ៈ
6	ឆ	ព	◌្ឆ	◌្ព	ឫ	៦	◌ុ	«
7	ជ	ភ	◌្ជ	◌្ភ	ឬ	៧	◌ូ	»
8	ឈ	ម	◌្ឈ	◌្ម	ឭ	៨	◌ួ	◌់
9	ញ	យ	◌្ញ	◌្យ	ឮ	៩	◌ើ	◌៊
A	ដ	រ	◌្ដ	◌្រ	ឯ	៛	◌ឿ	◌៉
B	ឋ	ល	◌្ឋ	◌្ល	ឰ	ៗ	◌ៀ	◌៎
C	ឌ	វ	◌្ឌ	◌្វ	ឱ	៖	◌េ	◌៏
D	ឍ	ស	◌្ឍ	◌្ស	ឲ	។	◌ែ	◌៌
E	ណ	ហ	◌្ណ	◌្ហ	ឳ	៕	◌ៃ	◌៍
F	ត	ឡ	◌្ត	WBK			◌ោ	◌័

Figure 2. Positions `0F80–0FFF` *of ISO 10646 (proposal).*

2. when code `0xBA` follows a pair of characters $\alpha\beta$, with $\alpha \in \{$`0x80`–`0x9E`, `0xC0`$\}$, $\beta \in \{$`0xA0`–`0xBE`, `0xE0`$\}$ then the glyph of `0xBA` must appear on the left of the glyphs of α, β, for ex. ក + ្យ + ្រ → ក្រ្យ.
3. when codes `0xE9`–`0xEC` and `0xEF`–`0xF0` follow a combination of character codes $\alpha, \alpha\beta, \alpha\beta\gamma$ where α and β are as in the previous item and $\gamma =$`0xBA`, then the glyph េ must appear on the left of the latter combinations. Example: ក + ្យ + ្រ + ោ → ក្រ្យោ.
4. when codes `0xED` and `0xEE` follow a combination $\alpha, \alpha\beta, \alpha\beta\gamma$ of codes as in the previous item, then their glyphs must appear on the left of these combinations;
5. when code `0x89` (ញ) is followed by a code in the range `0xA0`–`0xBE`, `0xE0` then the variant glyph ញ must be used. Example: ញ + ្ជ → ញ្ជ.
When the second code is `0xA9` then a variant glyph must be used for it as well: ញ + ្ញ → ញ្ញ.

These contextual transformations have been implemented by the author into a modified version of the Macintosh freeware text editor Tex-Edit by Tim Bender, included in the package. In fig. 3 the reader can see the effect of striking successively keys <ក>, <្យ> (<subscript modifier> followed by <យ>), <្រ> (<subscript modifier> followed by <រ>), <ោ>, to finally obtain the consonantal cluster ក្រ្យោ.

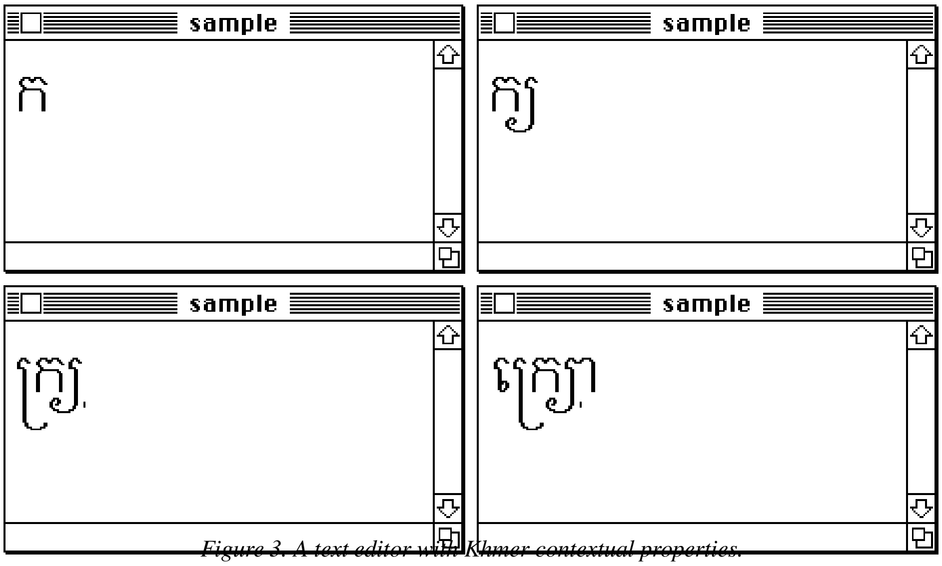


Figure 3. A text editor with Khmer contextual properties.

4 THE DESIGN OF KHMER FONTS IN METAFONT

4.1 Font styles

There are three styles used in Khmer typesetting: standing (aksar ch-hor), oblique (aksar chrieng) and round (ak-sar mul). The latter is virtually identical to inscriptions of the 12th and 13th centuries at Angkor Wat and is reserved for religious texts, chapter headings, newspaper headlines, inscriptions and on other occasions where there is a desire to create a contrast with the oblique script, to add a touch of formality, or to provide variation of emphasis (see [5]).

The author has designed three METAFONT font families, corresponding to these styles; samples of these fonts in 14.4 point size follow hereafter; figure 4 shows a sample headline using the round font.

Standing characters

នៅឆ្នាំ ១៩៦២ ប្រទេសកម្ពុជាបានចេញមក នៅសល់តែស្បែកនឹង-
ឆ្អឹង អំពីការប្រយុទ្ធដ៏យូរអង្វែង នឹងដ៏ឃោរ ឃៅមួយ ប្រឆាំងនឹង-
ការជិះជាន់របស់យួន នឹងសៀម ។ ការឈឺចាប់នៃប្រជាពលរដ្ឋ-
យើងនៅពេលនេះ គឺហួសពីការគិតទៅហើយ ប៉ុន្តែពួកបស្ចិមប្រ-
ទេស គេនៅតែពុំដឹងឮ ចំពោះការឈឺចាប់នេះទេ ។ ធំណែកខ្មែរវិញ
យើងមិន មានភ្លេចសោះឡើយ ។

Oblique characters

*នៅឆ្នាំ ១៩៦២ ប្រទេសកម្ពុជាបានចេញមក នៅសល់តែ-
ស្បែកនឹងឆ្អឹង អំពីការប្រយុទ្ធដ៏យូរអង្វែង នឹងដ៏ឃោរ ឃៅ-
មួយ ប្រឆាំងនឹងការជិះជាន់របស់យួន នឹងសៀម ។ ការឈឺ-
ចាប់នៃប្រជាពលរដ្ឋយើងនៅពេលនេះ គឺហួសពីការគិតទៅ-
ហើយ ប៉ុន្តែពួកបស្ចិមប្រទេស គេនៅតែពុំដឹងឮ ចំពោះការ-
ឈឺចាប់នេះទេ ។ ធំណែកខ្មែរវិញ យើងមិន មានភ្លេចសោះ-
ឡើយ ។*

Round characters

**នៅឆ្នាំ ១៩៦២ ប្រទេសកម្ពុជាបានចេញមក នៅសល់តែស្បែក-
នឹងឆ្អឹង អំពីការប្រយុទ្ធដ៏យូរអង្វែង នឹងដ៏ឃោរ ឃៅមួយ ប្រ-
ឆាំងនឹងការជិះជាន់របស់យួន នឹងសៀម ។ ការឈឺចាប់នៃប្រ-
ជាពលរដ្ឋយើងនៅពេលនេះ គឺហួសពីការគិតទៅហើយ ប៉ុន្តែ-**

ពួកបស្ចិមប្រទេស គេនៅតែពុំដឹងឮ ចំពោះការឈឺចាប់នេះទេ
។ ចំណែកខ្មែរវិញ យើងមិន មានភ្លេចសោះឡើយ ។

Figure 4. Headline in round style.

In contrast to systems like PostScript, in which fonts are interpreted during the printing process and where similar complexity can slow the process down, in the case of METAFONT (see [8]) fonts are compiled separately and stored on disk in a highly compacted form. On powerful platforms, fonts can be created by METAFONT just before the printing process, stored on hard disk, and removed afterwards. On slower platforms (for example personal computers), fonts are stored permanently on the hard disk. A METAFONT font package takes much more space than a set of PostScript fonts; this disadvantage is counterbalanced by the fact that fonts created by METAFONT are under the complete control of the user: characters are already rasterized in an optimal and homogeneous way. Other advantages of using METAFONT, in particular for the design of Khmer fonts, are the following:

1. *Modularity*. Characters are designed in a modular way: descriptions of parts which are repeatedly used are stored as subroutines with an arbitrary number of parameters for adapting them to different situations where they can occur. A modular design makes the font more homogeneous and easier to modify: a change in a subroutine will affect the whole font.
2. *Metaness*. In Khmer, standing and oblique letters share the same design, except that certain curves of the latter are rounder than the corresponding curves of the former (for example, compare ឈ with *Ch*, or ខ្រ្ឈ with */rKh/Ch*). To preserve the similarity between the two styles, standing and oblique fonts are generated using the same METAFONT code; only the values of slant, roundness and interletter spacing parameters are different.
3. *Raster optimization*. Vertical strokes of Khmer letters must always be of the same width, regardless of the resolution or of the output device. In the METAFONT "drawing space", coordinates are given in pixels; this is possible because of the fact that output device characteristics are given at the beginning of the METAFONT run. The condition "two vertical strokes should have the same width" is given by a simple linear equation "*width of left stroke* = *width of right stroke*" with the sole precaution that the left edges of strokes fall on the pixel raster (this is obtained by the METAFONT primitive `round`).

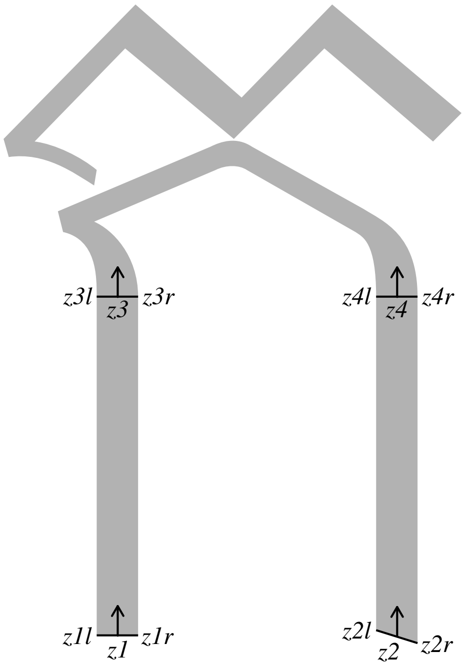


Figure 5. Raster optimization with METAFONT.

To illustrate this we will take the example of the two vertical strokes of letter ᕼ. As in fig. 5 let's call z_1 and z_2 the points which are on the baseline and on the central paths of the two vertical strokes. Also let z_3, z_4 be the upper extremities of central paths of the two vertical strokes. Let the straight segments $[z_{1l}, z_{3l}]$, $[z_{2l}, z_{4l}]$ be the left edges of the vertical strokes, and $[z_{1r}, z_{3r}]$, $[z_{2r}, z_{4r}]$, their right edges. The fact that z_1, z_2 are on the baseline can be expressed as $y_1 = y_2 = 0$, where y_* is the y-coordinate of z_*. In the same way, the fact that the strokes are vertical can be expressed by the couple of equalities $x_1 = x_3, x_2 = x_4$ where x_* is the x-coordinate of z_*. Their precise location is given as a multiple of a global variable w, corresponding to the width of a generic character: $x_{1l} = 1/14w, x_{2l} = 0.86w$. As mentioned in the previous paragraph, the two strokes must have the same width (called *stem*). Since they are vertical, we can determine their width by using only x-coordinates. The width equality can be expressed as $x_{2r} - x_{2l} = x_{1r} - x_{1l} =$ *stem*.

So far, so good. But let us consider an example in which things can go wrong. METAFONT does its calculations in pixels or fractional parts of pixels which are rounded afterwards to the closest integer value. Let us suppose that the two strokes are *stem* $= 2.2$ pixels wide. Of course this should always be rounded to 2 pixels. Now suppose $x_{1l} = 1/14w = 1.2$ and $x_{2l} = 0.86w = 14.4$. Values will be rounded in the following way: $x_{1l} = 1.2 \to 1, x_{1r} = 1.2 + 2.2 = 3.4 \to 3$, and hence $x_{1r} - x_{1l} = 2$, while $x_{2l} = 14.4 \to 14, x_{2r} = 14.4 + 2.2 = 16.6 \to 17 \Rightarrow x_{2r} - x_{2l} = 3$ and so the right stroke is one pixel wider than the left one.

To prevent this, one simply instructs METAFONT to round values *while calculating point locations*. By writing $x_{1l} = \text{round}(1/14w)$ and $x_{2l} = \text{round}(0.86w)$, both

x_{1l} and x_{2l} will have integer values (in our example, 1 and 14); this implies that the fractional parts of $x_{1r} - x_{1l}$ are the same $x_{2r} - x_{2l}$ and hence both will get rounded in the same way (either to the right or to the left), and their values will remain equal after rounding.

The METAFONT code[3] that follows implements these operations; it is meant to illustrate the ease of raster optimization (*hinting*, in PostScript lingo) in this programming language. Lines starting with % are comments.

```
x1r-x1l=x2r-x2l=x3r-x3l=x4r-x4l=stem;
% strokes are of same width "stem"
x1r=x3r; x2r=x4r;
% and they are vertical
x1l=round(1/14w); x2l=round(0.86w);
% their left edges take integer pixel values
fill z1r--z3r--z3l--z1l--cycle;
fill z2r--z4r--z4l--z2l--cycle;
% fill the strokes with black
```

The reader may have noticed that the "pen position" z_2 is actually oblique, while z_1 is horizontal. This fact has not influenced rasterization, since all roundings done in this example are on the x-coordinate level. For the z_2 pen position, a different kind of optimization can be performed: in low resolutions the straight segment $[z_{2l}, z_{2r}]$ may look "broken". This will mean that the angle being too small with respect to the pixel size, the segment will be displayed as a certain number of concatenated horizontal rows of pixels. The number of these rows is the rounded value of $y_{2l} - y_{2r}$. We can decide to replace the oblique segment $[z_{2l}, z_{2r}]$ by a horizontal one, if this number is smaller than a certain value, for example 2. This will be written as:

```
if (round(y2l-y2r) <= 2): y2l:=y2; y2r:=y2; fi
```

where the assignment operator := will change the values of y_{2l}, y_{2r}.

Special care has been taken for raster optimization, since output devices in Cambodia are mostly of very low resolution.

4. *parametrization and optical scaling*. When type is scaled, widths of strokes are not necessarily scaled by the same factors. Large point sizes must be narrower and thinner proportionally to standard point size; small point sizes must be larger and with increased interletter space, to enhance readability. This problem is very well known for the Latin script and is solved in METAFONT created font families like Computer Modern. Similar solutions have been adopted for Khmer.

   There is a second advantage of optical scaling and parametrization of character shapes. In Cyrillic and Greek scripts one can define font families similar to Latin ones: there already exist Cyrillic and Greek Times, Helvetica, Courier, Garamond, Baskerville etc. The choice of a Khmer/Latin font combination is more delicate. Parametrization of character widths gives the user the possibility to change the *gray density* factor of the Khmer font and adapt it to the Latin font he is using.

   In figure 6, the Khmer letter ក has been reproduced 256 times, with different values of two parameters: the widths of "fat" and "thin" strokes. The central vertical symmetry axis represents "Égyptienne"-like characters, where the parameters have the same

[3] This code is voluntarily kept simplistic; there are more elegant ways to program the same operations…

value. This classification can of course be refined and enables an arbitrarily precise choice of the font gray density.

5 TRANSLATING A PHONIC TO A GRAPHIC DESCRIPTION

In 3.3 we have given a quick overview of the minimal contextual analysis involved in displaying Khmer script on screen. The situation is much more complicated in the case of high quality typesetting.

TEX is the ideal tool for typesetting in Oriental scripts like Khmer, because of the inherent fundamental concept of *boxes* (see [9], [10], [11]). Like in mathematical formulas, elements of a consonantal cluster are moved to aesthetically correct positions and then grouped into a single and indivisible "box" which TEX treats as a single entity.

In this section we will see how the graphical representation of a cluster is constructed, using both the preprocessor and TEX.

5.1 Graphical classification of Khmer cluster components

As already mentioned, there is a strong divergence between the phonic and graphical representation of a consonantal cluster: for example, is $c =$ឈ៊, $s_1 =$◌្ឈ, $s_2 =$្រ◌, $v =$េ◌ា, then for the same cluster ឈ៊្ឈ្រោ, the former representation is <c><s_1><s_2><v> and the latter <v (left branch)><s_2><c><s_1><v (right branch)>.

A thorough study of Khmer script and traditional typography, has resulted in the following classification of graphical components of a consonantal cluster:

1. **the "left part".** Four elements which are placed on the left of a consonant: េ◌, ើ◌, ែ◌, ្រ◌.
2. **the "central part".** All consonants: ក, ខ ... អ. Also consonant + vowel ∈ {◌ា, ◌ាំ, ◌ៅ} combinations, whenever the vowel is attached to the consonant and not to a subscript: កា, កាំ, កៅ etc. but *not* ក្បា.
   The difference between "left" and "central" part is that only the latter is taken into account when determining the symmetry axis of the cluster.
3. **the "movable" part.** Subscripts and superscripts which are moved horizontally so that their symmetry axis coincides with the axis of the central part: ◌្ក ... ◌្អ, and ◌ិ, ◌ី, ◌ឹ, ◌ឺ, ◌ុ, ◌ូ, ◌ំ, ◌៉, ◌៊, ◌៌, ◌៍, ◌៎.
4. **the "right" part.** Elements placed on the right of the central part, and not involved in the determination of the cluster symmetry axis. In this category we have certain subscript characters: ◌្ឃ ...◌្ឡ, as well as selected subscript and superscript vowels and diacritical marks: ◌្វ, ◌ឿ, ◌ៀ, ◌ះ, ◌ៈ, ◌់, ◌៏, ◌៓.

The effective graphical construction of a consonantal cluster by TEX, is done in the following way: the preprocessor's output replaces the phonic representation of a cluster (in the encoding described in 3.1) by a TEX macro `\KHccl` with 5 arguments: the first is a 9-digit number representing the phonic representation of the cluster (and with the property that if N, N' are numbers representing clusters C, C' then $C \succ C' \iff N > N'$, where $\succ$ is

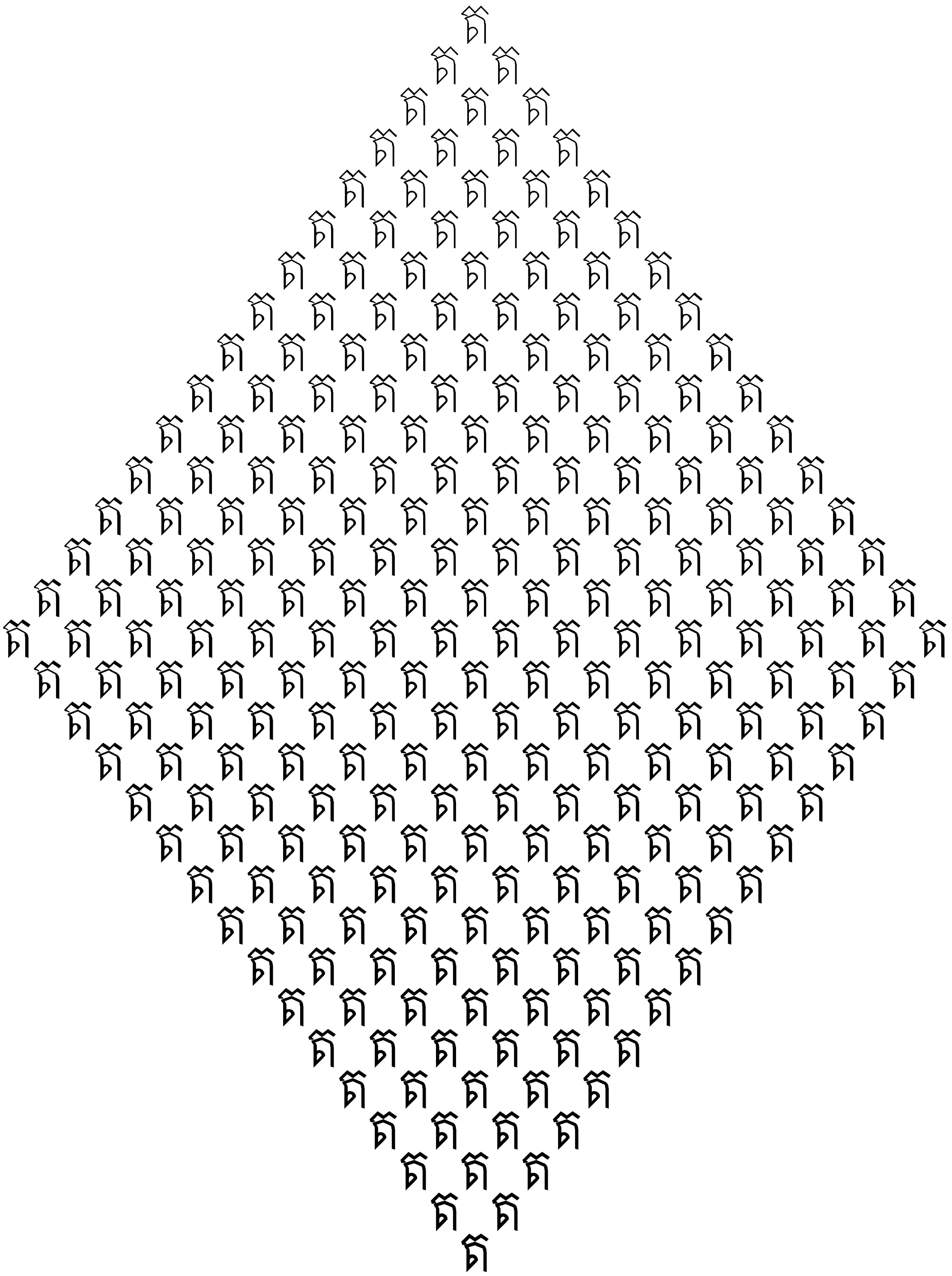


Figure 6. Test table for gray density fine-tuning of Khmer font

the collating order of clusters and > the usual ordering of integers); the remaining four arguments correspond to the four parts of the graphical decomposition of a cluster as described above. For example,

```
\KHccl{050311501}{e/r}{gA}{/k}{'}
```

indicates a left part `e/r` (េ្រ), a central part `gA` (ញ), a movable part `/K` (្ក) and a right part `'` (់). This example illustrates the important fact that the symmetry axis of the central part is not necessarily the middle axis of the box containing the central part:

ញ្ក and not ញ្ក

The difference is more than just of aesthetic nature: in some cases the vertical alignment of elements within a cluster is necessary to determine the cluster itself. Take for example characters `0x89` (ញ) and `0x96` (ព). When the latter is followed by a vowel ា it becomes ពា, which is indistinguishable from the upper part of the former: it is the lower part ◌ that enables differentiation. But when both happen to carry the same subscript consonant, then this lower part vanishes. The difference will be found in the alignment of the subscript consonant: in the case of ញ one would have for example ញ្ក, while in the case of ពា it would be ព្កា.

From these considerations we conclude that the symmetry axis location is a vital piece of information for every character; it depends on the shape of the individual character and *cannot* be given by a general font-independent rule.

In TEX there are several global parameters for a given font, but only 4 for every individual character of the font: width, height, depth, italic correction. The author has used the parameter "italic correction" as a carrier of the information related to the symmetry axis location.

The construction mechanism is very simple: TEX typesets first the left part and the central part of the cluster; then it moves to the left, by an amount equal to the italic correction of the central part and typesets the movable part; finally it moves back to the right edge of the central part and typesets the right part of the cluster.

To simplify this mechanism, all movable elements are of zero width. The reader can see an example in fig. 7, where TEX boxes are displayed in gray and the symmetry axis of the central part is shown as a dotted line.

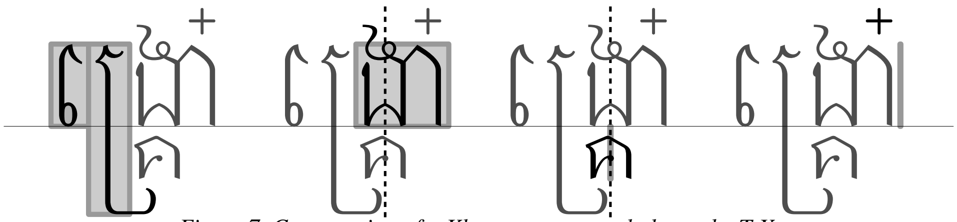


Figure 7. Construction of a Khmer consonantal cluster by TEX.

5.2 Special cases and exceptions

The mechanism of cluster construction described above fails in certain special cases. These are handled by using variant forms of graphical elements. A quick description of these cases follows.

1. often two or three subscripts or superscripts are found in the same cluster. In these cases the following rules apply:
   (a) in the case of two subscript consonants, the second being necessarily ◌្រ, a deeper form of the latter is used: ◌្រ + ◌្ក = ◌្ក្រ;
   (b) in the case of a subscript consonant and a subscript vowel, the vowel is placed under the subscript consonant: ◌្ក + ◌ួ = ◌្កួ. This rule also applies to the subscript consonant ◌្រ: ◌្រ + ◌ួ = ◌្រួ;
   (c) in the case of two subscript consonants and a subscript vowel, the consonants are placed as in (a) and the vowel is placed on the right of ◌្រ: ◌្រ + ◌្ក + ◌ួ = ◌្ក្រួ;
   (d) in some cases we have both a superscript vowel and a diacritical mark. The following combinations are known: ◌ិ៍, ◌ឹ៍, ◌ី៍, ◌ឺ៍, ◌ុំ, ◌៉៍;
2. to prevent confusion between the letter ប followed by vowel ◌ា, and the letter ឋ, the former combination of consonant and vowel is written បា. A variant of this letter is used in the presence of a subscript: បា + ◌្ក = ប្កា.
3. when a cluster with ◌្រ contains vowel ◌ឿ or ◌ៀ, then the width of the primary consonant determines the depth of the vowel: ត + ◌ៀ + ◌្រ = ត្រៀ, but ណ + ◌ៀ + ◌្រ = ណ្រៀ;
4. the letter ឡ is not supposed to carry a subscript consonant; in some rare cases, it carries subscript ◌្ហ: ឡ្ហ.

5.3 Operations depending on collating order

As mentioned in the previous section, the TEX command `\KHccl`, obtained by the preprocessor, describes a cluster by means of five arguments. The last four arguments describe the cluster graphically: they correspond to the four parts of the graphical decomposition of a cluster, according to 5.1. The first argument corresponds to the phonic decomposition of the cluster; it is a 9 digit number $N = c_1c_2s_1s_2s_3v_1v_2d_1d_2$ where

1. c_1c_2 determines the primary consonant of the cluster: c_1c_2 goes from $\texttt{01} =$ ក, to $\texttt{33}$ $=$ អ;
2. s_1s_2 determines the (first) subscript consonant: $s_1s_2 = \texttt{00}$ if there is no subscript consonant, otherwise s_1s_2 goes from $\texttt{01} =$ ◌្ក, to $\texttt{32} =$ ◌្អ;
3. $s_3 = \texttt{0}$ if there is no second subscript consonant, $\texttt{1}$ if there is a second subscript ◌្រ;
4. v_1v_2 determines the vowel: $v_1v_2 = \texttt{00}$ if there is no vowel, otherwise v_1v_2 goes from $\texttt{01} =$ ◌ា, to $\texttt{28} =$ ◌ះ;
5. d_1d_2 determines the diacritic mark: $d_1d_2 = \texttt{00}$ if there is no diacritic, otherwise d_1d_2 goes from $\texttt{01} =$ ◌់, to $\texttt{08} =$ ◌៍.

A complete list of characters, alphabetically ordered, is given in the introduction. Collating order rules mean that for clusters C, C' and their corresponding 9-digit numbers N, N', we have

$$C \succ C' \iff N > N'.$$

where $\succ$ is the collating order of clusters. The numbers N, N' can easily be ordered since the collating order of clusters corresponds to their order as integers. This fact enables straightforward searching, sorting, indexing and other operations involving collating order.

6 HYPHENATION AND OTHER PREPROCESSOR FEATURES

6.1 Hyphenation

Hyphenation in Khmer obeys a very simple rule: words are hyphenated between syllables. Unfortunately this rule can hardly be implemented on a computer since there is no algorithmic way of detecting syllables: a syllable can consist of one *or two* consonantal clusters.

With the help of Prof. Alain Daniel, an empirical hyphenation mechanism has been developed out of several general rules and observations. Below is a first set of rules — there will be further refinement after thorough testing on bigger amounts of Khmer text.

Let C, C' be consonantal clusters. Hyphenation C-C' is possible whenever:

1. C' contains a vowel;
2. C contains a vowel such as ◌ិះ, ◌ឹះ, ◌ុះ, ◌ើះ, ◌េះ, ◌េ៉ះ, ◌េ៊, ◌ោះ, ◌ៅ, ◌ុំ, ◌ំ, ◌ះ, or one of the diacritical marks ◌់, ◌៍;

Hyphenation is always possible before or after special characters.

TeX provides an internal hyphenation mechanism based on *hyphenation patterns*. Unfortunately this mechanism cannot be used in the case of Khmer consonantal clusters, since these are enclosed in boxes and hence cannot be considered as characters by TeX. For this reason, the hyphenation algorithm is performed by the preprocessor; whenever one of the two above rules is satisfied, the TeX macro `\-` is included in the output. This command expands as

```
\def\-{\discretionary{-}{}{}}
```

so that a hyphen is obtained whenever a word is hyphenated. There is no algorithm yet for automatic decomposition of sentences into words: the user is asked to include `WBK` (word-break) codes between words inside a sentence. These codes are converted into `\KHwbk` commands by the preprocessor; `\KHwbk` expands into

```
\def\KHwbk{\discretionary{}{}{}}
```

that is: a potential hyphenation point, *without* hyphen.

6.2 Decomposition of special characters and spelling reforms

The special characters (codes `0xC1`–`0xCE`) are mostly historical residues and loans from other languages (Pali and Sanskrit). There have been many attempts by the Cambodian Ministry of Education to restrict their number, and eventually replace some of them by regular consonantal clusters.

This replacement can vary from word to word. Prof. Alain Daniel has established a list of reformed words and their replacements. This list is known to the preprocessor, which will output every special character as a TEX macro with a numeric argument indicating the potential replacement by some other special character or by a consonantal cluster. For example, depending on the surrounding word, ឳ is output as `\KHao0`, `\KHao1`, `\KHao2`, `\KHao3` or `\KHao4`. If a certain boolean variable `\ifreformed` is false, then all five macros will always expand into ឳ. On the other hand, if the boolean is true, then the first macro will expand into ឳ, the second into អោ, the third into ឱ, the fourth into អ and the fifth into អៅ.

Below is a first list of reformed words, known to the preprocessor. The special characters and their decompositions are set in bolder type.

រំឥល → រំអិល	ឪកា → ឳកា	ឮលាន់ → លឺលាន់	សំឳក → សំអោក
រឥល → រអិល	ឪដ្ឋ → អូដ្ឋ	ឮសាយ → លឺសាយ	ឳវ៉ែ → ឱវ៉ែ
ឥសុការ → ឱសុការ	ឪឆ → អូឆ	ឰរាវ័ណ → អៃរាវ័ណ	ឳ! → អៅ
ឦស → ឥស	ឪម! → ឳម	ឰ! → អៃ!	ឳងឡោង → អោងឡោង
ឦសធរ → ឥសធរ	ឪរុ → ឳរុ	ឰក- → អៃក-	ឳដ្កា → អដ្កា
ឦសាន → ឥសាន	ឪរ្វ → ឳរ្វ	ឰក្យ- → អៃក្យ-	ឳបុសថ៍ → ឱបុសថ៍
ឦស្វរ → ឥស្វរ	ឫ → រឺ	ឰរាវ័ត → អៃរាវ័ត	ឳត → អោត
ឦស្វរៈ → ឥស្វរៈ	សំឫទ្ធី → សំរឹទ្ធី	ឰស្វរ → អៃស្វរ	ឳឡាន → ឱឡាន
ឳក → អុក	រំឭក → រំលឹក	ឰស្វរ្យ → អៃស្វរ្យ	ឳបប្បាតិក → ឱបប្បាតិក
ឳស → អុស	រឭក → រលឹក	ប្រឳង → ប្រអោង	ឳវ៉ើ → ឱវ៉ើ
ក្រឳ → ក្រអៅ	ឮ → លឺ	រឳក → រអោក	ឳសហ៍ → ឱសហ៍
ក្រាងក្រឳ→ក្រាងក្រអៅ	ឮជ័យ → លឺជ័យ	លំឳន → លំអោន	ឳ → អៅ!
	ឮជា → លឺជា		ឳទក → ឱទក

7 SHORTCOMINGS AND PLANS FOR FURTHER DEVELOPMENT

The system presented in this paper enables high quality Khmer typesetting. It is the first Khmer typesetting system which solves problems such as text input in phonic order, positioning of subscripts and superscripts, optical scaling, hyphenation and replacement of special characters.

Nevertheless the graphical cluster-construction algorithm described in this paper has certain flaws; a few examples:

- if a consonant with subscript consonant carries the ុ vowel, then the latter should be justified at the right edge of the *subscript*, which is not necessarily aligned with the right edge of the consonant. For example, in the (hypothetical) cluster រ្ឝ្ញុ, the ុ is badly positioned;
- take a narrow letter (like រ, វ) which carries a large subscript (like ្ឝ or ្ឋ) and suppose you are at the line boundary (either left or right); then contrarily to the normal use of subscripts, it is the subscript which should be used for line justification, and not the consonant.

These problems cannot be solved using the current mechanism (in which TEX considers that all subscripts and superscripts are of zero width). It could be possible to use subscripts with non-zero width, but (a) this would slow the process down, (b) it wouldn't solve the problem of the line boundary, since we are asking for contradicting properties: inside a sentence subscripts should not interfere in determining the distance between clusters, while at the line's boundary they should[4]. Furthermore, one could imagine a sentence ending with and the next sentence starting with . The blank space in between is hardly sufficient to prevent clusters from from overlapping. Visually, the beginning of the sentence is lost.

Corrections to these problems can be performed manually (because these problems occur very rarely). However, a much more natural and global solution would be to treat consonantal clusters as individual codes in a 16-bit encoding scheme. As mentioned in the introduction, only 2,821 clusters (out of 535,000 theoretical possibilities) have been detected in the fairly complete dictionary of Prof. Alain Daniel, so a 16-bit table would be more than sufficient to cover them.

This method of Khmer typesetting, is part of the Ω project, undertaken by John Plaice (Université Laval, Canada) and the author. The first realisation of Ω is an extension of TEX (and the two utilities VPtoVF and DVICopy) to 16-bit fonts (allowing the use of 65,536 characters and 4,294,967,296 ligatures or kerning pairs). These fonts will be exclusively virtual: since the DVI file format allows up to 32-bit fonts there is no need to extend its specifications; DVI-files with 16-bit Ω fonts will be "devirtualized" through DVIcopy: the 16-bit virtual fonts will be replaced by their 8-bit base fonts. In this way Ω DVI files will be converted to standard 8-bit DVI files; no special DVI drivers will be needed (not even virtual font compatible ones). In the case of Khmer, the (unique) base font will contain the glyph descriptions (in `PK` or PostScript format) and the virtual font will contain the definitions of consonantal clusters. Since consonantal clusters will be treated by TEX as individual characters, one will be able to define kerning pairs between them and solve the main problem of Khmer typesetting.

Text input could still be done using the 8-bit encoding of section 1; internal ligaturing will map the 8-bit description of consonantal clusters into their codes in the 16-bit table (a preprocessor can still be used to perform explicit construction, if for any reason they are not included in the table). This approach is similar to Kanji construction out of Kana characters in Japanese, or to Hangoul construction out of elementary strokes in Korean.

Other projects using Ω concern vowelized Arabic, typesetting in Indic languages, Thai, Amharic without preprocessor, use of calligraphic fonts (such as Adobe's Poetica), redrawing of Garamont's *Grecs du Roy* etc. First releases of Ω projects are expected to take place in fall 1994.

AVAILABILITY

The METAFONT, TEX and C sources of all software presented in this paper belong to the *public domain*. They consitute a proposal for a Khmer *TEX Language Package*, submitted to the Technical Working Group on Multiple Language Coordination of the TEX Users Group and will be released after ratification. The α version of the package is currently being tested in Cambodia, and can be obtained from the author.

Khmer keyboard layouts using phonic input of consonantal clusters are currently being tested as well.

[4] Unfortunately, in TEX there is no such thing as a `\everyline` command.

REFERENCES

1. Daniel Berry and Johny Srouji, 'Arabic formatting with ditroff/ffortid', *Electronic Publishing—Origination, Dissemination and Design*, **5**(4), 163–208, (1992).
2. Y. Haralambous, 'Typesetting the holy Qur'an with TEX', in *Proceedings of the 2nd International Conference on Multilingual Computing—Arabic and Latin script (Durham)*, (1992).
3. A. Daniel, *Dictionnaire pratique cambodgien-français*, Institut de l'Asie du Sud-Est, Paris, 1985.
4. A. Daniel, *Lire et écrire le cambodgien*, Institut de l'Asie du Sud-Est, Paris, 1992.
5. D. Tonkin, *The Cambodian Alphabet*, Transvin Publications, Bangkok, 1991.
6. A. Nakanishi, *Writing systems of the World*, Charles E. Tuttle Company, Tokyo, 1980.
7. *[*Dictionnaire Cambodgien*]*. Phnom-Penh, 1962.
8. D. E. Knuth, *The METAFONTbook*, Computers & Typesetting, Addison Wesley, 1986.
9. D. E. Knuth, *The TEXbook*, Computers & Typesetting, Addison Wesley, 1986.
10. H. Kopka, LATEX*, eine Einführung*, Addison Wesley, 1991.
11. H. Kopka, LATEX*, Erweiterungsmöglichkeiten*, Addison Wesley, 2 edition, 1991.