Glyph Choices and Techniques for Building Unicode Based Tamil Fonts

Muthu Nedumaran muthu@murasu.com
Kuala Lumpur, Malaysia

Abstract

Tamil font development has been around for a few decades. Developers have been creating fonts for screen displays and print environments. Almost all of these efforts can be classified into two categories:

The first one attempts to replace the 7bit ASCII characters with Tamil glyphs so that a user will be able to compose Tamil text with a standard keyboard. As there is an obvious limitation in the number of keys available on the keyboard, developers had ‘creatively’ dissected the glyphs so that a single key can be assigned to commonly used shapes. Examples of these are the modifiers such as the *pulli, ikara* and *iikaara* hooks, the below baseline stroke and *suzi* for *pu, puu, yuu, yuu, vu, vuu*. As the focus of the effort was to ‘somehow’ render Tamil on the screen and on print, quality and appearance of the glyphs were compromised to a great extent.

The second attempt placed the glyphs in the 8bit space, retaining the 7bit ASCII slots. This required a piece of software to map the characters to keys and there were a number that were freely available. While in the initial stages, there were so many different schemes of placing glyphs, two were commonly adopted – TSCII and TAB. Neither of these is endorsed by any standards organization but they provided the opportunity for the exchange of text in Tamil even on legacy operating systems. As there was no code-page (or character set) support in the operating system for these schemes, some slots had to be carefully avoided. This resulted in limited space to place all Tamil glyphs. Although it was a lot better than the 7bit ASCII attempt, there were still limitations. The same *ikara* and *iikaara* hooks were used over all base letters. In TAB, the *pulli* was encoded separately and thus it appeared at the edge for longer glyphs such as *nna* and *nnna*. To overcome this, monolingual schemes (such as TAM) were introduced. They addressed the quality issue but again did not comply with any standard. Also, the same font can’t be used to include even a single Roman alphabet in the Tamil text. This is an absolute requirement in bilingual environments, specifically where Tamil is used as a second language.

With more and more platforms providing system level support for Unicode and complex script rendering, it is now possible to build very high quality fonts for Tamil, assigning unique glyphs for each Tamil letter and tuning them for best appearance.

This paper will discuss the choices a font developer will have in building a Tamil font based on the Unicode standard. Techniques such as grouping glyphs for typographic effects, rendering the same text in ORNL and the new form, group wise kerning and some guidelines for shaping will be discussed.