Design of Oriental Characters with METAFONT

DON HOSEK

Platt Campus Center
Harvey Mudd College
Claremont, CA 91711
dhosek@ymir
dhosek@jarthur.claremont.edu

ABSTRACT

The goals of a meta-font can perhaps be best realized in the design of Oriental characters. These characters, unlike Western alphabets, are composed of a finite number of identifiable components. For example, the Kanji characters used by Japanese are each composed of a number of radicals which are then composed of a set of strokes. Variations in the size and appearance of these elements have a certain degree of regularity to them.

The Quixote Oriental Fonts Project (QOFP)[1] has two goals: the first is to simplify the creation of the large number of characters required by languages such as Chinese and Japanese by making the top-level description of the characters as simple as possible. Ideally the program for a single Kanji character would be composed entirely of names of radicals and mnemonic names for their placement. No coördinates would appear at all.

The second goal is to pave the way for the creation of new families of Oriental typefaces by reducing the required work to that of re-designing the component strokes of a character (and possibly making minor changes to the programs for radicals) rather than requiring the designer to modify thousands of individual character programs.

1. Introduction

Before considering the details of QOFP, it is worthwhile to consider the question of why an American with little or no knowledge of *any* Oriental language would take on a project as large as that entailed in the design of character sets for Chinese, Japanese, and Korean (there are a total of nearly twenty thousand characters involved in this endeavor).

The main reason relates to my interest in the philosophy of METAFONT. In Knuth (1982), the sentiment is expressed that ultimately we might hope to find the "essence" of a letter such as A and be able to express all possible typefaces by changing the values of various parameters. However, the long evolution of Western letterforms makes this a difficult process.[2] It seemed to me, however, that the appearance of the Kanji characters used in Oriental typefaces was ideal for implementation in METAFONT. One almost wonders if the originators of the Oriental characters foresaw the creation of a system such as METAFONT.

While it is difficult to define the "essence" of the letter "A", this is not so big a problem with the typical Kanji character. Each character is not defined so much by its total appearance as it is defined by the combined appearance of the strokes that comprise it.

My earliest efforts to create a Kanji font were frightfully bad.[3] For one thing, my lack of knowledge of Chinese or Japanese caused me to make many invalid simplifications of certain characters and to introduce unnecessary complications. After showing some of the early samples to some friends studying Japanese, I received many helpful suggestions. One of these was to define characters in terms of their

[1] The reason for choosing this name should be apparent to anyone who thinks about it.

[2] Modern Western typefaces have a variety of influences in their form stemming from the different manners in which they have been produced over the centuries.

[3] They are now hidden away in archival storage and I refuse to show them to anyone.

radicals rather than their strokes. This has two main advantages: first, it dramatically simplifies the character programs and second, it makes it far easier to correct my mistakes as I go along.

2. The Plan of Attack

Rather than rush into the character design on my second try, I spent more time on developing support code for the characters and came up with what might best be described as an "object-oriented metafont." The main principle behind this technique is to reduce the amount of information any particular portion of the code needs to "know" about the remainder of the system. For example, a typical character program might look like the following:

```
beginjchar(">", "=")(">", "'")(UNDEFINED);
triplet(1, 2, 3, 4, 5, 6);
nichi(1, 2);
nichi(3, 4);
nichi(5, 6);
endjchar;
```

In this character program, not a single explicit coördinate is specified. For that matter, there is not even any need for the program to be aware of the shapes of the radicals used either. The program for 品 is identical to 昌 with the sole exception that *nichi* is replaced with *kuchi*. Note also that the dimensions of the character itself are not given, but rather are specified elsewhere.

These two items are one of the primary things setting off QOFP from the work done by Guoan and Hobby in the old METAFONT (see Guoan and Hobby 1984). While they also took the approach of calling subroutines to put strokes and radicals together into characters, they additionally specified all dimensions of the character and internal coördinates explicitly, in this manner limiting the possibilities of the meta-ness of their font.[4]

2.1 How Meta- is my Font?

In keeping with my philosophy of object-oriented METAFONT, I decided to determine parameters for the font in a manner similar to that described by John Sauter (1986) for the Computer Modern fonts. A top-level input file would contain only the bare minimum specifications that define that typeface:

```
if unknown qjbase: input qjbase fi          % Make sure qjbase is present.

font_identifier := "QKJM"; font_size 10pt#;
font_coding_scheme "JIS";

driverfile "qkanji";
input bbqkjm                                % Generate QKJM at 10pt.
```

Through some fortunate coincidences, the syntax of the opening file has developed some pleasant regularity. Any declaration in which := appears is of no importance in the generation of the font while the remainder of the declarations are used in producing the font.

Numerous behind-the-scenes macros are used in QOFP to convert information from the human-readable format in which it is input into something that will be more useful for the system. For example, the **begin_jchar** macro examines the value given by **font_coding_scheme** to determine which pair of character codes given (if any) should be used in determining the final code used. The use of these codes is not simply confined to a basic mapping of the two-byte code to METAFONT *charcode* and *charext*: for example, if the **font_coding_scheme** is set to "jTeX JIS", the code will be automatically re-mapped into the subfont divisions used in jTEX (see Saito 1987). This partially explains the use of **default_coding_scheme** in the above example. Since qkjm10 divided into subfonts is still qkjm10, it makes sense to re-use that part of the code with the following file used as a top-level input file:

[4] They are to be forgiven for this, however, since they were among the first pioneers of METAFONT. Knuth does similar things with his character programs in Computer Modern (1986); for example, it is not possible, without changing the character program itself, to create a Century Schoolbook version of the Æ ligature from the CM description of it due to the hard-coding of such things as the height of the crossbar of the "E".

```
font_coding_scheme "jTeX JIS";
choose_subfont 8;

input qkjm10
```

which will generate the subfont indicated by `\ja` in Saito's jTEX. Note that **font_coding_scheme** was modified to not change the coding scheme if one is already in affect (which makes this particular application practicable).

By organizing the top-level input files in this fashion, it should be a fairly simple procedure to generate any font in a given family with minimal effort.

One thing worth noting is the use of "design-sized" fonts in this system. In a letter from Edgar Cooke, I was informed that at present, this practice does not occur in Japanese typography. This is doubtless due to the monumental effort that would be required to do such a task. However, just because this is not a current practice doesn't mean that it shouldn't be done. Only experience will tell whether the ability to have an Oriental font tailored to a specific typesize will indicate whether such a practice is worthwhile.

2.2 Choosing Radical Placements

Perhaps the most revolutionary aspect of QOFP is the fact that no coördinates are specified explicitly in the character program. Instead, as demonstrated above, symbolic names are given for the placement of the radicals in the characters. A cursory examination of any Oriental code table will indicate that radicals come together to form characters in a relatively small number of positional combinations. By exploiting this, characters can be easily generated. At present, I am still experimenting with the potentials of this technique, so I am unable to include any examples of the lowest level code involved.[5] The big obstacles in this approach are designing radical programs to be general enough to fit together in the different placement combinations, and prudently choosing the coördinates to be generated for the different placement codes.

Korean

The Hangul alphabet of Korean is of particular interest in this project since Hangul characters are subject to the same sort of regular placement rules as Kanji are, but to an even greater extent. It may be possible to even simplify Hangul character programs to the point where all that needs be specified for any character is the letters which compose it (assuming that the organization of Hangul characters in the Korean character set is reasonably algorithmic). I hope to have some samples of this available for display at the conference in August.

3. Plans for the Future

Ultimately, QOFP will result in at least three, possibly more, families of typefaces for the Japanese, Chinese, and Korean national character sets. Once the basic routines are working reliably, creation of a single character can be accomplished in five minutes or less (depending on how familiar the individual inputting the character definition is with the names and use of the METAFONT macros involved). Even after the project is completed, this will be a useful feature since the assorted national character sets omit many thousands of uncommon Kanji from their coding which could possibly be necessary for certain documents. Ideally, the code will be distributed freely, but economic circumstances may make this impractical.

[5] A printed copy of all METAFONT code written to date will be available for perusal at the conference.

Appendix: Oriental TEX

QOFP is currently *not* concerned with the problems of special versions of TEX for Oriental processing or with solving the Kanji input problem. However, I have collected a few thoughts on the problem of creating a suitable Oriental TEX processor and a corresponding environment for `dvi` output:

- A "big" TEX implementation which uses 64-bit words rather than 32-bit words could be used for Oriental processing (the main memory array does not necessarily need to be increased to greater than 64,000 words if this would cause problems in a low-memory environment). If the word size is 64 bits, then the size of a quarterword would then be 16 bits, which would allow for character codes adequately large enough for two-byte character sets.

- A `dvi` driver for an Oriental language which uses non-printer-resident fonts should only download the characters in the font actually used. It's a good idea to remember (for all `dvi` drivers, actually) that the character codes used by TEX do not necessarily need to correspond to those used in the printer. For example, the fact that TEX accesses some character at character code 255 does not mean that that character must be accessed as character 255 when it is used on the printer (this is especially important for those output devices which have a limit on the size of a character set less than 256).

Bibliography

Fenn, C.H. *The Five Thousand Dictionary.* Cambridge, Mass.: Harvard University Press, 1955.

Guoan, Gu and John Hobby. "A Chinese Meta-Font." *TUGboat* 5:119–136, 1984.

Knuth, Donald E. "The Concept of a Meta-Font." *Visible Language* 16:3–27, 1982.

Knuth, Donald E. *Computer Modern Typefaces.* Reading, Mass.: Addison-Wesley, 1986.

Rose-Innes, Arthur. *Beginners' Dictionary of Chinese-Japanese Characters.* New York: Dover Books, 1977.

Saito, Yasuki. "Report on JTEX: A Japanese TEX." *TUGboat* 8:103–116, 1987.

Sauter, John. "Building Computer Modern Fonts." *TUGboat* 7:151–152, 1986.

Tobin, Georgia K.M. "Designing for Low-Res Devices." *TUGboat* 9:126–128, 1988.