Creating Unicode Compatible OpenType Telugu Fonts

Tirumala Krishna Desikacharyulu
Winnipeg, Manitoba, Canada

Abstract

Pothana2000 is the first Unicode compatible OpenType font designed for Telugu. It was designed under Windows 2000, with generous help from Apurva Joshi of Microsoft's Opentype Division. This font was working in Windows 2000 even before Microsoft officially released the Gautami Opentype Telugu font with Windows XP Professional. In this paper I discuss the practical aspects of designing the Pothana2000 font so as to provide stimulus and guidance to other Telugu font designers.

1.0 Introduction

With Windows 2000 Microsoft started supporting Indian languages. The initial support was limited to Hindi, Sanskrit, Marathi, Konkani and Tamil. This support was extended to Telugu and Kannada and some other languages in their Windows XP Professional operating system, and in their Office products such as Word XP. With XP, Microsoft also provided an Opentype Telugu font named Gautami. But it is a proprietary font, not available without license from Microsoft. Although Gautami serves the basic needs of the operating system, it is not the best looking font for aesthetic composition of Telugu documents. So, there is a great need to develop aesthetically appealing Telugu fonts for publication of Telugu documents and books.

I was active in Telugu font design for more than a decade. My Pothana font, which was designed for pre-2000 Windows systems is widely used for publications in North America. However, like many other Telugu fonts designed in these environments, it doesn't conform to any standard. So, when Microsoft started supporting Unicode Indian fonts in Windows 2000, I upgraded this font to Unicode compatible OpenType format, thus making it the first Telugu font working in this environment. I have placed it under GPL (General Public License), making it accessible to researchers and the Telugu language enthusiasts world wide. In this paper I will go into some details of designing this font, hoping it stimulates similar work from other enthusiastic font designers.

2.0 Theoretical Background

2.1 The Unicode Script Processor (Uniscribe)

The processing of Unicode fonts under Windows proceeds as follows. Each language is given a locale, which defines the language environment, including the keyboard input. This locale definition maps user's key strokes to the appropriate Unicode range. For ex.., when you choose an English locale and press A, a decimal code of 65 is passed to the computer. If you define a

locale for Telugu that maps A to అ, a decimal code of 3078 is passed to the computer when you press A. An application such as Word receives this keyboard input and lays out text in proper format with the help of the Unicode Script Processor (USP10.dll), which comes with the operating system. This processor, called "Uniscribe" in short, contains APIs that facilitate applications in formatting complex scripts. Uniscribe has multiple "shaping" engines, which know how to layout multiple complex scripts.

Uniscribe subdivides a character string into "items", "runs", and "clusters". An Item is a set of characters having the same script and direction attributes; Runs are portions within an item that have the same formatting attributes; Clusters are the script-defined, indivisible character groupings. An application such as Word XP builds runs based on its own stored formatting attributes, and on item boundaries obtained from Uniscribe. Using Uniscribe, client applications only need to manage a buffered storage of Unicode character codes, input by the user in phonetic order. This buffered sequence of characters are passed on to Uniscribe which does the layout of the script.. As such this buffered sequence of characters remains intact, untouched by the layout operations of the Uniscribe. Actually this sequence is what is stored in the Unicode text file.

2.2 OpenType Shaping Engine (OTL Services)

Windows also contains an OpenType Shaping engine for handling script features unknown to Uniscribe. These are the features embedded into Opentype fonts. An OpenType font is an extended version of a TrueType font, which may contain one or more additional tables to handle features such as contextual glyph substitutions, relative glyph positioning, kerning etc. This engine frees the client applications from handling complexities of script rendering, such as applying contextual glyph substitutions, relative glyph positioning etc. Thus the applications work at the level of features and characters, with which they are already familiar, and let OTL Services handle the details of lookup tables, glyph Ids, which are built into the font.

2.3 How The Telugu Shaping Engine Works

In this paper, we are only interested in Telugu shaping engine of Uniscribe. Here a brief description of this engine is given based on the detailed documentation published on Microsoft's web site, http://www.microsoft.com/typography/otfntdev/teluguot/shaping.htm. It is essential to understand how this engine works because we have to design the font to conform to its working. The Telugu shaping engine works on a Telugu Unicode character string in the following stages:

1. Identifies syllables
2. Reorders characters within the syllables
3. Shapes glyphs using OTL services
4. Positions glyphs using OTL services

First the shaping Engine identifies the boundaries of syllables and isolates certain parts. It does this to be able to place syllable elements in required positions and to determine if combining elements have to be preferentially replaced by graphically distinct forms, such as the

syllable kssa (క్ష). Then the base consonant, which is displayed in full form, is found, and all other elements are classified by their position relative to this base. These other elements could be pre-base (non-existent in Telugu), below-base (quite common), above-base (quite common) and post-base (quite common). The shaping engine splits the matras that occur on more than one side of the base consonant into their constituents.

The phonetic sequence of Unicode characters initially passed on to Uniscribe may not be in the order that it can work with. So, Uniscribe reorders this sequence and maintains a buffer of such appropriately reordered character codes, delineated as "clusters". It then gets the appropriate glyph substitutions for these clusters where necessary, using OTL services. Since Uniscribe uses reordered character sequences to render the font on the screen, it is essential that the font designer take this behavior into account when designing the font. This re-ordering is done according to the following rules:

1. The base consonant is found by the following algorithm: starting from the end of the syllable, move backwards until a consonant is found that does not have a below-base or post-base form (post-base forms have to occur after below-base forms), or arrive at the first consonant. The consonant stopped at will be the base consonant.

2. If the base consonant is not the last one, then Uniscribe moves the halanth from the base consonant to the last one.

3. Uniscribe splits two part matras such as AIMatra into above base and below base components. In Telugu there is only one such case, i.e., AI Matra ై. This is split into ె్, i.e., into an above-base E Matra (code 0C46) and a below-base AI Length Mark (code 0C56).

4. Uniscribe classifies consonants and matra parts as above-base, below-base and post-base. This classification is a language feature, not a font dependent feature. For Telugu this classification is as follows:.

Below-base Markers: All consonant markers (i.e., Ottulu), AI length mark, Anudatta. and the matras corresponding to Vocalic L and Vocalic LL. Note that the latter three markers are not yet defined in the Telugu Unicode range, which they must be. They are mentioned here for completeness.

Above-base Markers : The matras corresponding to vowels ఆ, ఇ, ఈ, ఎ, ఏ, ఒ, ఓ,ౌ, Udatta, Double Udatta, and Nukta. Note that Udatta, Double Udatta and Nukta are not yet defined in the Telugu Unicode range, which they must be. [1]They are mentioned here for completeness.

[1] Here I am assuming that we can define a Nukta marker in Telugu similar to the Devanagari Nukta, to obtain dantya (dontal) ca and ja. Another way of achieving this is to allocate two additional code points for these two characters in the Telugu Unicode range. This is the

Post-base Markers : The matras corresponding to vowels ఉ, ఊ, ఋ, ౠ (i.e., Kommu, Kommu-deerghamu, Rutvamu, Rutvam-deerghamu) are post-base matras. Although the Anuswara (Sunna), Ardhanuswara (Arasunna), Visarga and Avgaraha occur as post-base characters, they do not affect the shape of the base consonant or ligature, so they are classified as Vowel Modifiers (VMs), not as post-base matras. Note that Avagraha is not yet defined in the Telugu Unicode range, which it must be. It is mentioned here for completeness.

There are no pre-base forms in Telugu.

5. Uniscribe then groups elements of the syllable (consonants and matras) according to this classification. Thus pre-base elements precede the base consonants, and the above-base, below-base and post-base components will follow the base glyph. Halanths are moved with the consonants they affect.

These principles are illustrated by the following example: Consider the Sanskrit word

లక్ష్మ్యై = lakshmyai = for Lakshmi.

This word has 2 syllables la (ల), kshmyai (క్ష్మ్యై). .The first syllable is very simple consisting of a single base consonant. So it doesn't need any rearrangements, substitutions or relative positioning. The second syllable is more complex, so let us analyze it further:

User's Phonetic Input	Keyboard Output	Rearranged by Uniscribe
క్ష్మ్యై	క్ష్మ్యై	క్ష్మ్యై

When the user enters క్ష, the codes క్ష are passed on to the computer because Unicode doesn't define a code point for క్ష, but treats it as a composite syllable consisting of the characters క్ష. Yet, క్ష is provided on the keyboard because it occurs very frequently and it will be very clumsy to enter 3 key strokes to compose it every time. So, when the user presses the key corresponding to క్ష, the keyboard driver decomposes it into Unicode compatible character cluster క్ష. Uniscribe knows that it has to substitute a single glyph (క్ష) for this cluster, irrespective of what transformations it has to apply to other characters that follow it. Thus when క్ష and its

---

preferred option. Even if these are allocated, it is still a good idea to have Nukta in Telugu to provide extendability of the script to foreign sounds.

vowel-conjugated (matra) forms occur (in this case క్షై), Uniscribe applies a language feature called "akhand" to process these glyphs first before processing any others. So, it rearranges the character sequence as కెషం by moving the Halanth (ం) after Ssa (ష), and moving in the Ai Matra marker (ె) next ka (క), and splitting it into its above-base and below-base components (ై). It then applies the akhand feature to get క్షై. Given the character sequence కెషం, it is actually the OTL services that render this final glyph using the lookup tables and glyphs defined within the OpenType font. Having taken care of this "akhand" feature, OTL services are used again to transform the remaining string మం యం by applying below base forms feature ("blwf") to get ma and ya consonant markers (Ottulu). Although complex, this example by no means illustrates all the features and transformations required to produce the entire Telugu script. In general terms syllables before being reordered by Uniscribe have the following form:

1. {C+[Nukta]+H}+ C + [M]+[VM]+[SM]
   - A syllable with consonants and vowels, Ex., the syllable "ర్కైఃలో" in the word "శార్కైఃలో".
2. {C+[Nukta]+H}+ C + H
   - A syllable without vowels. Ex, the syllable "ష్ట్ర్యి" in the word "రాష్ట్ర్యి"
3. VO+[VM]+[SM]
   - A syllable without consonants. Ex., "ఓం"

After reordering by Uniscribe, the above syllables have the following forms, respectively:

1. Cbase + [Mabove]+[Mbelow]+[Mpost]+{Cbelow+H}+[VMpost]

2. Cbase + {C+H}+ H

3. VO + [VM1] + [VM2]

where C = Consonant character
*Nukta = Nukta marker
H = Halanth (commonly known as Pollu ం to Telugu people)
M = Matra marker (Such as AAMatra ా, EMatra ె, OMatra ొ, UUMatra ూ etc.)
VM = A Vowel modifier character or glyph (such as Visarga ఃఽ, Anuswara ం etc.)
*SM = A stress Mark, such as Udatta ం॑, Anudatta ం॒ etc.
VO = A free standing (independent) vowel such as అ, ఇ, ఊ,ఏ,ఐ etc.
{ } - indicates 0 or more occurrences; [ ] - indicates 0 or 1 occurrence.

* These are not yet defined in the Telugu Unicode range, but included here for completeness.

These rules result in the following character sequences for the three example syllables:

Syllable	Input Char Sequence	Rordered Character Sequence
1. ర్జ్గైః	ర ్ జ ్ గ ై ః	ర ె ౖ జ ్ గ ్ ః
	C+H+C+H+C+M+VM	Cbase + Mabove+Mbelow+2*(Cbelow+H)+VMpost
2. ష్ట్ర	ష ్ ట ్ ర ్	ష ట ్ ర ్ ్
	C+H+C+H+C+H	Cbase + 2*(C+H) + H
3. ఓం	ఓ ం	ఓ ం
	VO+VM	VO+VM

The above three examples illustrate how Uniscribe splits AImatra into its above-base and below-base components and rearranges the initial phonetic sequence of characters. It is this reordered sequence that is formatted by OTL services using the lookup tables built into an OpenType Telugu font. So, when building the lookup tables, the font designer must have a clear concept of this Uniscribe's behaviour.

2.4 Features For the Telugu script

Taking Uniscribe's reordered sequence of characters as input, the shaping engine applies various shaping and positioning features in a fixed order. So, the font designer must build these features into the font and sequence them in the right order to get proper shaping and positioning of the glyphs that make up a syllable. All these features are implemented through lookup tables built into the font. Here the word "syllable" is important because all features operate within one orthographic syllable. The features that shape Telugu script are given below in the order they are applied:

Language based forms:

1. akhnd - Akhnad feature.

   This feature applies to the cluster that makes up the syllable KSSA (క్ష) and its matra variants and shapes them in preference to all others. We have already discussed this in section 2.3.

2. blwf - Below-base form.

   This feature shapes the consonant markers (Ottulu) in the Telugu script. Recall that all the consonant markers are below-base glyphs in the Telugu script. All the consonant markers (Ottulu) discussed in section 2.3 were shaped by this feature.

Conjuncts and typographical forms:

1. blws - Below-base substitution

   This feature produces conjuncts of the base glyph with below-base consonant variants (Ottulu). Context for this feature is C + {Ottu} -> Consonant Conjunct. Examples: ష + ◌ట్ + ర ఒత్తు -> ష్ట్ర ; ష + ట ఒత్తు + ల ఒత్తు -> ష్ట్ల ; ర్ + ర ఒత్తు -> ర్ర్ etc.

2. abvs - Above-base substitution

   This feature shapes the above-base matra conjuncts, and (not yet implemented) above-base accent marks. Recall that the matra conjuncts corresponding vowels ఆ, ఇ, ఈ, ఎ, ఏ, ఒ, ఓ, ఔ fall into this class. Udatta, Double Udatta, and Nukta are the (not yet implemented) above-base stress marks. Context for this feature is C + M -> Matra Conjunct. Examples: క + ◌ా -> కా; గ + ◌ి -> గి; ఘ + ◌ే -> ఘే; ర్ + ◌ో -> ర్ఓ etc.

3. psts - Post-base substitution

   This feature produces conjuncts with post-base vowel markers. In Telugu the matras ◌ు, ◌ూ, ◌ృ, and ◌ౄ are the post-base markers. Of these only ◌ు and ◌ూ need contextual substitution. Context for this substitution is C + {Marker} -> Consonant Conjunct. The matras ◌ృ and ◌ౄ do not need such substitution and can be treated similar to Anu-swara (◌ం), Visarga (◌ః), Arasunna (◌ఁ), and Avagraha (ఽ). Examples:

   ప + ◌ు -> పు; ప + ◌ూ -> పూ; ర్ + ◌ూ -> ర్ూ; ఘ + ◌ు -> ఘు; జ + ◌ూ -> జూ etc.

Halanth forms:

1. haln - Halanth-form substitution

   This feature produces halanth forms where a consonant or a ligature ends with a pollu (halanth) Examples:
   హ + ◌్ -> హ్; క్ష + ◌్ -> క్ష్; ఱ + ◌్ -> ఱ్; భ + ◌్ -> భ్; గ్ + ◌్ -> గ్, etc.

Positioning features:

1. abvm - above-base mark positioning

   This feature can be used to position above-base matras (such as EMatra) and swara markers (ex., Udatta) with the base glyph. This feature is not used by Pothana2000, so we don't discuss it further.

2. blwm - below-base mark positioning

   This feature can be used to position below-base marks with the base glyph. This feature is extensively used by Pothana2000 to position below-base consonant markers (Ottulu) with the base glyph. The impressive positioning of Ottulu in syllables such as ష్ప, క్ర, ర్క్ etc. is achieved by this feature.

2. dist - Distances

   This feature can be used to adjust relative position of base and dependent glyphs in a syllable. This feature is not currently used by the Pothana2000 font, so we don't discuss it further.

So, to summarize, the processing of Telugu script takes place as follows. The user inputs the text in phonetic order from the keyboard. The language locale (driver) maps user's input into appropriate Unicode range for the language. The Uniscribe re-orders the characters in user's input if required. Then features discussed above are applied on this re-ordered character sequence in the order specified above to shape and render the syllable.

3.0 The Pothana2000 Font

All the character outlines in the Pothana2000 font were designed using the font editor Fontographer 4.1. This file was then ported into Microsoft's Visual OpenType Layout Tool (VOLT) to add lookups corresponding to the features described above. Here I describe the architecture of this font as one example of building a Telugu font. I do not claim this is the best or optimal way of doing it, as there could be many alternate designs. This font contains 630 glyphs, which are divided into the following categories:

1. Punctuation symbols and control characters: 39
2. Arabic Numerals: 10
3. Telugu Numerals: 10
4. Zero Width Joiner and Non-joiner : 2
5. Avagraha , Nukta, Vocalic L Marker, Udatta, Anudatta (all taken from Devanagari range): 5
6. Vowels defined in Telugu Unicode range: 16
7. Vowel markers (Matras) defined in Telugu Unicode range: 14
8. Anuswara, Arasunna, Visarga and Halanth (Virama) : 4
9. Consonants defined in Telugu Unicode range and KSSA (క్ష): 36
10. Below-base consonant markers of the above consonants (Ottulu): 36
11. Vowel conjugated (Matra) forms of the above consonants: 360
    These are limited to Matras ా, ి, ీ, ు, ూ, ె, ే, ొ, ో, ౌ

12. Dantya ca, ja and their matra forms with Matras ా, ు,ూ, ృ,ౄ,ె: 14

13. Halanth forms of all the above consonants: 38
14 Other contextual forms and markers: 46

The font includes Nukta and the dantya ca, ja and their matra forms with the hope that they will be supported in future versions of Unicode. It also includes Udatta, Anudatta, Double Udatta, and vocalic L markers with the same expectation. Actually, the Unicode standard says that the markers defined in one script's range can also be used in other ranges, but Microsoft's implementation doesn't allow this, if any contextual shaping and positioning involves these "foreign" markers. Hence, only Avagraha works well as borrowed from Devanagari at present.

Because most of the control characters and punctuation symbols come from Latin1 range, and some other symbols/markers come from Devanagari range, these languages are declared within the font definition without any features. Telugu language is declared with all the features, because all the shaping and positioning in this font occurs only for Telugu glyphs. The feature definitions within the Pothana2000 font arranged in the order they are processed is given below:

Table 1: Features defined in the Pothana2000 Font

Feature	**What it processes**
Nukta forms <nukt>	Nukta Consonant forms. Note: this feature doesn't work because Nukta is borrowed from the Devanagari range as it is not defined in the Telugu range. It is included for completeness, hoping that future revisions of Telugu Unicode allow this feature, hence the dantya ca and ja and their volwel-conjugated (Matra) forms.
Akhand <akhn>	క్ష , క్ష Ottu and క్ష Matra forms
Below-base forms <blwf>	Below-base forms (Ottulu) of all consonants except క్ష. The క్ష ottu is processed by the akhn feature.
Below-base substitutions <blws>	Performs the following substitutions: a) Substitutes two consecutively occurring below-base consonant markers (Ottulu) by a single double ottu form. Ex., ష ్ ట ్ ల -> ష్ట్ల; ర ్ ష ్ ణ -> ర్ష్ణ b) In syllables such as ష్ట్లై, where AI Length mark and below-base ottu occur together, it first substitutes AI Length mark by an alternate form of AI Length mark and then maps this AI Length mark and below-base ottu combination by a properly formed ottu-AI length marker glyph. c) In syllables such as ష్ట్రి, ర్ద్రి, క్రి,ష్రీ, ర్ష్రి where below-base ottu and ra-ottu combinations, or AI Length Mark and ra-ottu combinations occur,

	it substitutes the below-base ra-ottu by its post-base variant. d) In syllables such as గ్ర, మ్ర, ఘ్ర,ర్ఘ్ర, where a below-base ra-ottu occurs, it replaces the default below-base ra-ottu with wider variants or a post-base variant of it, depending upon the width of the base glyph or ligature.
Above-base substitutions <abvs>	This feature processes the consonant-vowel conjugated forms where the context is a consonant followed by an above-base matra (ా, ి, ీ,ు, ూ, ె,ే, ై,ొ,ో,ౌ are the above-base matras). Examples: క ా -> కా, క ◌ర ి -> క్రి, ష్ ◌ణ ే -> ష్ణే, గ ై -> గై, ఘ ొ -> ఘొ etc.
Post-base substitution <psts>	Performs the following substitutions: 1. When a consonant is followed by the post-base matras ు, ూ it obtains a proper consonant-vowel conjugated form. Examples: క ు -> కు, క ూ -> కూ, ష్ ◌ణ ూ -> ష్ణూ, , గ ◌ర ూ -> గ్రూ etc. 2. When the above context occurs with ligatures involving a ra-ottu, it substitutes the default below-base ra-ottu by its variants depending upon the width of the ligature. Ex: క్ర ు -> క్రు, ర్ఘ ◌ర ు -> ర్ఘ్రు, ఖ ◌ర ు -> ఘ్రు; మ ◌ర ూ -> మ్రూ; ఫ ◌ర ూ -> ఫ్రూ etc. 3. It also substitutes the special form of ppu-ottu. Examples: ప ◌ప ు -> ప్పు; ప ◌ప ◌ర ు -> ప్ప్రు
Halanth forms <haln>	Puts the base consonant in halanth form when the syllable ends with a halanth. Examples: క ◌ -> క్; ఱ ◌ -> ఱ్; ఘ ◌ -> ఘ్ ప ◌ -> ప్; ప ◌ప ◌ -> ప్ప్; స ◌ట ◌స ◌ -> స్ట్స; ష ◌ట ◌ర ◌ -> ష్ట్ర; ర ◌ధ ◌య ◌ -> ర్ధ్య etc.
Below-base mark positioning <blwm>	This feature includes a number of positioning lookups which place the below-base glyphs derived by previous features into aesthetically appealing positions with respect to the base glyph. All the complex syllables such as స్ట్స, మ్రూ, ష్ట్ర, ష్ణూ etc., shown in all previous examples employ this feature to place below-base marks aesthetically below the base glyphs.

The above features enable the Pothana2000 font to render the Telugu text in a pleasing manner. Here is some text to illustrate this point, composed in 12 point size Pothana2000 font.

శ్లో॥బ్రాహ్మీ తు భారతీ భాషా గీర్వా గ్వాణీ సరస్వతీ/ వ్యాహార ఉక్తిర్లపితం భాషితం వచనం వచః॥

అపభ్రంశోఽపశబ్ద స్స్యాత్ శాస్త్రే శబ్దస్తు వాచకః/ తి ఙ్సు బన్తచయో వాక్యం క్రియా వా కారకాన్వితా॥ (అమరకోశము)

శా॥ శ్రీకైవల్యపదంబుఁ జేరుటకునై చింతించెదన్ లోకర
క్షైకారంభకు, భక్తపాలనశాసంరంభకున్, దానవో
ద్రేకస్తంభకుఁ, గేళిలోలవిలసద్దృగ్జాతసంభూతనా
నాకంజాతభవాండకుంభకు, మహానందాంగనాడింభకున్. --- (శ్రీమద్భాగవతం - బమ్మెర పోతన)

శా॥ పశ్యత్ఫాలవిలోచనానలశిఖాభాస్వన్మహోమిత్ర, మి
త్రశ్యామారమణేక్షణప్రవణచేతఃపద్మ, పద్మాంగభూ
దృశ్యాగద్యుతిసంగ, సంగరమదోద్వృత్తారిరాహుత్తరా
డ్వేశ్యాలోకభుజంగ, జంగమనగద్వేదండపూర్ణాంగణా. ---- (వసుచరిత్ర - రామరాజభూషణుడు)

శా// వర్ధిష్ణుప్రతిభాపరాశర, శరస్వర్ధేను వాణీసుధా
వార్ధిస్పర్ధియశోధురంధర, ధరధ్వంసిస్వరుక్రీమహా
దోర్ధామర్ధివిశేషభాసుర, సురస్తోమస్తుతాజిక్షమా
దుర్ధర్షధ్వజినీసముత్కర, కరాధూతేందుధారాధరా. ---- (వసుచరిత్ర - రామరాజభూషణుడు)

మఱియుఁ దత్కాసారం బపరిమితపాండుపక్షవిశేషభూషితంబు గావునఁ బ్రదర్శితధార్తరాష్ట్రముఖశ్యామికంబై, బహుల హరివిలాసభాసురంబు గావునం బ్రథమానమీనకూర్మాద్యవతారధారకంబై, యూర్మిలాభాసమానంబు గావున లక్ష్మణోప భోగభవనంబై, మానసజయమహనీయంబు గావున నంతరున్మీలితనాళికోపరివికస్వరకమలకర్ణికానిర్మిరోధబోధమాధురీ ధురీణంబై రాణించుట కుదంచితరోమాంచకంచుకితదేహలై యవ్వారారోహలు ముదంబునం దదంబుఖేలనాయత్తచిత్తలై .... --- (వసుచరిత్ర - రామరాజభూషణుడు)

4.0 A Locale Definition for Telugu

When I first designed Pothana2000 on Windows 2000, there existed no Telugu locale in that environment. So, I had to create my own locale using the Tavultesoft Software's shareware utility named Keyman (http://www.tavultesoft.com). Keyman versions 5 and 6 are excellent utilities for creating locales. Version 6 also provides an environment in pre-2000 Windows systems to input Telugu using Unicode compatible fonts such as Pothana2000. At that time I looked critically into the issue of designing a user friendly keyboard for Telugu in particular and for Indian scripts in general, because the usability of a script on computer directly depends upon the ease of its entry on the keyboard. The main criteria used in designing my Telugu keyboard were the following:

a) Familiarity

As can be seen from the ISCII standard, all Indian scripts basically have the same alphabet set although their shapes differ from language to language. So, they can use the same phonetic keyboard, perhaps with minor exceptions. Also, along with Hindi, English happens to be a link language in India in which people from different linguistic regions communicate. So, they have a need to type in English, Hindi, and in their own language. Most literate Indians have a working knowledge of English and they have used a QWERTY type writer or computer keyboard before. In fact, they are more familiar with a QWERTY keyboard than Indian language keyboards, because the latter started appearing only recently. So, it makes sense to map Indian characters to homophonic locations on a QWERTY keyboard so that they do not have to switch to a very dissimilar keyboard environment when they alternate between Indian scripts and English. **Another primary consideration is that the punctuation symbols and numerals map to their customary locations**. In other words, these keys should not be allocated to any of language characters; they should mean the same irrespective of the language locale used. **It is best to maintain this conformity as much as possible.**

Even if a person does not know or use the English language, it is still helpful to maintain this conformity, because if he/she needs to type in English at a later stage, he/she can leverage this familiarity with the native language keyboard to type English easily. **One has to learn a keyboard anyway; let its design be logical enough to help the person input other languages easily if need arises.**

One can argue that the keyboard mapping should be based on the frequency of occurrence of various characters within the language. In fact I have looked at this aspect for the Telugu language, and have come to the realization that QWERTY like mapping is not too far from such a frequency based configuration. In any case, familiarity and the ability to input Indian and English languages without changing keyboard configurations should be very important.

b) Ease of use

The keyboard should be easy to use with most commonly used characters easily accessible on the keyboard. It is assumed that characters typed by holding down control keys Shift, AltGr, Shift AltGr are harder to type in this order than unshifted characters. I.e., if we grade the ease of typing from 1 to 4 (1 being the best and 4 being worst), then k, Shift k, AltGr k, Shift AltGr k fall into grades 1 to 4, respectively, where k stands for a key that is pressed.

c) Logical Design

Characters should be logically mapped. For example, the mahaprana (aspirate) counterparts of letters ka (క), ga (గ), ca (చ), ja (జ), tta (ట), dda (డ), ta (త), da (ద) should be mapped to shifted positions of the corresponding keys. I.e., if ka (క) is mapped to k, kha (ఖ) is mapped K (Shift k) etc. Similarly, the less frequently occurring nukta forms of consonants ka (క), kha (ఖ), ga (గ), ca (చ), ja (జ), dda (డ), ddha (ఢ), pha (ఫ)and ya (య) should be mapped to AltGr k, Shift AltGr k etc. The main idea here is to map all variants of a given character to the same key, and aceess them using Shift, AltGr and SHIFT AltGr control key combinations where needed.

Extending the above consideration to vowels, the short and long vowels should be mapped to unshifted and shifted positions, and the corresponding Matra markers to AltGr and Shift AltGr positions of the same keys. Here it is important to notice that the Unicode specification requires that in addition to the free standing vowels their matra symbols should also be present on the keyboard. In fact to get matra-conjugated forms of consonants, one has to type the matra symbol after the consonant. This means that the keyboard should contain 13 additional symbols corresponding to matra markers of AA(ఆ), I(ఇ), II(ఈ) , U (ఉ), UU(ఊ), Vocalic R(ఋ), Vocalic RR(ౠ), E(ఎ), EE(ఏ), AI(ఐ), O(ఒ), OO(ఓ) and AU(ఔ). In an attempt to accommodate this requirement, some keyboard designs, such as the INSCRIPT keyboard, have used up the locations normally used by the familiar punctuation symbols with the result that to get these frequently used symbols one has to switch to a new keyboard locale. INSCRIPT also maps the Indian characters to totally unintuitive locations on a QWERTY keyboard, making it extremely difficult to combine English with the Indian language text. This inconvenience can be avoided if one were to adopt a QWERTY like mapping with some simple contextual dependency built into the keyboard driver. For example, look at the following extract from my locale's definition:

```
store(VowKeys)  'aAiIuUeEYoO[fFqQ]{}'
store(VowCodes) U+0C05 U+0C06 U+0C07 U+0C08 U+0C09 U+0C0A U+0C0E U+0C0F\
                U+0C10 U+0C12 U+0C13 U+0C14 U+0C02 U+0C01 U+0C0B U+0C60 \
                U+0C03 U+0C0C U+0C61
store(ConsKeys)  'k' 'K' 'g' 'G' 'M' 'c' 'C' 'j' 'J' \
                 'V' 'z' 'Z' 'N' 't' 'T' 'd' 'D' 'n' \
                 'p' 'P' 'b' 'B' 'm' 'y' 'r' 'R' 'l' \
                 'L' 'v' 'S' 'x' 's' 'H' 'h' 'w' 'W'
store(ConsCodes) U+0C15 U+0C16 U+0C17 U+0C18 U+0C19 U+0C1A U+0C1B \
                 U+0C1C U+0C1D U+0C1E U+0C21 U+0C22 U+0C23 U+0C1F \
                 U+0C20 U+0C26 U+0C27 U+0C28 U+0C2A U+0C2B U+0C2C \
                 U+0C2D U+0C2E U+0C2F U+0C30 U+0C31 U+0C32 U+0C33 \
                 U+0C35 U+0C36 U+0C37 U+0C38 U+0C39 U+0C4D U+0C24 \
                 U+0C25
```

The first store statement declares an array of keys that map stand-alone (primary) vowels, which are identified by the Unicode points specified in the second store statement. Likewise, the 3rd store statement declares an array of keys that map to the Unicode points specified in the 4th store statement. This is how one can map the Telugu characters to the keys on a QWERTY key-board. Now, consider the following statements:

```
any(Conscodes) +  'A' > context  U+0C3E
any(Conscodes) +  'i' > context  U+0C3F
any(ConsCodes) + 'I' > context  U+0C40
any(ConsCodes) + 'u' > context  U+0C41
any(ConsCodes) + 'U' > context  U+0C42
any(ConsCodes) + 'e' > context  U+0C46
```

any(ConsCodes) + 'E' > context U+0C47
any(ConsCodes) + 'Y' > context U+0C48
any(ConsCodes) + 'o' > context U+0C4A
any(ConsCodes) + 'O' > context U+0C4B
any(ConsCodes) + 'I' > context U+0C4C
any(ConsCodes) + 'q' > context U+0C43
any(ConsCodes) + 'Q' > context U+0C44

These statements say whenever the user types one of the primary vowels after a consonant, change the context to map this primary vowel to its matra code, which is specified on the right hand side of the expression. This works very well because a free-standing vowel very rarely occurs in the middle or at the end of a syllable in Indian scripts. Thus we have:

క ఆ -> కా; క ఇ -> కి; క ఉ -> కు; క ఊ -> కూ; క ఋ -> కృ; క ౠ -> కౄ;

క ఎ -> కె; క ఏ -> కే; క ఐ -> కై; క ఒ -> కొ; క ఓ -> కో; క ఔ -> కౌ; శ ర ఈ -> శ్రీ;

క్ష ఐ -> క్షై; ర త స న య ఆ -> ర్త్స్న్యా etc.

So, in this implementation, we never need to type matra markers explicitly to obtain matra-conjugated consonant forms, so they can be relegated to hard to access AltGr, and Shift AltGR locations. This saves us valuable "real estate" on the primary keyboard, which can be allocated to punctuation symbols. This is exactly what my locale does. In rare cases when a free-standing vowel has to be typed immediately after a consonant without combining it with the consonant, the context can be broken by typing the zero width non-joiner character in between (zero width non-joiner is mapped to AltGr . in my locale).

The keyboard design based on these considerations is given in the appendix I.

Acknowldgements

I am thankful to Apurva Joshi of Microsoft Corporation for clarifying many points about OpenType layout features and to Dr. Uma Maheswara Rao for encouraging me to prepare this paper.

Appendix 1: Keyboard Mapping for Pathana2000

(Shifted - top to bottom)

~	!	@	#	$	%	^	&	*	(	)	_	+
~	!	@	#	$	%	^	&	*	(	)	_	+

Q	W	E	R	T	Y	U	I	O	P	{	}
ఋూ	థ	ఏ	ఱ	ర	ఐ	ఊ	ఈ	ఓ	ఫ	ఌ	ౌ

A	S	D	F	G	H	J	K	L	:	"
ఆ	శ	ధ	◌ె	ఘ	హ	ఝు	ఖ	ళ	:	"

Z	X	C	V	B	N	M	<	>	?	|
ఢ	ష	ఛ	ఞ	భ	ణ	ఙ	<	>	?	|

(Unshifted - top to bottom)

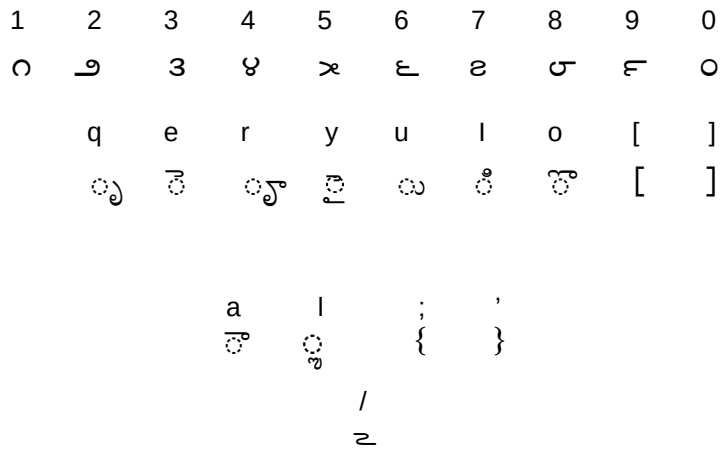
‘	1	2	3	4	5	6	7	8	9	0	-	=
`	1	2	3	4	5	6	7	8	9	0	-	=

q	w	e	r	t	y	u	i	o	p	[	]
ఋు	త	ఎ	ర	ట	య	ఉ	ఇ	ఒ	ప	ౌ	◌ఃు

a	s	d	f	g	h	j	k	l	;	’
అ	స	ద	◌ం	గ	◌ా	జ	క	ల	;	'

z	x	c	v	b	n	m	,	.	/	\
డ	ష	చ	వ	బ	న	మ	,	.	/	\

(With AltGr - top to bottom)



In addition, AlltGr , and AltGr . are mapped to ZWJ (zero width joiner) and ZWNJ (zero width non-joiner), respectively.

(With Shift AltGr)