Indic Typesetting – Challenges and Opportunities

S. Rajkumar
Linuxense Information Systems, "Lalita Mandir", 16/1623,
Jagathy, Trivandrum 695014, India
raj@linuxense.com

Abstract

Asia boasts a wide variety of scripts, most of which are complex from the perspective of a computer scientist or engineer. This is true in the case of Indic scripts which are classified in the realm of complex scripts. All of the Indic scripts require a special process to transform from the a Unicode text to the actual glyph metrics for TEX to process.

In this paper I will talk about the processing of Unicode text to produce high quality typeset material for Indic scripts using OpenType fonts. I will cover the OpenType standards for Indic scripts and other facilities OpenType provides for advanced typesetting.

Introduction

TEX has remained and continues to remain one of the best typesetting systems of the world. But with the passage of time, TEX has been used for purposes that were not taken into account during its design phase. One such "flaw" is that it was based on 8-bit tables internally. This applies among others to the number of glyph in font files and the number of characters in a language. The original 8-bit design is fine for Roman scripts but inadequate for complex scripts like Indic or CJK scripts.

When TEX was designed there were very few programs that could really do any meaningful "typesetting". Font standards were almost nonexistent. As time passes the non-TEX world is also trying to catch up with TEX. One such promising technology is OpenType. It is an industry standard font technology formed by the fusion of TrueType and Type 1. OpenType has many advanced features that were the exclusive domain of TEX so far and adds even more.

The rest of the paper dwells more on the two subjects covered above and takes a look at how they can be used to advance the capabilities of TEX.

OpenType and Internationalization

One of the most exciting internationalization development happening outside the TEX world is the development of OpenType technology [1]. OpenType is an amalgamation of earlier TrueType and Type 1 technologies and designed from ground up to support complex scripts like Indic and Arabic. It uses a full 16 bits and is based on Unicode, and thus supports 64k glyph in a single font.

OpenType also supports advanced typographical control such as ligatures, kerning, small caps etc, which were available in TEX for a long time, plus swash variants, contextual ligatures, old-style figures, multi-script baselines etc, which are not part of TEX. What sets apart OpenType from others that offer these features, including TEX, is that the renderer need not be aware of all the features available in the fonts. OpenType fonts are capable of giving this information to the rendering engine. This enables the type designer to let her imagination run loose without being hampered by the limits of the rendering engine. If it is present in the font it will be used by the renderer.

This is in contrast with the TEX, where a clear encoding of a font is required for TEX to work. This problem has been sorted out in Latin scripts where the possible numbers of glyph to render a text is finite and a robust encoding scheme is in place for font designers to follow. But not so in Indic scripts. In languages like Malayalam there are very many conjuncts or ligatures possible, and not all fonts have all of them. Unlike in Latin scripts, the ligatures are not an advanced typographic tool but an absolute requirement for legible reading.

The number and placement of glyphs for high quality typography in Malayalam is still being debated. Furthermore, the Malayalam script itself is divided into an original script and a reformed script, each with a slightly different set of glyphs and rules for vowel consonants formation.

Another problem with Indic computing in general is that there are very few high quality fonts available and thus reusing fonts is a priority. Since the entire non-TEX world is moving towards OpenType as a single font standard, more and more new fonts to appear will be based on OpenType. All this points to the fact that a possible Omega implementation that uses OpenType will be ideal for Indic languages implementation of TEX.

OpenType Rendering for Indic scripts

OpenType rendering is the process of converting the plain Unicode input into a set of glyph indexes in a font file so that the text can be rendered on a device such as a screen or printer. This is much more complex than it sounds, since the font itself contains meta-information in the form of various *features*. Features are script- and language-dependent and standard features are registered in the renderer. This enables font designer to use the registered features so that consistent high quality output is produced when rendered and renderers can be written without being tied to particular fonts.

The standards for Indic scripts are designed by Microsoft Corporation [2] and are publically available. The latest Microsoft rendering engines support most of the Indic scripts but not all. Yudit [3] is one free editor which also supports most Indic languages.

An Indic OpenType rendering engine processes text in different stages. These stages include:

1. analyzing the syllables
2. reordering characters
3. shaping (substituting) glyphs according to the instructions in the font
4. positioning glyphs

During the analysis stage the engine takes the stream of Unicode characters and finds the syllable boundaries. Once a syllable boundary is found it is analyzed to find the features that can be applied to that syllable, such as possibly combining to produce a distinct glyph. Next the base consonant is identified. All other elements are classified according to the base consonant like pre-base, post-base, etc. Next the components that appear in more than one side are split into component parts. At this point the syllable is reordered according to the appropriate rules of the language. The reordered part is passed to the glyph substitution part of the engine to obtain a reordered glyph string. The various contextual shape features are applied to this nominal glyph string to obtain the final output for rendering.

Registered Features for Indic Scripts

Registered features can be divided into language features which encode the linguistic rules; conjuncts and typographic forms, which are used for typographical substitution; and conjunct creation and positioning features which are used to position the various markings for vowel modifiers along the final glyph. The language features listed in their order of applications are:

Linguistic Rules

nukt This feature takes nominal (full) forms of consonants and produces *nukta* forms. All nukta forms must be based on an input context consisting of the full form of consonants. All consonants in a font must have an associated nukta form, and nukta forms must exist in the font for all glyphs with akhand forms as well.

akhn This feature creates an *akhand* ligature glyph from two consonants in nominal forms separated by a halant. The input context for the akhand feature always consists of the full form of the consonant.

rphf Applying this feature produces the reph glyph. If the first consonant of the cluster consists of the full form (Ra + Halant), this feature substitutes the combining-mark form of Reph. In addition, the glyph that represents the combining-mark form of Reph is repositioned in the glyph string: it is attached to the final base glyph of the consonant cluster. The input context for the Reph feature always consists of the full form of Ra + Halant.

blwf Applying this feature creates below-base forms of consonants. The input context for the 'below-base form' feature must always consist of the full form of the consonant + Halant. The feature 'below-base form' is applied to consonants having below-base forms and following the base consonant. The exception is vattu, which may appear below half forms as well as below the base glyph. The feature 'below-base form' will be applied to all such occurrences of Ra as well.

half Applying this feature produces so-called half forms: forms of consonants used in pre-base position. Half forms must exist for all consonants in the font, and half forms of nukta consonants and Akhand consonants also must exist. Use the halant form for consonants that do not have distinct shapes for half forms. This feature is not applied to the base glyph even if the syllable ends with a halant.

pstf Applying this feature creates post-base forms. Examples include Bengali and Oriya 'Ya' and Malayalam 'Ya' and 'Va'.

vatu Vattu variants are formed by combining consonants with the vattu mark. Vattu ligatures can be either half or full form, and fonts must contain both. The input context for the 'vattu variants' feature must always consist of a consonant (in full or half form) + vattu glyph.

Conjuncts and Typographic Forms The conjuncts and typographical features are not strictly required, but almost all fonts will have to use some of these to produce readable text.

pres Pre-base substitutions: this feature produces conjuncts with half forms, the type most common in Devanagari. This feature also produces the correct shape of I-Matra (in Devanagari and similar scripts) and also may take care of pre-base matra ligatures like Tamil 'elephant trunk' shape of AI-Matra.

blws Below-base substitutions: This feature produces conjuncts of the base glyph with below-base consonants. Any specific context-dependent forms of below-base consonants are handled here as well. Finally, this feature produces matra ligatures with the base consonants and below-base stress and tone marks.

abvs Above-base substitutions: This feature produces the correct typographic shape when an above-base matra forms a ligature with the base glyph. This feature also produces conjuncts of the base glyph or matra with Reph, ligatures and forms involving above-base vowel modifiers and above-base stress and tone marks.

psts Post-base substitutions: This feature produces ligatures of the base glyph with post-base forms of consonants. It also produces the correct typographic shape when a post-base matra forms a ligature with the base glyph and different forms of post-base vowel modifiers like visarga.

haln Halant form of consonants: This feature produces the halant form of the base glyph in syllables ending with a halant. This features also takes care of *chillaksharams* in Malayalam.

Positioning Features

blwm Below-base marks: This feature positions all below-base marks on the base glyph.

abvm Above-base marks: This feature positions all above-base marks on the base glyph or the post-base matra.

dist This feature covers all other positioning lookups defining various distances between glyphs, such as kerning between pre and post-base elements (like Visarga) and the base glyph.

Moving Ahead

Currently the Omega typesetting system does not support OpenType technology. But this is one of the planned features for Omega in future.

References

[1] Adobe Corporation. `http://www.adobe.com/type/opentype/main.html`

[2] Microsoft Corporation. `http://www.microsoft.com/typography/otfntdev/indicot/features.htm`

[3] Sinai, Gasper. `http://www.yudit.org/`