PERGAMON

Computers & Graphics 23 (1999) 7–24

COMPUTERS & GRAPHICS

Computer Graphics in India

An architecture for the shaping of Indic texts

S.P. Mudur*, Niranjan Nayak, Shrinath Shanbhag, R.K. Joshi

Graphics and CAD Division, National Centre for Software Technology, Juhu, Mumbai 400 049, India

Abstract

There has been virtually no software localization into any of the major languages of India. One important reason for this is the fact that enabling Indic scripts at the base software level involves sophisticated computational process that are far from the traditional font level substitutions that suffice for a number of other world languages. Indian languages are basically phonetic in nature and fortunately, their writing systems are amongst the most logical and are thus highly amenable to being embedded in software. This paper is primarily concerned with the shaping process needed at the kernel level of the operating system so that software systems can include support for enabling Indic scripts. The shaping architecture and computational process described are based on over two decades of work in trying to build basic support for Indian languages in computer systems. We first present the basic phonetic nature of Indian scripts and the unique characteristic nature of writing in Indian languages. Next the computational process of shaping and a general architecture for its implementation are described. Finally the specific implementation for Unicode encoded texts displayed using TrueType Open fonts is briefly presented. © 1999 Elsevier Science Ltd. All rights reserved.

Keywords: Indic texts; Complex scripts; Uniscribe; TrueType Open fonts

1. Introduction

Of India's nearly one thousand million people, over nine hundred million are currently excluded from actively participating in this so called information age – internet and the web by the near total absence of software that can deal with information in the language which the majority of the Indians speak. In the recent past internationalization has been of concern in all major software development efforts in the world.[1] Thus, a number of software systems of wide use have been localized to enable their use by many non-English speaking persons in Europe, Japan, China and Arabic countries. However, there has been virtually no 'software localization' into any of the major languages of India. The reasons for this could be many – cultural, market size, etc. However, technologically too localization of software for use in India poses a number of challenging problems. The first and most important being the fact that Indian language encoding and display requires computational processes that are very different from those required by other languages of the world. Thus enabling Indian languages in software does not happen by simple font level substitution. Far more sophisticated processing of the encoded text is needed for determining the final appearance on the display or on hard copy (Fig. 1).

This paper is primarily concerned with the shaping process needed in software systems to include support for enabling Indic scripts. There are a number of software applications that support Indian scripts in the Indian market today. However, all of these are specific applications and hence even standard desktop metaphor operations like cut and paste are not available. It is important that Indic script shaping is enabled at the kernel level to be able to truly derive the benefits of the tremendous developments taking place through software. In this paper we specifically address the development of software with this goal. The work described in this paper is based on over two decades of research and development in trying to build in Indian language support in computer systems [1–9].

In the next section the basic phonetic nature of Indian scripts and the unique characteristic nature of writing in Indian languages are discussed. Next the system

*Corresponding author.

[1]All trademarks appearing in this paper are properties of their respective companies.

0097-8493/99/$ – see front matter
PII: S0097-8493(98)00113-7

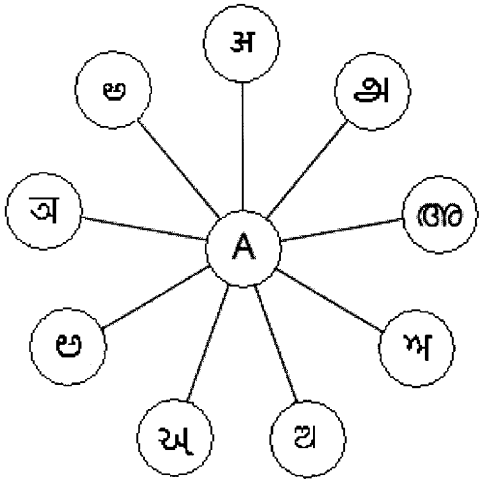

Fig. 1. Letters of major Indian scripts corresponding to letter 'A' from Roman.

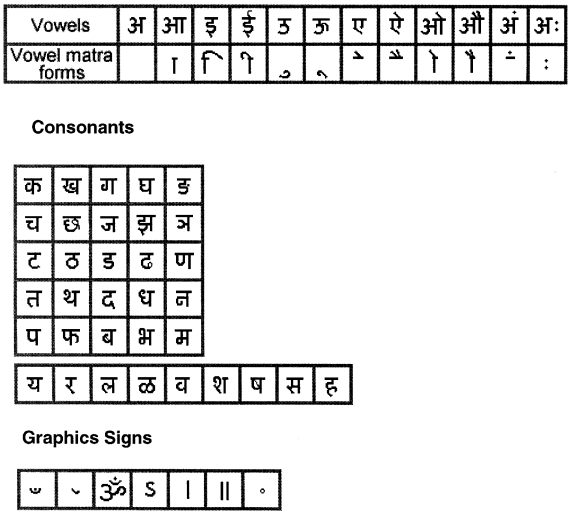

Fig. 2. Vowels, consonants and graphic signs in Devanagari.

architecture framework within which the shaping engine operates is briefly discussed. Then the computational process of shaping and a general architecture for its implementation are presented in detail. Finally the specific implementation for Unicode encoded texts displayed using TrueType Open fonts are briefly discussed. The paper includes a large number of illustrations to enable people not knowledgeable in Indic scripts to understand the basic requirements and shaping solutions.

1.1. The phonetic basis of Indian languages

India is a multilingual country with 18 recognized official languages and over 6000 dialects. Of these 18 official languages, some are of Perso-Arabic origin with script and writing rules similar to Arabic. Most of the others have their orthography derived from the ancient Brahmi script. As a result the orthographic rules for writing text in these scripts are more or less the same, even if their scripts are totally distinct. This is a great boon when it comes to computerization. One software engine with different parameterizations for the different scripts/languages will be able to handle this group of languages. The rest of this paper is about this software engine, the shaping engine for Indic scripts.

Panini's phonetic classification of the Indian alphabets into vowels (V) and consonants (C), (known as Swaras and Vyanjanas in Sanskrit, Hindi, Marathi and Konkani, Atchulu and Hallulu in Telugu, uir and mey in Tamil, etc.), serves as a common base for all Indian languages of non-Perso-Arabic origin. It also provides us with a unique encoding for any word in the language. There are differences in their written forms, as different letter shapes and different shaping rules get used. In addition to the vowels and consonants, there are also a few graphical signs used for denoting nasal consonants, for nasalization of vowels, etc. (denoted hence forth by G).

The effective unit of the writing system for all the Indian languages is the orthographic syllable, (known as Akshara in Hindi and Varna in Sanskrit), consisting either of a lone vowel, optionally, followed by a graphical sign with the structure (V) (G) or a consonantal syllable consisting of a consonant and a vowel (CV) core and, optionally, one or more preceding consonants, and an optionally following graphical sign (Fig. 2). The canonical structure for a syllable is thus of the form (C (C)) (C (V)) (G). Two consonants in a syllable is a common phenomenon. In some syllables the number of consonants can go even up to five.

The methodology of combining these two basic groups (C and V) to form various syllables is in itself a unique and scientific approach, common to all the Indian scripts. The various combinations of one (or more) consonants with each of the vowels turns out to be a perfect matrix called Barakhadi in Hindi, Marathi, Konkani and Gujarati, Matralu in Telugu, Varnamala in Sanskrit, Bannan in Assamese, Bangla, Oriya and so on. The Barakhadi table is used to teach a child writing in an Indian script in the same way as multiplication tables are used to teach arithmetic. In fact all primary language teaching books will include such a table in some form or the other [10].

The vowels include short and long versions of the same pronunciation. There are 12 basic vowels which are common to all languages, and hence the term Barakhadi (Fig. 3). Each of the languages may have vowel sounds that are slight variations and additions to these basic set of 12 vowels.

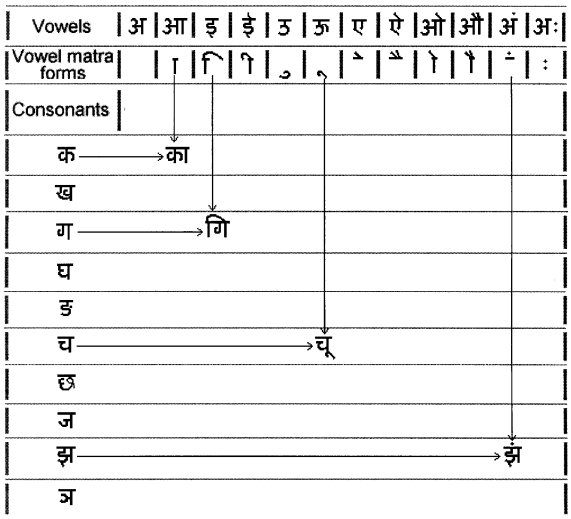

Fig. 3. Forming the Barakhadi.

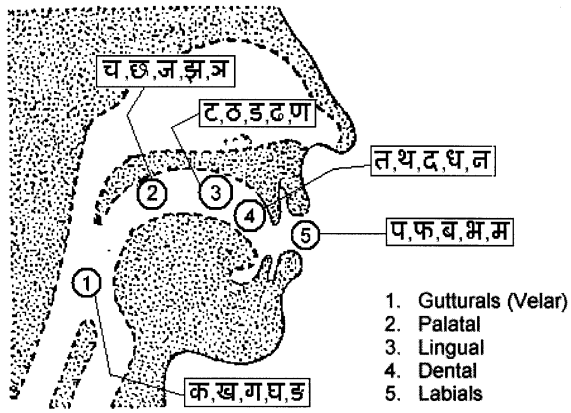

Fig. 4. Five vargas and their source.

The consonants: The basic set of consonants common to most Indian languages have been categorized according to their source center's in the human body which help us in producing these sounds, *guttural* (throat), *palatal* (palate), *dental* (teeth), etc. There are 5 *vargas* (*groups*) and a set of non-varga consonants. Each Varga contains 5 consonants, the last of which is a nasal one. The first four consonants of each Varga, constitute the *primary* and *secondary* pair. The second consonant of each pair is the aspirated counterpart (has an additional 'h' sound) of the first one (Fig. 4).

Apart from this basic set, like in the case of vowels, there are a few more consonants in use in some of the other languages, essentially variations in pronunciation of some of the consonants.

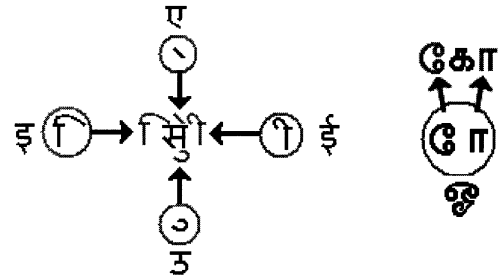

Fig. 5. Vowel matras are attached from all sides to a given consonant.

Anuswar: Anuswar indicates a nasal consonant sound. When an Anuswar precedes a consonant belonging to any of the 5 Vargas, then it represents the nasal consonant of that Varga. Before a non-Varga consonant however the anuswar represents a different nasal sound. Though pronunciation of the nasal consonant sound occurs first it has become standard practice when writing or typing to add it to the end of the previous syllable.

In Fig. 4, Varga 1 are the Gutturals. Varga 2 are the Palatals. Varga 3 are the Linguals. Varga 4 are the Dentals. Varga 5 are the Labials

2.1. Non-Varga

Chandrabindu: This sign denotes nasalization of the preceding vowel (can be implicit vowel within a consonant).

Visarg: This sign appears after a vowel sound, and represents a sound similar to 'h'.

Ayudha eluthu: This sign of Tamil appears after a vowel sound, and represents a sound similar to 'kh'.

2.2. Characteristic features of the writing system in Indian languages

Vowels and Vowel Matras: A characteristic feature of the writing system is that the rendering of a vowel is based on its relative position with respect to consonants (Fig. 5).

There are separate shapes for the stand-alone vowel (when it appears at the beginning of a word or when it immediately follows another vowel).

The medial vowel, as it is called, when it immediately follows a consonant, usually takes the form of a sign (known as Matra in Devanagari) to be attached to the basic consonant shape. It is the attachment of the matras to the consonant shapes that makes rendering Indian scripts more complex. While the encoding order is CV, the matra sign corresponding to V can attach itself to the left, right, top, bottom or even surround the consonant

shape. The precise shape of a matra sign will depend on the consonant shape that it gets attached to. It is not uncommon for any single vowel to have many different matra glyphs.

It is noteworthy that the rules pertaining to the sequential arrangement of vowels and consonants are common and consistent. A vowel preceded by a vowel (VV), or a vowel followed by a consonant (VC) but not preceded by a consonant would always result in a stand alone vowel whereas a consonant followed by a vowel would always result in a consonant combined with a vowel matra.

2.3. Single consonants and half-consonants

The basic consonant shape in the Indian script always has the implicit vowel (the vowel sound as of the last 'a' in the word 'consonant'), and hence there is no explicit Matra form for the [a] vowel. However, there are equivalent Matras for all the other vowels, which get attached to the basic consonant shape whenever the corresponding vowel follows (immediately) the consonant (Fig. 5).

The written form of a basic consonant without the implicit [a] vowel either has an explicit shape or it has the graphical sign, known as the virama (or halant in Hindi) attached to its basic consonant shape. This is then referred to as the 'Halant form' of the consonant.

Often when the basic consonant without the implicit [a] vowel is immediately followed by one or more consonants, then this consonant may take what is called as the half-form in the ligaturized version of the combination of consonants. The half-form of a consonant closely resembles the basic consonant, usually with some part (say stem) removed (Fig. 6).

2.4. Conjuncts

In Indian scripts when two or more consonants cluster together as part of a syllable, then the consonant cluster can take a very different shape from that of the constituent consonants. These are known as conjuncts (Samyuktakshara in Sanskrit, Jodakshar in Marathi, Dhwitwaksharallu in Telugu, Juktakshara in Bengali, etc.) This orthographic unit of pure consonant + pure consonant + ⋯ + vowel (C…CV) is common to all these Indian scripts. Therefore, the basic unit of the pure consonant (basic consonant without the implicit [a] vowel), and its combination with vowels is the common base for all Indian scripts (Fig. 7).

In a number of languages like Sanskrit, Bengali, Oriya, etc. the number of conjunct shapes can be very much

Consonant	Full Form	Half Form
क	क	क्
ट	ट	ट्
ಕ	ಕ	ಕ್
ഗ	ഗ	ഗ്
ઞ	ઞ	ઞ્

Fig. 6. Consonant, their full forms and half forms.

BICONJUNCTS

KA AT FIRST PLACE

क्क क्ख क्च क्छ क्ट
क्ठ क्ण क्त न्क क्प
क्फ क़्फ़ क्ब क्म क्य
क्र क्ल क्ष क्ष क्स
क्श

KA AT SECOND PLACE

ङ्क ट्क त्क र्क ल्क
श्क ष्क स्क

TRICONJUNCTS

KA AT FIRST PLACE

क्क्ण क्क्य क्ख्य क्थ्न क्श्य
क्ष्ण or क्ष्ण त्क्र क्ष्म or क्ष्म

KA AT SECOND PLACE

ङ्क्त ङ्क्य ल्क्य
त्क्र or त्क्र

QUADRI CONJUNCTS

ङ्क्त्र or ङ्क्त्र
ङ्क्त्य

PENTA CONJUNCTS

ङ्क्त्य or ङ्क्त्र्य or ङ्क्त्य्य

Fig. 7. Various combinations of conjuncts.

larger than the number of consonants governed by the system. Conjunct shapes are complex and may be composed out of simpler shapes in a number of ways:

- A distinct conjunct: a completely different shape for the consonant cluster, in which the individual consonants are not distinguishable. Graphically these conjuncts are like basic single consonants and they have their half-forms as well.
- A horizontal conjunct: all preceding consonants in the syllable cluster take the half-consonant shape while the last consonant takes the basic consonant shape. The vowel matra then gets attached to this cluster.
- A vertical conjunct: to a basic consonant or to a distinct conjunct shape, special signs depicting the other consonants get attached vertically. The matra then gets attached to the vertically composed conjunct.

2.5. Link language diacritic marks

In-order to graphically denote vowels and consonants of another language special diacritic marks are often used in a specific script. In particular Hindi (written in Devanagari script) being the national language, addresses this. A number of diacritic marks have been devised for extending Devanagari as shown in Figs. 8 and 9.

- the vowel sounds, short E and short O of the South Indian scripts,
- the vowel sounds in the English words cat, pat, rat and cot, pot, rot, etc.,
- the flat vowel sounds in Kashmiri,
- the Nukta is used for denoting consonant sounds close to the original Sanskrit pronunciation (takes its origin from Arabic/Persian scripts),

Fig. 8. Extended vowels for link language diacritics.

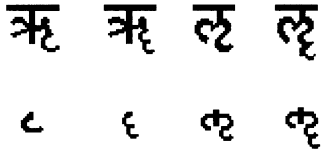

Fig. 9. Extended vowels for ancient Sanskrit text.

- Some times the same consonant has two different variations in two different languages. For this a second diacritical mark called as the implosive is used. The implosive, however, has not been included as a character in Unicode.
- Vedic signs – Sanskrit marks specific to Vedic Sanskrit text.

2.6. Numerals

All the Indian scripts have their own distinct shapes for the numbers. In all official communication in the national language Hindi one must use the international shapes (Roman script numbers). However, Indian script numbers are in regular use in all other printed material and also in most of the states.

2.7. Punctuation and special Indian script symbols

All punctuation marks used in Indian scripts are borrowed from English. There are a few special symbols and punctuation marks particular to Indian scripts. Here we include only those symbols, which are common to most of the languages, and are also included in Unicode Standard Version 2.0 [11]. There may be special symbols specific to a language.

Viram: *is* used instead of a full stop in the Northern scripts.

Avagraha: *is* primarily used in Sanskrit texts. It creates an extra stress lengthening the preceding vowel.

Aum: *is* a Hindu religious symbol.

Abbreviation: *is* used like the full stop in English after Mr., Dr. etc.

Halant: *is* a mark that is primarily used to graphically indicate vowel-less ending of a word.

Udatta and anudatta: These are two marks, one below and the other above a character, essentially vowel elongators.

The barakhadi for the consonant KA is shown in Fig. 10.

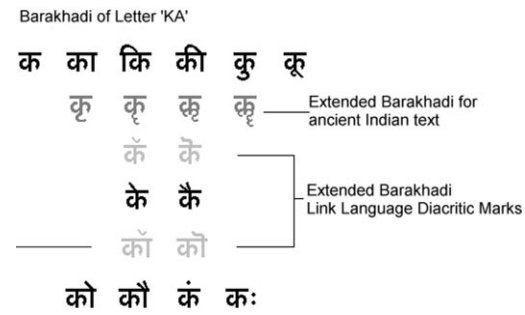

Fig. 10. Barakhadi of letter KA.

3. Requirements for Indian language enabling in software

3.1. Character encoding

Indian language text input differs from that in English. The most significant difference of these is that in English, each keystroke maps directly to a letter. Each letter has a unique code. A '*Syllable*' – the Indian language equivalent unit of writing letter, however is composed of one or more characters entered through the keyboards or any other input mechanism. There are far too many syllables to be encoded separately.

The syllable is composed of vowels, consonants, modifiers and other special graphics signs. These are encoded, just as roman alphabets are. The user types in a sequence of vowels, consonants, modifiers and the graphic signs. The machine then composes syllables at run time based on language dependent rules. Every syllable is thus represented in the machine as a unique sequence of vowels, consonants and modifiers. In a text sequence, these characters are stored in logical (phonetic) order.

3.2. Rendering Indic characters

Indic characters can combine or change shape depending on their context. A character's appearance is affected by its ordering with respect to other characters, the font used to render the character, and the application or system environment. These variables can cause the appearance of Devanagari characters to be different from their nominal glyphs (used in the code charts). Additionally, characters cause a change in the order of the displayed glyphs. This reordering is not commonly seen in non-Indic scripts and occurs independent of any bi-directional character reordering that might be required.

Each syllable has a unique visual representation. However, there are too many syllables to design individual glyphs for each. So a font normally contains certain component glyphs from which a syllable is composed at run time. The onscreen representation of a syllable is then a composition of glyphs from the Indian language font.

There is no direct mapping of glyph codes to the consonant, vowel or modifier codes. However, for every syllable (a sequence of consonant, vowels and modifiers) there is a corresponding sequence of glyphs. This constitutes a many-to-many mapping from keystrokes to glyphs as opposed to the simplistic one-to-one mapping in roman scripts.

3.3. Caret positioning

In a roman editor, carets are positioned in between alphabets. In an Indian language editor, carets are positioned in between syllables. Syllables are a sequence of character codes in memory and a sequence of glyphs on screen. Moving over a syllable means moving over appropriate number of character codes in memory and over combined advance width of glyphs on the screen.

As the user types in a sequence of vowels, consonants and modifiers, syllables start forming. These syllables are formed progressively. For example a sequence of one consonant and one vowel may constitute a syllable. Adding a modifier to this combination changes it to a different syllable. The states keep changing. A new keystroke that does not form part of the current syllable, marks the beginning of a new syllable.

As the syllables keep changing, their representations on the screen change. The number of bytes in a syllable is not linearly related to the number of glyphs required to represent it on screen. This is radically different from the simple roman script model. What it means is that even if a character is added to the edit buffer the line extent may decrease, or even if a character is deleted from the edit buffer the line extent may increase. The system/application has to be aware of this.

Since the syllable's onscreen representation is a composition of different glyphs, there has to be support for such composition at runtime. Ideally this support must come from the operating system. The complete syllable, in its current state, is required for this composition. This means that the system/application has to maintain the syllable break up information amongst the characters in its buffer.

3.4. Backspace and delete

Backspacing removes immediately preceding consonant, vowel or modifier from the syllable, like a stack's pop operation. When there are no more items to be removed, the syllable is deleted. The syllable and its on-screen representation changes after every backspace. The delete operation however removes the entire syllable.

Thus editing in Indic script involves maintaining a clear distinction between character codes and glyphs. Each set of character codes forming a syllable are then shaped together to get the resulting glyphs. Editing involves maintaining separate counts of the number of characters and number of glyphs for each syllable.

4. The shaping architecture

Shaping is the process of taking an input set of characters to their final glyph presentation forms. The set of rules governing the shaping and positioning of glyphs are specified and cataloged within the Unicode standard. Shaping indic text involves understanding of language/script specific rules, and basic system/font support so as to encode these rules.

The shaping architecture described below is a generic architecture to be used by applications supporting Indic text layout. In a later section we will briefly discuss specifically how this is implemented for Unicode encoded text and TrueType Open fonts.

From the above discussion it should be clear that Indic scripts require special processing to display and edit because the characters are not laid out in a simple linear progression, as most is the case with most other world languages. This special processing falls into several general classes. In the architecture described below, the font encodes not only the geometric shapes of glyphs but also the shaping rules applicable. The shaping engine works concurrently with the font to transform the given sequence of coded characters into the right displayable/printable sequence of font glyphs. Using this architecture, it becomes possible not to freeze the glyph layout or even the actual glyphs or their shapes. For a single language and script two fonts could have differences in their glyphs, with correspondingly applicable but different shaping rules.

4.1. Indic shaping engine components

Syllablic cluster separator: Indic text shaping operations are carried out at the syllabic level. All input text is first grouped into syllables as defined by the logic in the previous section.

Shaping context extraction: The choice of which glyph sequence to display depends on the surrounding characters. The context and the corresponding rules to be applied are encoded within the font. The shaping engine then has to match the contexts from those within the fonts and apply matching rules in the order specified.

Character/glyph reordering: Certain characters/glyphs within a syllable are not necessarily in the same visual order as the input character code sequence. One option is to embed reordering rules in the font itself. However, reordering is essentially font independent and a language dependent issue. Reordering apriori before font lookups using language semantics/rules is another option. Separating this script/language dependent part out of the font removes un-necessary duplication of common rules and also helps reduce complexity of rules encoded within the font. Some characters get rearranged from logical (keystroke input) order to visual order.

Glyph compositing: The shaping engine has to convert the syllabic cluster of character codes, reorder, determine contexts, apply context specific rules and finally composite the sequence of font glyphs that represent the correct visual rendering for that syllabic cluster in that language using the font.

Glyph positioning: Application of font encoded rules can result in glyphs resulting from multiple characters being stacked or combined into one cluster. The shaping engine handles the correct relative positioning of the individual glyphs in the composited cluster of displayable glyphs.

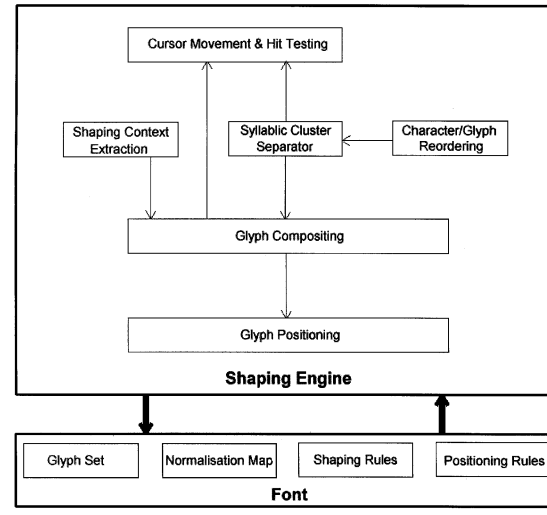

Fig. 11. Architecture of Indic shaping engine.

Cursor movement and hit testing: The mapping between screen position and a character index for, say, selection of text or cursor display requires knowledge of the layout algorithms especially since there is a many to many mapping possible in the Indic language context. The shaping engine has also to include suitable APIs for this purpose.

An architecture of the shaping engine and its associated components is shown in Fig. 11.

4.2. The Indic script shaping process

The shaping process involves the following steps. In this we have used the same character names as are used in the Unicode Version 2 standard.

1. Obtain syllablic cluster of character codes from the given input sequence.
2. Handle character reordering if any.

   The above steps occur in the character domain.
3. Apply individual character code mapping to get it normalized to the specific font.
4. Extract contexts. The list of contexts needed in our architecture is given below. More detailed description of these contexts and the rules to be applied in these contexts are presented later.

   (REPH): When the 'R' pure consonant is the first in a syllabic cluster.

   (AKHAND): When the syllabic cluster includes a substring which has to be treated like a separate letter.

   (NUKTA): When a consonant is followed by a 'Nukta' character code to denote that it's sound is being modified.

The three above are distinct contexts that an Indic script shaping engine would have to handle. Not all scripts would require all these contexts. For example while the reph context is used in many North Indian languages it is not present in South Indian languages like Tamil.

5. Reorder code sequence if the syllabic code sequence requires it.
6. Apply rules for glyph compositing depending on context in the following order.

   The syllable ending with a pure constant (without an ending vowel) is not very common. For the following steps if a syllable ends with a pure consonant, the glyph compositing process will work as if the last consonant is a consonant with the implicit 'a' vowel and then attach the 'halant' glyph last. This form is referred to as the halant form of the consonant.

   Apply REPH substitution rule.

   Apply AKHAND substitution rule.

   Apply NUKTA CONSONANT substitution rule.
7. Apply consonant HALF form rule to the whole syllable. This provides a unique representation for the pure form – it substitutes Consonant + Halant with its pure form as an intermediate step. The pure form is then suitably substituted by either the half-form or the halant form or ligaturized depending on the subsequently applicable rules.
8. The following compositing rules are applied independent of context.

   Apply VATTU substitution rule. The vattu is a consonant glyph shape for combination of consonants. The vattu shape is determined by the preceding pure consonant. This handles all vattu glyph substitution.

   The font designer can build in suitable rules in the font, and design and place as many vattu ligatures/marks or none at all as required by the font. In case there is no vattu rule with some consonant, then the half form is retained.

   Apply CONSONANT CONJUNCT substitution rule.

   Apply HALANT substitution rule. This replaces the Consonant + halant sequence with a single ligature.

   Apply MATRA substitution rule. This handles substitution of appropriate matra glyphs, ligaturization of matras, rephs and vowel modifier glyphs. (Matra here is taken to mean all glyphs forms that do not depict a Stand alone vowel or consonant glyphs.)

   Apply REPHCONJUNCT substitution rule. This handles ligaturization of reph with the preceding base glyph.

   Apply VOWELMODIFIER substitution rule. This handles ligaturization of matras and reph with vowel modifier. It also handles context based substitution of vowel modifiers.

We now have completed the compositing of the syllable in the form of a cluster of glyphs.

9. Glyph positioning

   First position the various combining marks at the end of a syllable.

   Specify the correct distances between ligatures if they differ from the nominal distances.

   The following rules are applied during the positioning operation:

   Apply the table *TopMarks* to the current sequence of glyphs:

   This table positions the top marks Reph, Chandrabindu, Anusvara, Udatta, Grave accent, acute accent and all top matras with respect to different types of glyphs of type.

   Apply the table BottomMarks to the current sequence of glyphs:

   This table positions the bottom marks Halant, Anudatta and other matras that appear at the bottom with respect to glyphs of different types.

The standard order for encoding all basic Indic shaping rules is listed below. Each rule represents a single type of layout/mapping operation. Typeface-specific script rules for Indic, and all complex scripts, must ultimately be represented in a font in a font specified standard manner.

Table 1 shows the standard order for encoding and applying Indic shaping rules to obtain consistent and standard visual forms. The table also indicates how each rule relates to the rendering rules for Indic scripts laid out in the Unicode Standard.

Table 1

Shaping rules listed in precedence order	Corresponding Unicode rules	Typical form of compositing operation
Reph	R2	Glyph substitution
Akhand	R3/R11/R12	Glyph substitution
Nukta	R9	Glyph substitution
Pure form of consonant	R1	Glyph substitution
Vattu	R6/R7/R8/R13	Glyph substitution
Consonant Conjuncts	R4/R11/R12	Glyph substitution
Top and Bottom Marks	R10/R14	Glyph substitution/Glyph Positioning

4.2.1. Indic script shaping contexts and rules

In this section we describe the contexts and rules in some detail.

4.2.2. Reph context, shaping and reordering rule

If the pure consonant 'R' (Ra + Halant in Unicode characters) is the first consonant in a syllabic cluster, then the REPH context is 'on' and the reph rule has to be applied. This rule substitutes the combining-mark form of Reph for the half form of consonant Ra. In addition, the glyph that represents the combining-mark form of 'RA' is repositioned in the glyph string so that it is attached to the final base glyph of the consonant cluster (Fig. 12).

In the following illustration (Fig. 13), a longer cluster is formed, and the mark glyph form of the 'RA' syllable is repositioned at the last consonant of the consonant cluster.

4.2.3. Akhand context and shaping rule

Certain combinations of consonants, though they are conjuncts, take distinctly different visual forms, to the extent that they have to be treated just like the standard consonant characters in the script. They must have their half forms, their pure forms, their full forms, their forms with Reph, with matras, etc. The shaping engine will check if the syllabic cluster contains a substring that has to take an Akhand form and turn this context 'on'. The Akhand rule provides the akhand ligature form in place of the proper consonant combination (or 'conjunct') (Fig. 14). Most other consonant conjuncts are represented by a ligature shape, which is composited using a composition rule,in which the preceding consonant assumes a half-form.

As a user types each character, the text-processing application will reshape the glyph or glyph cluster which is displayed, as illustrated above.

4.2.4. Nukta context and shaping rule

The Nukta is a special character code which alters the way a preceding consonant is pronounced. All of the Nukta forms (combinations of consonants and the Nukta character) have been defined as separate glyphs in Unicode, each with their own code points.

Using this context and rule, a glyph representing the proper Nukta form is substituted for a consonant + Nukta combination.

As a user types each character, the text-processing application will reshape the glyph or glyph cluster which is displayed, as illustrated in Fig. 15.

4.2.5. Consonant half-form shaping rule

This shaping rule is the most widely used feature in the Indic scripts, and substitutes the half-form of a consonant when a consonant or akhand or nukta consonant is followed by the Halant character. Half forms of Nukta consonants and Akhand consonants also exist.

In the case that a language does not have half-forms of consonants, this rule will substitute a ligature representing the consonant and the Halant that follows it (consonant + Halant).

Consonant conjuncts (ligatures) are always formed from the right. 'Loose' left-hand glyphs are replaced by their respective half-forms.

As a user types each character, the text-processing application will reshape the glyph or glyph cluster which is displayed, as illustrated in Fig. 16.

4.2.6. Vattu shaping rule

Some Indian languages substitute a consonant within a syllabic cluster with its Vattu (diacritic suffix like mark) form depending on the position of the consonant in the cluster or syllable. This rule enables the substitution of these special glyphs with a ligature glyph.

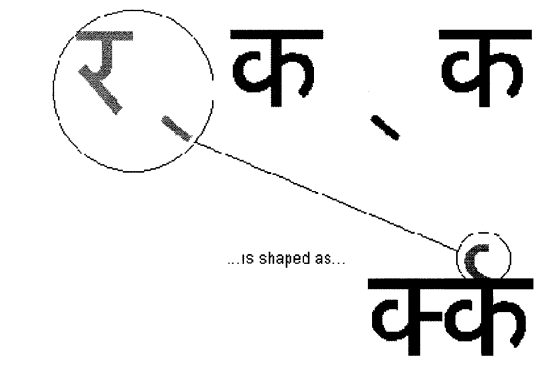

Fig. 12. Reph glyph form substituted and positioned.

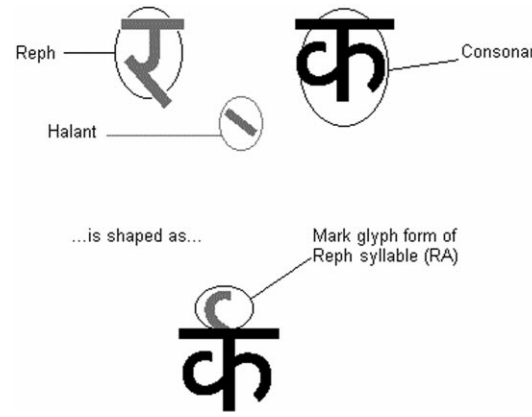

Fig. 13. Mark glyph form substituted and repositioned.

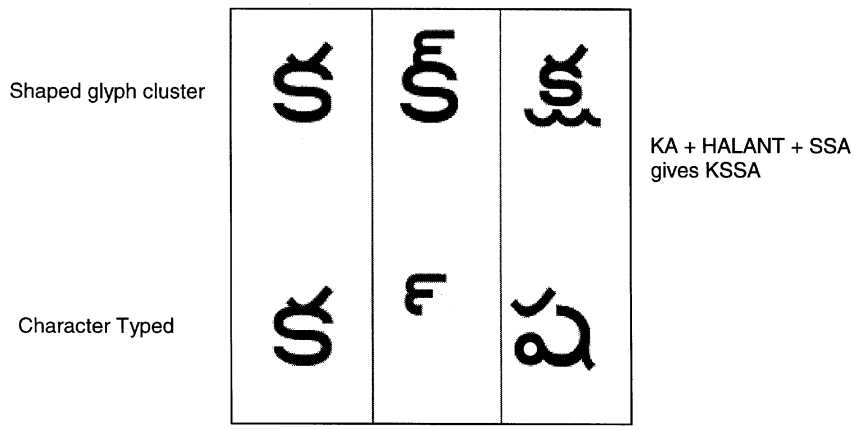

Fig. 14. Akhand form substitution.

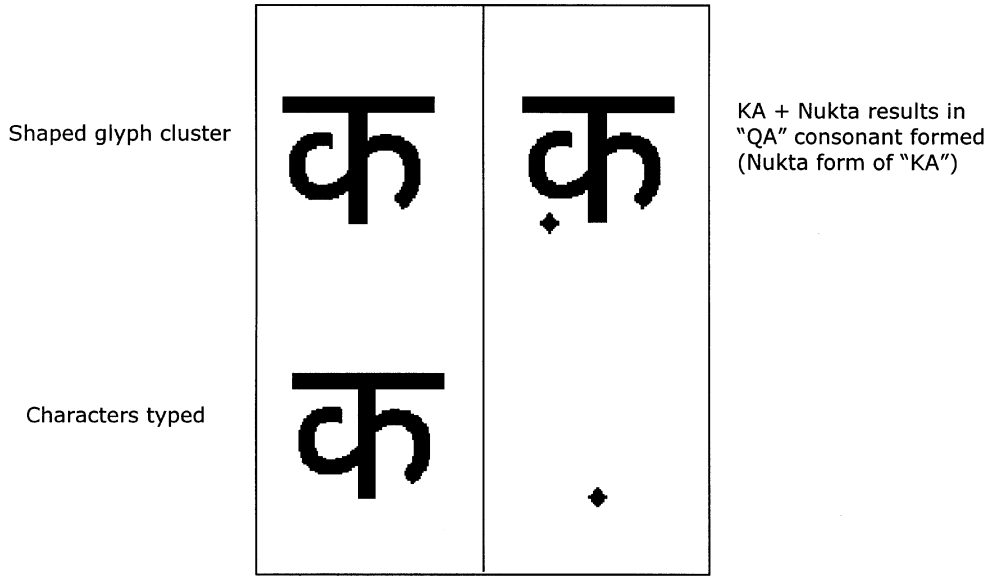

Fig. 15. Application of the nukta shaping rule.

As a user types each character, the text-processing application will reshape the glyph or glyph cluster which is displayed, as illustrated in Fig. 17.

4.2.7. Consonant conjunct shaping rule

Most often a complex conjunct is visually shaped as a sequence of half-form consonants followed by a consonant, optionally with its associated vowel character. In such a case, this rule substitutes the ligature glyph representing the combination.

As a user types each character, the text-processing application will reshape the glyph or glyph cluster which is displayed, as illustrated in Fig. 18.

4.2.8. Consonant Halant form shaping rule

This rule provides a ligature glyph representing the Halant form of a consonant, whenever a consonant is followed by a Halant. This visual form of the consonant remains in the final appearance if this is the end of the

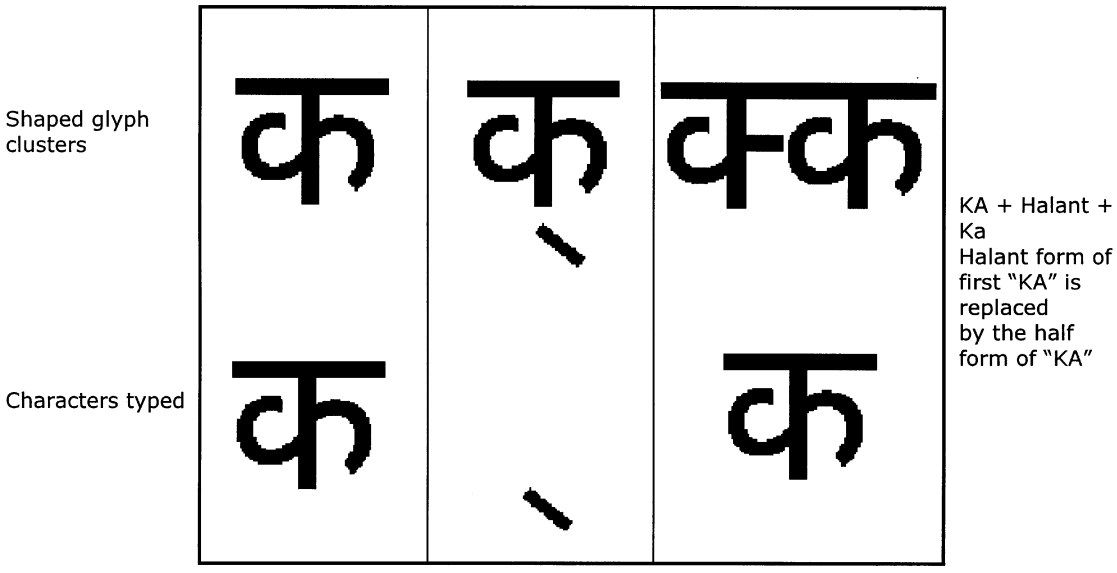

Fig. 16. Half-form of consonant rule applied.

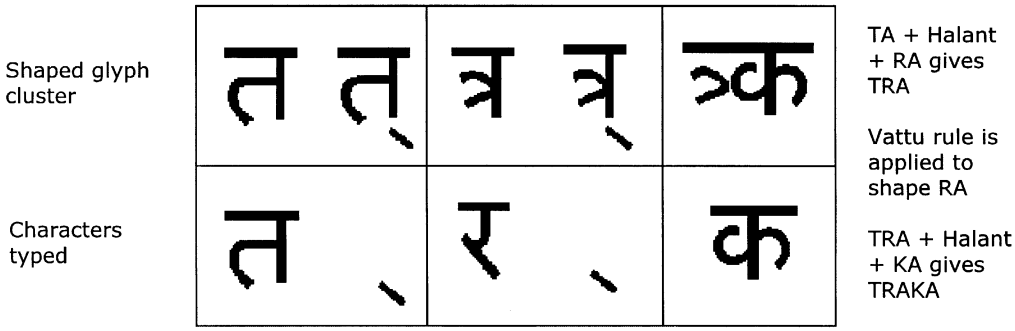

Fig. 17. Vattu shaping rule applied.

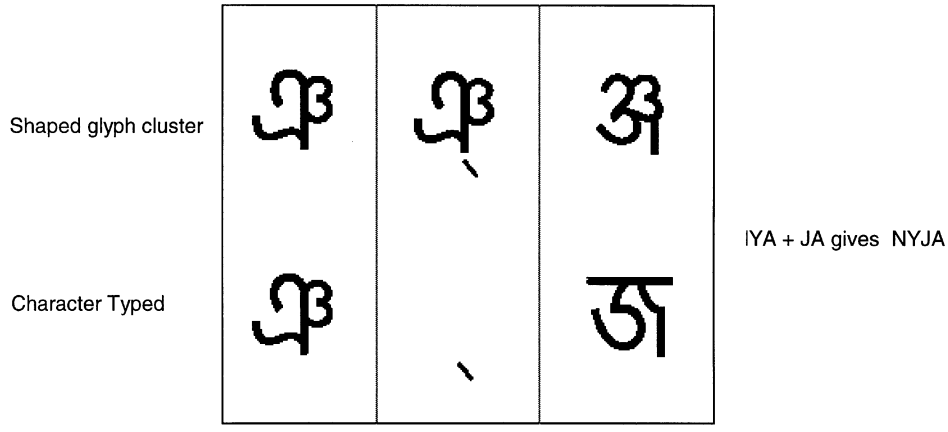

Fig. 18. Consonant conjunct shaping rule applied.

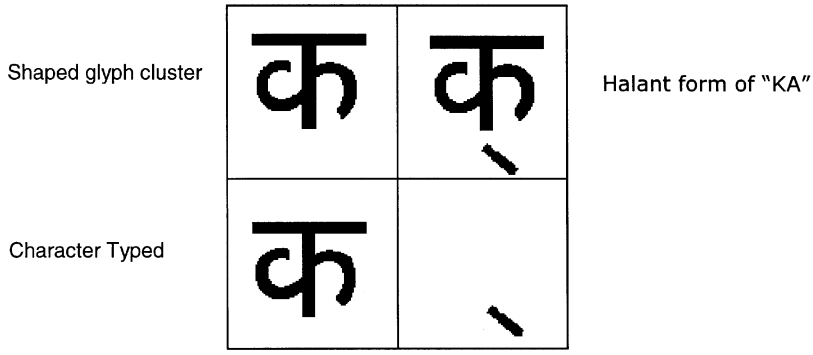

Fig. 19. Consonant halant shaping rule applied.

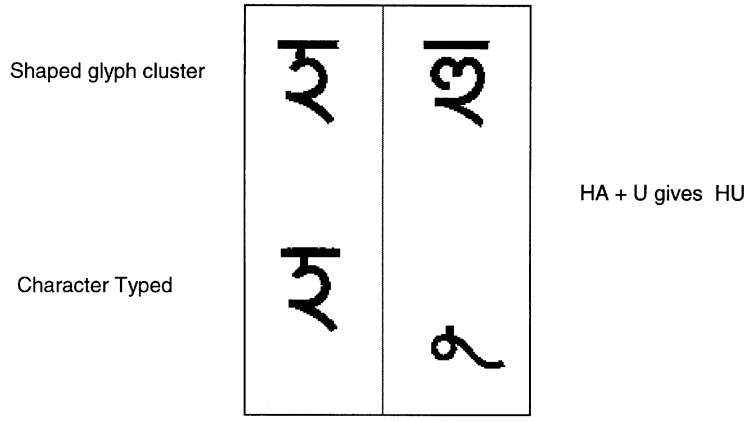

Fig. 20. Matra ligaturization rule applied.

syllabic cluster. Otherwise, it depends on what character codes follow in this syllabic cluster.

As a user types each character, the text-processing application will reshape the glyph or glyph cluster which is displayed, as illustrated in Fig. 19.

4.2.9. Matra substitution and matra ligaturization shaping rule

The 'Matra substitution' rule contextually chooses the correct shape of the matra character, depending on the base consonant or the ligature that precedes the matra.

As a user types each character, the text-processing application will reshape the glyph or glyph cluster which is displayed, as illustrated in Fig. 20.

This feature is also used for substituting a ligature glyph for the combination of a base consonant followed by a matra (Fig. 21).

4.2.10. Reph ligature shaping rules

This rule provides a ligature form of a matra combined with reph when a matra glyph is followed by the reph glyph.

As a user types each character, the text-processing application will reshape the glyph or glyph cluster which is displayed, as illustrated in Fig. 22.

4.2.11. Vowel modifier shaping rules

This rule chooses the correct shape of the vowel modifier (combining mark). It can be used to choose the correct form of the vowel modifier depending on the preceding ligature. This rule is also used to substitute matra + vowel modifier, and reph + vowel modifier with their ligature forms if required.

As a user types each character, the text-processing application will reshape the glyph or glyph cluster which is displayed, as illustrated in Fig. 23.

4.2.12. Positioning rules for top and bottom marks

The 'Top Marks' rule positions mark glyphs which appear at the top of a base glyph or ligature glyph. This rule is applied to glyphs such as dependent vowel signs (matras), the reph form of the consonant 'Ra,' combining marks (vowel modifiers such as anusvara), and top accents.

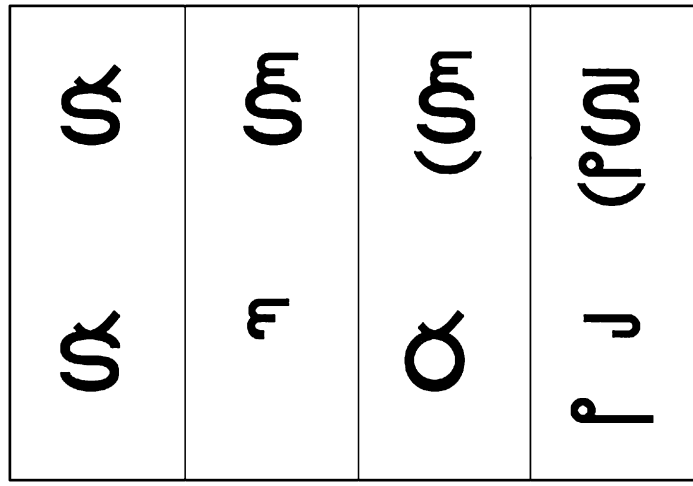

Fig. 21. Matra ligaturization rule applied.

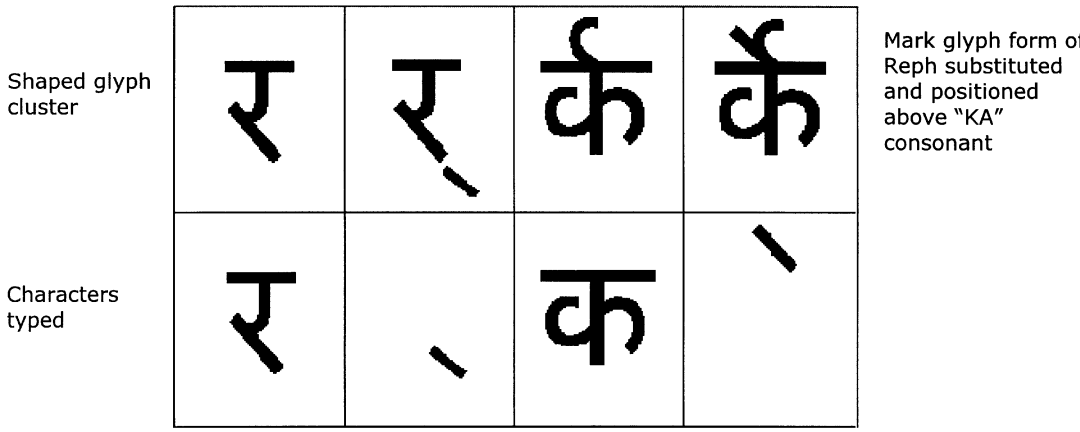

Fig. 22. Reph ligature rule applied.

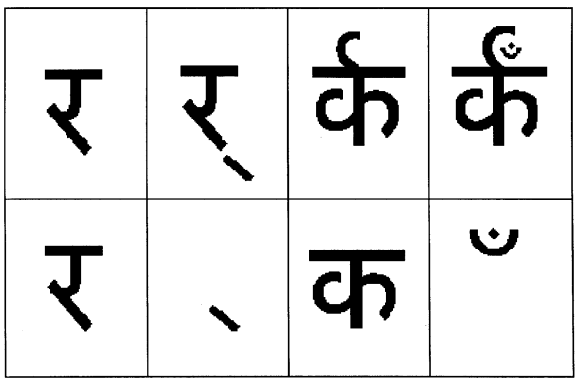

Fig. 23. Vowel modifirs substitution/ligaturization rule applied.

This is a glyph positioning feature, and must be encoded in the POSITIONING RULES table of the font. Each glyph that is to be used as a top mark must be identified as such in a suitable table of the font.

Top marks positoning is shown in Fig. 24.

The 'Bottom Marks' feature positions mark glyphs that appear at the bottom of a base glyph or a ligature glyph, such as dependent vowel signs (matras).

This feature is a glyph positioning feature, and is encoded as a positioning rule in the font table. Use of the bottom marks feature is illustrated in Fig. 25.

4.2.13. Unicode encoding for Indic scripts

The Unicode Standard is a fixed-width, uniform encoding scheme for written characters and text. The

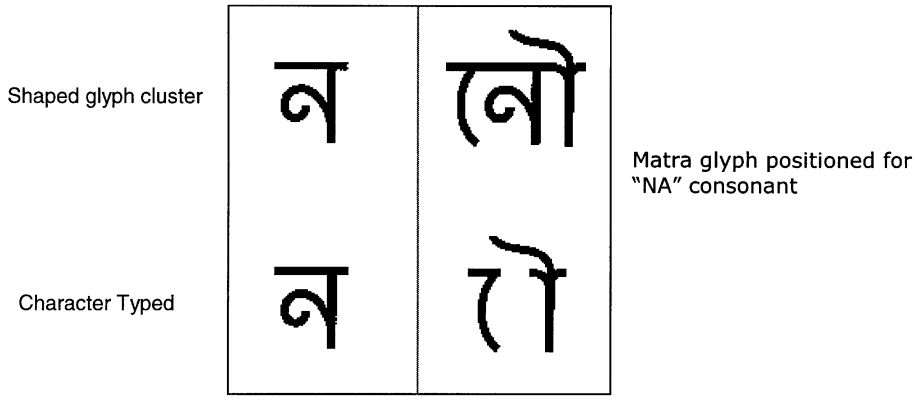

Fig. 24. Top marks rule applied.

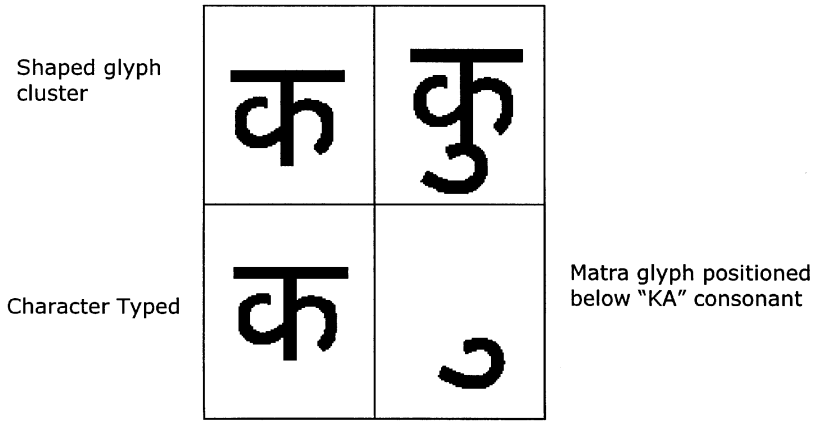

Fig. 25. Output of the bottom marks shaping rule.

repertoire of this international character code for information processing includes characters for the major scripts of the world, as well as technical symbols in common use. The Unicode Standard, Version 2.0 contains 38,885 characters from the world's scripts. These characters are more than sufficient not only for modern communications, but also for the classical forms of many languages. Languages that can be encoded include Russian, Arabic, Anglo-Saxon, Greek, Hebrew, Thai and Sanskrit.

The Unicode Standard draws a distinction between characters, which are the smallest components of written language that have semantic value, and glyphs, which represents the shapes that characters can have when they are rendered or displayed. There are various relationships between character and glyph: a single glyph may correspond to a single character, or to a number of characters, or multiple glyphs may result from a single character.

This distinction is important and applicable to Indic Script display and processing needs.

The Unicode code ranges for the various Indian scripts are as shown in Table 2.

Unicode does not encode the pure form of the consonant. Instead it encodes the form of the consonant with an implicit 'a' vowel. Similarly, the vowel is not uniquely encoded. Instead its standalone form and its matra form are separately encoded. This is similar, though not exact as the Indian common standard for encoding all India scripts [13].

4.3. The TrueType open font format

TrueType Open (TTO) is an extension to the TrueType font standard [12], the default in Microsoft's Windows operating systems. TrueType Open fonts contain additional information that extends the capabilities of

Table 2

Script	Code range
Devanagari	U + 0900–U + 097F
Bengali	U + 0980–U + 09FF
Gurumukhi	U + 0A00–U + 0A7F
Gujarati	U + 0A80–U + 0AFF
Oriya	U + 0B00–U + 0B7F
Tamil	U + 0B80–U + 0BFF
Telugu	U + 0C00–U + 0C7F
Kannada	U + 0C80–U + 0CFF
Malayalam	U + 0D00–U + 0D7F

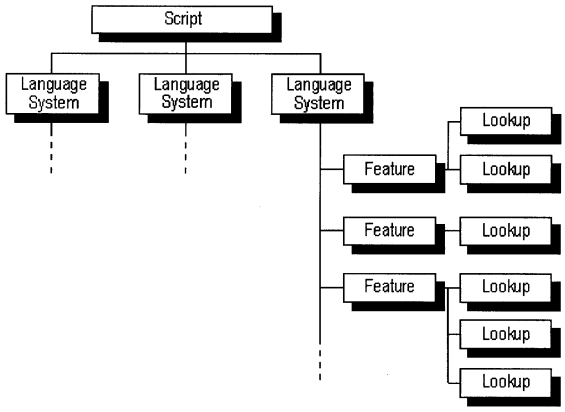

Fig. 26. Relationship between scripts, languages, features and lookups in open type.

the fonts to support high-quality international typography. The most important features of TTO are:

- can associate a single character with multiple glyphs, and – conversely – it can associate combinations of characters with a single glyph.
- includes two-dimensional information to support features for complex positioning and glyph attachment.
- contains explicit script and language information, so a text-processing application can adjust its behavior accordingly.
- has an open format that allows font developers to define their own typographical features.
- fonts support Unicode character to glyph mappings.

These features of TTO make it an excellent choice for Indic scripts

As shown in Fig. 26, layout features within TTO fonts are organized by scripts and languages allowing a single font to support multiple writing systems, even within the same script.

TTO fonts are also not dependent on a single character-encoding scheme, and in fact the format can support all the encoding schemes in common use today.

TrueType Open font format allows a font designer to provide glyphs for the many different forms that a character may take. It also lets the designer embed information required for proper context based glyph selection, within the font. This information is stored in TrueType Open fonts tables.

5. Developing TTO fonts for Indic scripts

To arrive at the final display form of a sequence of characters, the TrueType Open encoding scheme uses a multi-stage pipeline. The number of intermediate stages and the resulting final output depend on the input sequence of characters.

5.1. Tables

Data in TrueType Open font files is organized into tables. There are tables to specify font name, font metrics, glyph outlines and more tables to specify a lot of other details. The tables that deal with complex-script shaping are the GSUB table, the GPOS table and the GDEF table. Of these, GSUB is the most important table as it embeds the compositing rules and thus determines the selection of the final glyphs for the given input character sequence.

5.2. Lookups and LookupLists

The model that TrueType Open uses for shaping is one of pattern matching and substitution. The original character sequence is presented to the start of the shaping pipeline. This sequence undergoes repetitive pattern matching and substitution operations as it moves from one stage of the shaping pipeline to the other. Here it is important to note that every stage acts on the pattern presented to it by the previous stage of the pattern-matching pipeline.

All the patterns and their associated substitutions are encoded as lookups, one for each input pattern. (Note this is a simplified view, as TrueType Open also allows encoding of more than one pattern in one lookup using its Class features).

All the lookups are collected together in the form of the lookuplists. Each lookup is uniquely identified by its index into the lookuplist.

5.3. Scripts, languages and features

Scripts can generally be used to represent more than one language. Just like the Latin script can be used to represent English and German languages, Indian scripts

like Devanagari and Bengali are used for writing in many different languages, with minor differences. The display forms of a given character sequence depends on the language which it represents. Different sets of lookups from the lookuplists are normally required to shape characters for different languages.

Generally, a collection of lookups is used to describe a desired shaping effect across all the different possible input patterns. In TrueType Open, such logical grouping of lookups is called a feature. In other words, a feature is nothing but a collection of lookuplist indices, all of whose lookups describe a common language specific shaping action. Thus it is this notion of TTO feature that has to be used to embed Indic script shaping rules. Most compositing rules, can be represented as an ordered sequence of string substitutions. However, reordering and context based rule application cannot be represented in a finite number of string matching – string substitution rules. Hence some Indic script specific processing of the input character sequence by the shaping engine becomes mandatory when using TTO fonts.

The GSUB table is organized by scripts, which further contain the languages that they support. Each language defines its own set of applicable features. Each feature defines a collection of lookuplist indices. A set of all lookups applicable to a particular language can thus be obtained easily. (The features only group lookuplist indices. The lookups themselves are defined elsewhere in the lookuplist. In this way more than one language can share a lookup without unnecessary duplication of data.)

5.4. Shaping Indic scripts in TTO

The shaping engine is a program that has to accept the input sequence of character code, say in Unicode and then interpret the TrueType Open font for defining the final appearance of this input sequence. Thus, given an input sequence of characters, it must produce the correct sequence of output glyphs as defined by the font, for a particular language. The shaping engine can be a simple one and may choose to apply all the lookups applicable to a language, in the order in which they are defined in the lookuplist. (Order is defined by the lookuplist indices.) Alternatively, the shaping engine may detect the applicable set of features for a given input character sequence and only choose to apply lookups defined therein. Either way the lookups are always applied in the order defined in the lookuplist.

Application of a lookup causes its input to change. This new sequence of glyphs then forms the input for the next lookup in the shaping pipeline.

Developing True Type OpenType fonts that can be used to consistently produce the correct visual form for a Unicode sequence in an Indian language involves: defining required and optional features, identifying the ordering of these features, design of all glyphs (base glyphs, marks, ligatures, etc.), and encoding of the shaping and glyph positioning features in the font.

6. Defining and ordering features

Developing OpenType fonts that support complex scripts becomes straightforward, once required features have been defined and organized for a specific shaping engine. Given any Unicode sequence of characters codes encoding text in any particular language, there is a fixed order for executing features to obtain the correct visual form. This order is defined in conjunction with standards defined by the shaping engine.

Using these standards, any font developer can efficiently encode features that will produce a consistent basic form, across all fonts. The only differences in the design and layout of a particular script will be typeface specific.

7. Uniscribe implementation in Mircrosoft Windows NT 5.0

Microsoft has recently announced that all language versions of Windows NT 5.0 will be enabled for all supported languages, including European and Far Eastern [14]. This includes languages written with complex scripts such as Arabic, Hebrew, Thai, Devanagari, and Tamil. The shaping of Indic scripts is handled by a Unicode Script Processor called as Uniscribe. Below we present the important points about Uniscribe. For a complete detailed description the reader is referred to [14].

7.1. Unicode Script processor

Windows NT 5.0 includes the new Uniscribe software, that supports line measurement, display, caret movement, character selection, justification, and line breaking of Unicode plain text. It implements rules governing the shaping and positioning of glyphs as specified and cataloged within the Unicode standard for applications performing complex text layout such as is required for Indic Text.

Uniscribe is composed of multiple 'shaping engines.' These shaping engines contain the layout knowledge for particular scripts (for example, Arabic, Hebrew, Thai, Hindi, Tamil). In addition, there is an OpenType Layout shaping engine for handling script features unknown to Uniscribe. It handles characters in 'clusters,' and the Uniscribe data structure is a large array of clusters. Uniscribe identifies cluster boundaries and thus achieves the granularity required by Indic and other complex scripts.

The overall system architecture in which the Uniscribe shaping engine gets employed is further described in the next subsections.

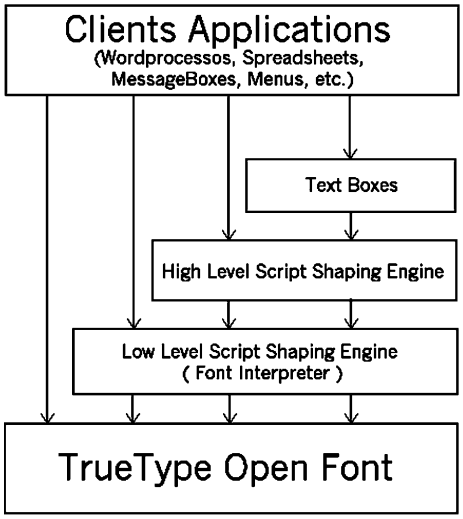

Fig. 27. Levels of shaping abstraction.

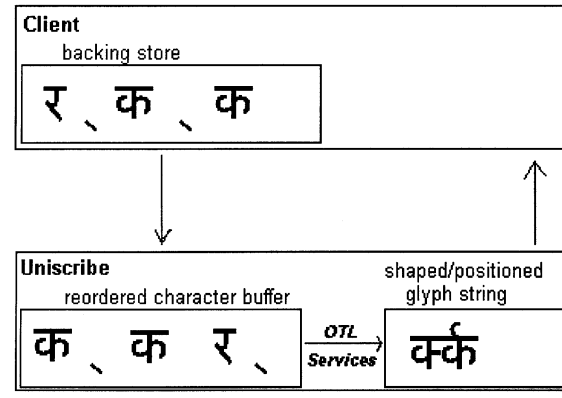

Fig. 28. Character reordering and backbuffer held by Uniscribe.

8. The text-layout client model

A text-layout client may apply certain script features, or it may rely on operating system services to apply features, or it may do both. The block diagram in Fig. 27 illustrates different implementation models, and the different approaches are discussed below.

Responsibilities for tracking and performing text-layout operations vary depending on a client's implementation model, such as, whether a client handles line breaking.

8.1. Models for client support of complex scripts

As illustrated above, the text-layout client model can have the following forms:

- *Client calls Textbox*
- *Client calls Shaping engine, say Uniscribe*
- *Client calls System supported Font Layout Services*
- *Client implements TTO Layout Services Directly*

What the Client Gets for Free when using Uniscribe Services

- Cluster breakup based on Unicode character values and specified language.
- Positioning of glyphs with in clusters.

9. Managing character reordering

Using Uniscribe, clients need only manage a backing store of Unicode character codes. Text-layout clients do not need to maintain any other glyph code buffer or mapping table to track character order. Consequently, clients can search and index shaped text easily, because the backing store never changes as a result of layout operations (Fig. 28).

10. Conclusion

The availability of shaping engines for Indic scripts at the Operating System level is a major advance for Indian languages. About nine hundred million people in India are not conversant with English. The easy availability of digital information in one's own language will have a far reaching effect socially, economically and even culturally in a multi-lingual country like India. The shaping engine architecture and the shaping process that we have described in this paper are fairly simple for implementation in software. At the same time adequate care has been exercised to ensure that all the finer nuances of writing in each of the languages are in no way missed out. And at the same time adequate flexibility is retained in the design of fonts, etc. We do hope that in the near future other platforms will also support Indic scripts.

Acknowledgements

Over the last two decades the National Centre for Software Technology has been working in this area, building shaping engines, font design systems, text processing packages, multi-lingual applications, etc. A very large number of persons have contributed on a continuing basis to this effort. The number is just too large to list here. The authors are grateful to all past members of the Graphics and CAD division, graphics and linguistic design consultants who have been associated with this activity, and to the Director NCST for his encouragement. Specifically, in more recent times, the authors

are grateful to Microsoft Corporation, USA, for providing the opportunity to build in some of this know how into the Windows NT 5 Operating system at the GDI level.

References

[1] Mudur SP, Wakankar LS. Computer input output in Devanagari. (Invited) Proceedings of the Symposium on Use of Indian Languages in Computer Based Information Systems, New Delhi, May 1978.

[2] Mudur SP, Narwekar A, Moitra A. Design of software for text composition. Software Practice and Experience 1979;9:325–37.

[3] Moitra A, Mudur SP, Narwekar A. Design and analysis of a hyphenation algorithm. Software Practice and Experience 1979;9:325–37.

[4] Mudur SP, Ghosh PK, Sujatha R. Software development for computer typography and type design. NCSDCT Annual Research Review 1981;3:13–26.

[5] Mudur SP, Wakankar LS, Ghosh PK. Design information on text composition in Devanagari. Published by Research Institute for Newspaper Development, 1980.

[6] Mudur SP, Ghosh PK. Computer aided text composition in Indian scripts. Proceedings of the International Conference INFORMATICS, Vol. 81, New Delhi, 1981.

[7] Mudur SP, Sujatha R. Three systems for typesetting: a survey. Computer science and Informatics, 1982;12(1):28–36.

[8] Mudur SP, Pattanaik SN, Nath SJ. Computer processing of Indian scripts – A pure consonant approach. Proceedings of the National Seminar on Computer Aided Language Processing, Delhi, 1987.

[9] Mudur SP, Pattanaik SN, Nath SJ, VIDURA: an interactive multilingual publishing system. In: vanVliet JC, editor. Document Manipulation and Typography. Cambridge; Cambridge University Press, 1988:249–60.

[10] Department of Official Languages, Prabodh Primer. Ministry of Home Affairs, Government of India.

[11] The Unicode Standard Version 2.0. The Unicode Consortium. Addison-Wesley Developers Press, Reading, MA, 1996.

[12] TrueType Open Font Specification Version 1.0. Microsoft. July 1995.

[13] IS 13194, Indian Script Code for Information Interchange — ISCII. Bureau of Indian Standards, December 1991.

[14] Bishop FA, Brown DC, Meltzer DM. Supporting Multilanguage Text Layout and Complex Scripts with Windows NT 5.0. Microsoft Systems Journal, 1998: http://www.microsoft.com/msj/1198/multilang/multilang.htm.