Personalized SMS in Indian Languages

Devanuj
TCS Innovation Lab - Mumbai
Yantra Park, Thane, INDIA
+912267788295
Devanuj.1@tcs.com

Sunil Kumar Kopparapu
TCS Innovation Lab - Mumbai
Yantra Park, Thane, INDIA
+912267788216
Sunilkumar.Kopparapu@tcs.com

ABSTRACT
Handwritten script synthesis is an attempt to impart individuality, which is diminishing due to growing dependence on the keyboard and keypads as text input tools, to electronic text. In this paper we propose a system which enables users to send personalized SMS in Indian languages. The SMS is personalized in the sense that the message received by a recipient is in the sender's handwriting, though the input is in Roman script. The system has been tested for two Indian languages, Telugu and Marathi.

1. INTRODUCTION
Pen has been the dominant instrument for writing. Today it is being replaced by electronic keys. E-mail has taken over post, and for writing notes, we have started using mobile phones. Though generation and transfer of text are easier in a keyboard/keypad centered scenario, the text itself lacks a sense of individuality. In a world dominated by pen, in contrast, every text used to have a personal imprint of the writer. Another problem related to text based communication is the fact that most input and output devices are not much suitable for non-Roman scripts. The problem becomes more interesting in case of mobile phones (which have become popular text based communication platform). A mobile keypad, for instance, has lesser number of keys than a regular PC keyboard and that results in the need for mapping many characters to a single key. The problem is aggravated by the fact that Indic scripts have many more characters than English. As a result, an acceptable scheme for input of Indic characters still needs to be evolved [17]. The case with the standards for display of the Indic scripts [16] is similar. Although people use phonetized Roman for their immediate needs, ability to read a text in native script results in a better user experience. In this paper, we deal with the above mentioned issues by addressing the following:

- Personalizing a typed text message in a person's own handwriting using handwritten samples provided by her.
- Rendition of the message in an Indian language that has been typed using a phonetic notation in Roman (script), in the corresponding Indic script.

TACTiCS – TCS Technical Architects' Conference'09

We address the two problems by proposing an Indic script synthesis system that is able to render a typed (Roman) text message in the sender's Indic handwriting and electronically send it to a recipient. The rendition process is accomplished with the help of handwriting samples, which are in the form of primitives derived from the characters of the Indic script. We propose a generic framework for synthesizing the script, which forms the underlying basis of the Indic script synthesis system. In order to define the framework, we attempt to seek a minimal set of cursive strokes derived from the alphabet of the script and use it to render the message. The focus therein is formalization of the process for formation of complex shapes from the basic cursive strokes based on the generic characteristics of Indic writing systems. Section 2 describes the architecture of the Indic script synthesis system. Section 3 describes a brief discussion on the characteristics and specific issues pertaining to the Indic writing systems. Section 4 outlines the framework, and we give some experimental results in Section 5.

2. SYSTEM ARCHITECTURE
Figure 1 captures the high level detail of the Indic script synthesis system. A set of primitive samples (for a user) are captured (beforehand) as a sequence of (x, y) coordinate pairs by using an electronic device, such as a graphics tablet or a digital pen. The set of primitives are specific to a language script. The primitive samples are stored in a database and tagged to the user's phone number. The synthesis of the handwriting for a user uses text in Roman script as input. The user types the message as an SMS using the keypad of a mobile phone. The SMS, has the following format:
<service_identifier_prefix><recipient's_number><actual_message_text>

Within the script synthesis system, an SMS gateway parses the SMS and passes the message and the recipient's number to a server script that, in turn, invokes the script synthesis engine with the message text as input. The output of the engine is an image of handwriting. The SMS gateway, on being notified about the completion of the rendition process, sends an SMS to the recipient that contains a link to the hosted image. The recipient can use her mobile phone's browser to display the synthesized message. Figure 2 provides a detailed view of the working of the script synthesis engine. The engine has several modules. The parser module segments the message text into syllabic segments using a database of (language dependent) combination rules, that are based on the framework described later in this paper.

Subsequently, the mapper module maps the segments to Indic *aksharas*. The *aksharas* are rendered by choosing the required primitive samples and then suitably combining them based on the combination rules. Finally, the engine assembles and formats the synthesized *aksharas* and generates the output in the form of an image.

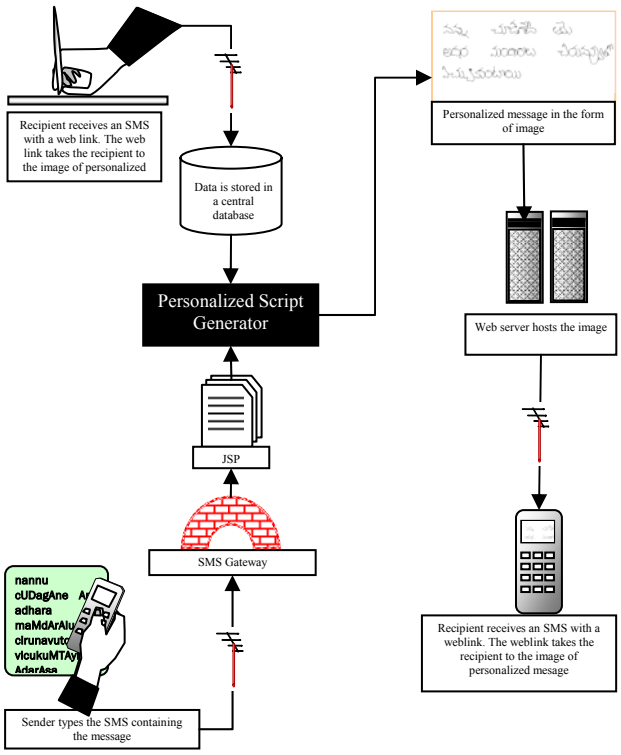


Figure 1: System Architecture

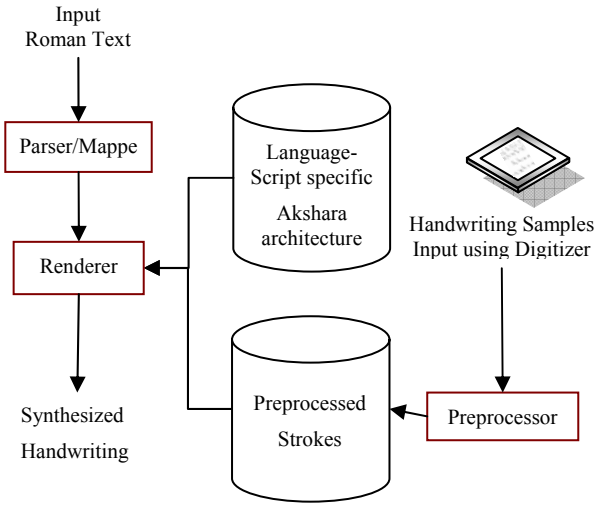


Figure 2: Detailed View of the Script Synthesis Engine

3. SYNTHESIS OF INDIC SCRIPTS

The problem of handwriting synthesis is to convert input text in a person's handwriting, using sample handwritten primitives, characters or words provided by her. Handwriting synthesis in literature uses three main approaches namely (a) motor models of handwriting generation, (b) grapheme based approaches and (c) those based on statistical models. The mechanical approach was adopted by Hollerbach [10], who modeled handwriting generation as a result of modulated oscillation across the vertical and horizontal axes, superimposed over a linear horizontal movement. One of the earliest and important works using a grapheme based approach was attempted by Guyon [5], who used commonly occurring glyphs to regenerate cursive English handwriting. Rao et al [6], on other hand, approached the problem as a connected articulation of isolated character shapes. Use of statistical approach can be noticed in the work of Wang et al [3], where they extracted the intrinsic letter and concatenation styles from the handwritten strokes to synthesize script. All the examples cited above are for Roman scripts. As for non-Roman scripts, Choi et al [1] generated Hangul script by using Bayesian network based on-line handwriting recognizers to train the system for synthesizing script using handwritten samples. Using a generic framework for Indic scripts, Jawahar et al [14] generated synthesis models from training data and employed them to generate handwriting for Roman input. They used two types of models: one for modeling the individual strokes, and the other for the spatial relationship between two strokes. They used a data of 14000 strokes and over 2000 words collected from 5 people covering 15 pages.

3.1 Graphical Representation of the Indic languages

Most of the Indic scripts are derived from Brāhmi and share the trait of being alpha syllabic [13]. On the representational level, these scripts follow identical rules regarding the formation of a grapheme corresponding to a given phoneme (see Table 3-1). Mudur et al. [15] provide an excellent description of Indic writing system in their work for laying out requirements for software localization for Indic scripts. The characteristic rules are the following:

Table 3-1: Rendition of text "Mahatma Gandhi" in various Indic scripts.

Script	Rendition	Break up of the rendition	
Roman	*Mahātmā Gāndhī*	*ma + hā + t + mā*	*gā + n + dhi*
Tamil	மகாத்மா காந்தி	ம + கா + த் + மா	கா + ந் + தி
Bengali	মহাত্মা গান্ধী	ম + হা + ত্ + মা	গা + ন্ + ধী
Devanagri	महात्मा गांधी	म + हा + त् + मा	गा + न् + धी
Gujarati	મહાત્મા ગાંધી	મ + હા + ત્ + મા	ગાં + ન્ + ધી
Telugu	మహాత్మా గాంధీ	మ + హా + త్ + మా	గా + న్ + ధీ

Rule 1: The vowels and the consonants are divided as two categories. The consonants inherently carry an 'a' [International Phonetic Association's (IPA)[1] notation: ^] vowel with them. In order to suppress this sound, one adds a special symbol (called *virāma* in Sanskrit). While writing, a silenced consonant can be written as a combination of simple consonant and a *virāma*. However, quite often a modified form of the consonant is used to represent the silencing.

[1] In this paper, we have used IPA notation (in italics) for Romanization of Indic text

Rule 2: The vowels exist in two forms, as standalones; and as diacritics acting as sound modifiers (called *mātrās* in Sanskrit). For example, in Telugu, 'ఆ' is used to denote standalone form of the vowel *ā*, and 'ా' is used as a vowel-modifier for the consonants (for example, Telugu *kā* (కా) and *pā* (పా).

Grapheme representation of the basic phonetic unit is called an *akshara*. An *akshara* is written in either of the ways (refer Table 3-2):

- A single consonant (consonant carry a schwa sound with them)
- An independent vowel
- A consonant followed by a *mātrā*
- A consonant stripped of its schwa sound with the help of a special symbol named as *halanta* or *virāma*.
- A sequence in the following form:

  C`C`…C`CV` or CV`C`C`…C`, where C` represents a half consonant; C full consonant, and V` *mātrā*.

Table 3-2: Formation of *akshara*s

Hindi	Telugu	SAMPA	Remarks
क	క	ka	standalone consonant
क्	క్	k	silenced consonant, constructed by appending *virāma*
क्य	క్య	kya	one silenced consonant + one simple consonant (please note that in case of Hindi, the silenced consonant has changed its form)
ई	ఈ	i:	standalone vowel, single *akshara*
ी	ీ	i:	the modified vowel acting as a transformer, cannot exist independently
की	కీ	ki:	consonant + modified vowel, single *akshara*
कई	కఈ	kai:	standalone consonant + standalone vowel, two *akshara*s

3.2 Issues regarding synthesis of the Script

There are broadly three issues that are encountered while trying to synthesize Indic scripts. First issue is that the number of *aksharas* possible for an Indic script is very large. For example, considering only CV, C`CV and V types of *aksharas*, if there are c_v vowels and c_c consonants, then there are $(1 + c_c + c_c^2) * c_v$ distinct possibilities (of *akshara* formation). Second issue is that Roman is a 'linear' script, where the successive graphemes are put to the right of their predecessors. In the case of Indic scripts, though the full consonants are put successively on a horizontal axis, half consonants and the *mātrā*s can have alternative positions with respect to the full consonant. In other words, they can be placed to the left, right, top or bottom of a (full) consonant (see Table 3-3). In the case where a *mātrā* has to be placed on the left, one has to beforehand make provision for the placement (of the *mātrā)*. Another issue is that of the text input. Computer keyboards are more suitable for the Roman script, in which case the graphemes do not change their forms. In Indic languages, the graphemes do so depending on both the phonetic and the graphical context (see Table 3-3). In case of text editors this problem is solved with the help of shaping engines [11].

Table 3-3: Interaction of *mātrās* with the consonants

Language	Orthography	SAMPA	Remarks
Hindi	क् + आ = का	k + a: = ka:	*mātrā* has succeeds the consonant
Hindi	क् + इ = कि	k + i = ki	*mātrā* has precedes the consonant
Hindi	क् + उ = कु	k + u = ku	*mātrā* is sitting below the consonant
Tamil	ல் + உ = லு	l + u = lu	*mātrā* abuts the consonants on two sides
Telugu	న్ + ఉ = ను	n + u = nu	*mātrā* succeeds the consonant

4. THE FRAMEWORK FOR SYNTHESIS

The idea is to use a reduced dataset derived from the alphabets to synthesize valid *aksharas* for a particular language-script combination. The framework uses a set of rules for forming the *aksharas*. We define the set of rules as *akshara* architecture. *Akshara* architecture is specified in a space separate from rendition and parsing mechanisms. In this section we describe the important elements of the framework.

4.1 The Basis

In order to describe the reduced dataset we define a stroke as a discrete and finite set of contiguous points. The start and end of a stroke is defined by the pen up and pen down instances. The reduced dataset forms the basis[2] for constructing the *aksharas* of a script. The basis (dataset) for each language is defined by an expert by visual observation (see Figure 12 for Telugu script and Figure 15 for Devanagri). Once the basis is defined, the dataset is instantiated for a user by her writing the strokes (defined by the elements of the basis) using a digitizer.

Let $\boldsymbol{A} = \{S_0, S_1, S_2, \ldots S_m\}$ represent m strokes in the dataset. Each of the strokes in $\boldsymbol{A}$ is represented by a set of points namely,

$$S^i = (P_0^i, P_1^i, P_2^i, \ldots P_k i_{-1}^i), \quad (1)$$

where a point P_j^i denotes the j^{th} point in an i^{th} stroke.

Formally,

$$P_j^i = (x_j^i, y_j^i) \quad (2)$$

For a script having m strokes, the dataset (A) is of size[3] m, and each stroke S_i in the dataset is made up of k_i points[4]. Additionally, we define the sets of x and y components of S^i as,

$$S_x^i \equiv (x_0^i, x_1^i, \ldots x_j^i \ldots x_k i_{-1}^i) \quad (3)$$

[2] A basis as defined here is a data set (vector) that is complete in the sense that the elements of dataset can be used (linearly combined) to construct all the character in the language (vector space), additionally no element of the basis can be constructed from a combination of other elements of the basis.

[3] |A|=m (cardinality of A is m)

[4] Assuming that the input is through a digitizer, a stroke can be defined as curve represented as a sequence of points between two pen lifts.

$S_y^i \equiv (y_0^i, y_1^i, \ldots y_j^i \ldots y_{k_i-1}^i)$ (4)

4.2 Hook Points

The concept of hook points is similar to that of bimolecular binding sites, which are specific regions where other molecules can attach. A hook point can be used to join a given stroke (defined as primary stroke) with another stroke (defined as secondary stroke) for the purpose of synthesis of an *akshara.* We define two kinds of hook points: regular and virtual.

4.2.1 Regular Hook Points

An analysis of joining pattern of the Indic scripts reveals that majority stroke joining takes place at

- Points lying at the x and y limits of the stroke
- Terminals
- Points of local inflection,
- Intermediate points between the two terminals

Hence, for each stroke, the regular hook points are defined at the four types of positions as described above. We represent the regular hook points (see Figure 3) by the set R^i, where $R^i \in S^i$.

4.2.2 Virtual Hook Points

A virtual hook point can be thought of as an imaginary point, conceptualized by pairing x-value of one point on the stroke, and y-value of another point in the same stroke. A virtual hook point may not be present in a stroke's path, and, thus does not have an index denoting its position in the sequence of points.

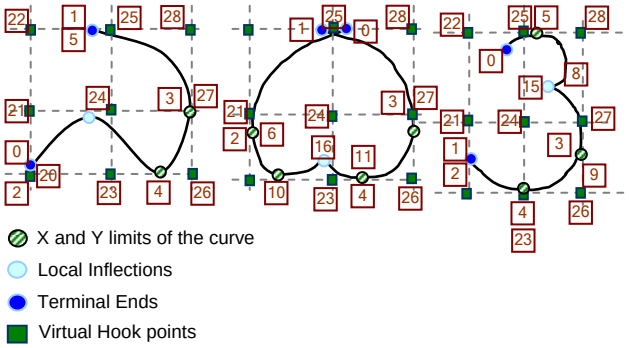


Figure 3: Regular and Virtual Hook Points (the numbers within the square denote the references to Table 4-1)

A virtual hook point (see Figure 3) can be defined as

$V^i = \{(x_p^i, y_q^i): x_p^i \in S_x^i \text{ and, } y_q^i \in S_y^i\}$ (5)

The set of all the hook points for a stroke S^i is defined as,

$H^i = \{ H^i : H^i \in R^i \cup V^i \}$ (6)

We define the hook point types by giving each type a number. It is tabulated in Table 2-1. Given the stroke information and hook point type number (T), the actual position (P) of the hook points in the Cartesian space can be determined by the lookup function (f_{det_hook}) using the look up table,

$f_{det\_hook}: \mathrm{N} \rightarrow \mathrm{X} \times \mathrm{Y}$ (7)

$(x_p^i, y_q^i) = f_{det\_hook}(S^i, T); (x_p^i, y_q^i) \in H^i$ (8)

Table 4-1: Types of Hook Points

Type	Type Description(D)	Actual Positions (P)
No. (T)		
0	Starting point	(x_0^i, y_0^i)
1	Ending point	$(x_{k_i-1}^i, y_{k_i-1}^i)$
2	Left limit	(x_α^i, y_α^i): $x_\alpha^i = \min(x_0^i, x_1^i, x_2^i, \ldots x_{k_i-1}^i)$
3	Right limit	(x_β^i, y_β^i): $x_\beta^i = \max(x_0^i, x_1^i, x_2^i, \ldots x_{k_i-1}^i)$
4	Lower limit	(x_γ^i, y_γ^i): $y_\gamma^i = \min(y_0^i, y_1^i, y_2^i, \ldots y_{k_i-1}^i)$
5	Upper limit	(x_δ^i, y_δ^i): $y_\delta^i = \max(y_0^i, y_1^i, y_2^i, \ldots y_{k_i-1}^i)$
6	Left limit of the previous half of the curve	$(x_\varepsilon^i, y_\varepsilon^i)$: $x_p^i = \min(x_0^i, x_1^i, x_2^i, \ldots x_{0.5k_i-1}^i)$
7	Left limit of the latter half of the curve	(x_ζ^i, y_ζ^i): $x_\zeta^i = \min(x_{0.5k}i^i, x_{0.5k_i+1}^i, x_{0.5k_i+2}^i, \ldots x_{k_i})$
8	Right limit of the previous half of the curve	(x_η^i, y_η^i): $x_\eta^i = \max(x_0^i, x_1^i, x_{0.5k}i_{+1}^i, x_{0.5k_i+2}^i, \ldots x_{0.5k_i-1}^i)$
9	Right limit of the latter half of the curve	(x_θ^i, y_θ^i): $x_\theta^i = \max(x_{0.5k}i^i, x_{0.5k_i+1}^i, x_{0.5k_i+2}^i, \ldots x_{k_i})$
10	Lower limit of the previous half of the curve	(x_ι^i, y_ι^i): $y_\iota^i = \min(y_0^i, y_1^i, y_2^i, \ldots y_{0.5k_i-1}^i)$
11	Lower limit of the latter half of the curve	(x_κ^i, y_κ^i): $y_\kappa^i = \min(y_{0.5k}i^i, y_{0.5k_i+1}^i, y_{0.5k_i+2}^i, \ldots y_{k_i})$
12	Upper limit of the previous half of the curve	$(x_\lambda^i, y_\lambda^i)$: $y_\lambda^i = \max(y_0^i, y_1^i, y_2^i, \ldots y_{0.5k_i-1}^i)$
13	Upper limit of the latter half of the curve	(x_μ^i, y_μ^i): $y_\mu^i = \max(y_{0.5k}i^i, y_{0.5k_i+1}^i, y_{0.5k_i+2}^i, \ldots y_{k_i})$
14	Inflexions between 6 and 7	(x_ν^i, y_ν^i): $x_\nu^i = \max(x_\varepsilon^i, x_{\varepsilon+1}^i, x_{\varepsilon+2}^i, \ldots x_\zeta^i)$
15	Inflexions between 8 and 9	(x_ξ^i, y_ξ^i): $x_\xi^i = \min(x_\eta^i, x_{\eta+1}^i, x_{\eta+2}^i, \ldots x_\theta^i)$
16	Inflexions between 10 and 11	(x_o^i, y_o^i): $y_o^i = \max(y_\lambda^i, y_{\lambda+1}^i, y_{\lambda+2}^i, \ldots y_\mu^i)$
17	Inflexions between 12 and 13	(x_π^i, y_π^i): $y_\pi^i = \min(y_\eta^i, y_{\eta+1}^i, y_{\eta+2}^i, \ldots y_\theta^i)$
18	Intermediate point between 3 and 5	(x_ρ^i, y_ρ^i), where $\rho = (\beta + \delta)/2$
19	Intermediate point between 4 and 5	$(x_\varsigma^i, y_\varsigma^i)$, where $\varsigma = (\gamma + \delta)/2$
20	Bottom left of the rectangle binding the stroke (virtual)	(x_{min}^i, y_{min}^i), where $x_{min}^i = x_\alpha^i$; $y_{min}^i = y_\gamma^i$
21	Middle left of the rectangle binding the stroke (virtual)	(x_{min}^i, y_{mid}^i), where $y_{mid}^i = (y_\gamma^i + y_\delta^i)/2$
22	Top left of the rectangle binding the stroke (virtual)	(x_{min}^i, y_{max}^i), where $y_{max}^i = y_\delta^i$
23	Bottom middle of the rectangle binding the stroke (virtual)	(x_{mid}^i, y_{min}^i), where $x_{mid}^i = (x_\alpha^i + x_\beta^i)/2$
24	Exact centre of the rectangle binding the stroke (virtual)	(x_{mid}^i, y_{mid}^i)
25	Top middle of the	(x_{mid}^i, y_{max}^i)

	rectangle binding the stroke (virtual)	
26	Bottom right of the rectangle binding the stroke (virtual)	$(x_{max}^{\ i}, y_{min}^{\ i})$, where $x_{max}^{\ i} = x_{\beta}^{\ i}$
27	Middle right of the rectangle binding the stroke (virtual)	$(x_{max}^{\ i}, y_{mid}^{\ i})$
28	Top right of the rectangle binding the stroke (virtual)	$(x_{max}^{\ i}, y_{max}^{\ i})$

4.3 *Akshara* Architecture

The basic operation in building of an *akshara* is that of joining two strokes S^p, S^q. The hook points act as attributes to the joint since they describe the mutual spatial relationship of the two strokes. Joining the p^{th} stroke (using m^{th} type of hook point) with q^{th} stroke (using n^{th} type of hook point) is defined as copying the q^{th} stroke from the dataset and translating with respect to the p^{th} stroke so that the respective hook point of the q^{th} stroke is exactly in the same position as that of the p^{th} stroke. The Cartesian value of a hook point is determined by the function f_{det_hook} [Eq^n. (7)] The *akshara* architecture constitutes joining rules needed for the formation of every valid *akshara* for the language-script. A joining rule is made up of simpler binary joining operation. A simple binary joining operation can be denoted in the following notation:

$$\vec{S}^{\,p}.h_m^p \oplus S^q.h_n^q \qquad (9)$$

The arrow on the first stroke $\vec{S}^{\,p}$ denotes that it has already been positioned on the view port. A joining rule is defined as a set of binary operations,

$$\mathbf{R_x} \equiv \left[\begin{array}{l} \vec{S}^{\,p}.h_a^p \oplus S^q.h_b^q \\ \vec{S}^{\,p}.h_c^p \oplus S^r.h_d^r \\ \cdots \end{array}\right] \qquad (10)$$

here, x denotes the index for an *akshara*.

The above notation means that the rendition is done in the following manner:

- S^p is placed on the view port
- S^q is copied from the dataset and translated to be joined to the already positioned S^p, in a way such that a^{th} hook point of S^p and b^{th} one of S^q are aligned
- S^r is joined similarly with S^p, using, their o^{th} and p^{ith} hook points, respectively.

Given a joining rule, one can either form the whole *akshara* before rendition, or, render the strokes piecemeal wise according to the binary operation encountered. We followed the latter approach.

4.4 Transition Generation

The operation $\mathbf{R_x}$ can be made more natural by generating smooth transitions between subsequent strokes. The shape of a handwritten *akshara* may differ from its typed counterpart owing to the fact that handwritten text is usually devoid of any abrupt changes in curvature due to the inertia of the hand-arm-pen system. This gives birth to smooth transitions between two strokes. This also helps us to reduce the elements of the basis, because many of the handwritten forms, that look unique are actually formed because of the transitions (see Figure 11). To capture this characteristic of the handwritten text, we generate transitional trajectories at the joints as suggested by Rao and Ramasubramanian in [6] by using a sigmoid function. We generate intermediate points in such a way that the first point of the generated sequence lies on the first stroke and the last point on the second stroke. To accomplish this we have a concept of prefix and suffix. Prefix (suffix) is a subset of a stroke comprising of first (last) few points of the stroke. The length of the prefix needs to be equal to that of the suffix. A generated point is given by:

$$P_t` = (P^a_{k^a-N+t} + P^b_t e^{-t}) / (1+e^{-t}),$$
$$0 < t < N,\ N=\min(p_m, s_n) \qquad (11)$$

where, p_m and s_n are the number of points in prefix of the second stroke and suffix of the first stroke respectively and k^a is the total number of points in the first stroke. The transition point generation process can be appreciated from Figure 9 and Figure 10.

4.5 Synthesis

The process is depicted in Figure 2. The strokes constituting the dataset corresponding to a user are captured using an electronic device, such as a graphics tablet or a digital pen. The strokes are in the form of disconnected points. Preprocessing is used to generate smooth curves and also helps in size normalization. The input Roman text is parsed into syllabic segments, which are mapped to stroke combination-rules that are needed to render the Indic *aksharas* represented by the syllable. The *aksharas* are realized by choosing the required strokes from the dataset and then suitably combining them based on the rules inherent in the *akshara* architecture. Finally, the rendering module places the synthesized *aksharas* on the output, which is in the form of an image.

4.6 Preprocessing

Size normalization is a prerequisite since the person providing the samples does not have the word level context to decide the size of individual strokes. In the preprocessing stage, the raw data is normalized with respect to scale. Based on size, the strokes are divided into two classes, namely, the *mātrās*, which are generally smaller in size; and the standalone vowels and consonants. The size normalization is done with respect to the average size of the strokes in that class, instead of the global average (Figure 4).

The strokes may not result in a smooth curve on account of constraints posed by the capturing mechanism. Therefore, smoothening of strokes is accomplished by applying a smoothening filter (Figure 5), for example,

$$P_j^{i`} = \sum_{t=0}^{N} ({}^N C_t / 2^N) P_{(j-0.5N+t)}^{\ i},\ j \in [0.5N,\ k^i\text{-}0.5N\text{-}1],$$
$${}^n C_r = n! / r!(n\text{-}r!) \qquad (12)$$

where N is the size of the smoothening window.

Finally, the disjoint points in the stroke, are joined to produce the final stroke (Figure 4).

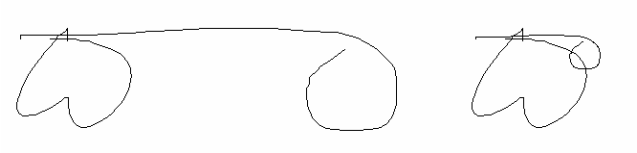


Figure 4: A consonant and *mātra* combination, normalized with respect to global average(left) , and, normalized with respect to class average (right). The latter rendition is correct.

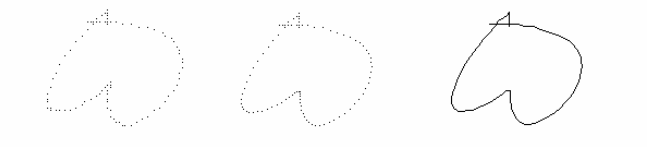


Figure 5: Jagged stroke (left); Smoothened but unfilled stroke (middle), and, Filled stroke (right)

4.7 Parsing and Mapping

Usually, text entry for Indic scripts is normally done in one of the two ways. One is graphical approach and the other is phonetic. We follow the second approach for text input. We use a transliteration scheme similar to ITRANS for parsing the text [12]. We parse an input text using the transliteration table (that helps in parsing the strings into syllabic substrings) in such a way that the substrings represent valid phonemes (see Table 4-2). The parsing is done by trying to lookup the first *n* characters of the string, where *n* is the length of the largest Roman string in the lookup table. In case the sub-string is not found in the lookup, the window size is reduced (see Figure 6). As a result of parsing, a sequence of phonemes is generated, which is given by

$$\varphi_{1,1},\varphi_{1,2}\cdots\varphi_{1,k(1)}\ldots\varphi_{2,1},\varphi_{2,2}\cdots\ \varphi_{a,b}\ldots\varphi_{1,k(2)}\ldots..\ \varphi_{3,1},$$
$$\varphi_{3,2}\cdots\varphi_{1,k(2)}\ldots\varphi_{N,1},\varphi_{N,2}\cdots\varphi_{1,k(N)} \quad (13)$$

where, $\varphi_{a,b}$ represents the b^{th} phoneme in a^{th} word of the sequence, and $k(a)$ represents the number of phonemes in a^{th} word.

The phoneme sequence is mapped to a sequence of *aksharas* given by

$$x_{1,1},x_{1,2}\ldots x_{1,k(1)}\ldots x_{2,1},x_{2,2}\ldots\ x_{a,b}\ldots x_{1,k(2)}\ldots..\ x_{3,1},$$
$$x_{3,2}\ldots x_{1,k(2)}\ldots x_{N,1},x_{N,2}\ldots x_{1,k(N)} \quad (14)$$

where, $x_{a,b}$ represents the b^{th} *akshara* in a^{th} word of the sequence and $k(a)$ represents the number of *aksharas* in a^{th} word.

Table 4-2: Lookup table for parsing Marathi input

Roman	Indic	Roman	Indic
a	अ	ow	औ
A	आ	M	अं
i	इ	a:	अः
I	ई	k	क
u	उ	kh	ख
U	ऊ	g	ग
e	ए	gh	घ
E	ऐ	NG	ङ
o	ओ	c	च
ch	छ	t	त
j	ज	th	थ
jh	झ	d	द
nX	ञ	dh	ध
T	ट	n	न
Th	ठ	p	प
D	ड	ph	फ
Dh	ढ	b	ब
N	ण	bh	भ
m	म	l	ळ
y	य	ksh	क्ष
r	र	Ro	ऋ
L	ल	Tr	त्र
v	व	Gny	ज्ञ
S	श	trru	तृ
Sh	ष	Dy	द्य
s	स	Shr	श्र
H	ह	OM	ॐ

ShrAvaN	Shr AvaN
ShrAvaN	Shr AvaN
ShrAvaN	Shr AvaN
ShrAvaN	Shr AvaN
ShrAvaN	Shr AvaN
ShrAvaN	Shr AvaN
Shr AvaN	
Shr AvaN	Shr AvaN
Shr AvaN	श्र ा व ण

Figure 6: Parsing and mapping of Roman Input

4.8 Rendition

An *akshara x* mapping to a phoneme φ can be rendered on the view port according to the corresponding rule $\mathbf{R}_x$ [Eq^n.(10)]. If placement point for the *akshara*, P_x is known, rendering on the view port can be represented as,

$$P_x \oplus \mathbf{R}_x \qquad (15)$$

The placements on viewport of all the *aksharas* in a sequence are dependent on the placement of the previous *akshara* of the sequence. The rendition of the *akshara* sequence shown in the previous section [Eq^n. (14)] can thus be depicted as

$$P_o \oplus$$
$$[R_{1,1} \oplus d \oplus R_{1,2} \oplus d \oplus \ldots \oplus R_{1,k(1)-1} \oplus d \oplus R_{1,k(1)}]$$
$$\oplus X^{gap} \oplus$$
$$[R_{2,1} \oplus d \oplus R_{2,2} \oplus d \oplus \ldots \oplus R_{2,k(2)-1} \oplus d \oplus R_{2,k(2)}]$$
$$\oplus X^{gap} \oplus$$
$$\ldots$$
$$\oplus X^{gap} \oplus$$
$$[R_{a,1} \oplus d \oplus \ldots \oplus R_{a,b} \oplus \ldots \oplus d \oplus R_{a,k(a)}]$$
$$\oplus X^{gap} \oplus$$
$$\ldots$$
$$\oplus X^{gap} \oplus$$
$$[R_{N,1} \oplus d \oplus R_{N,2} \oplus d \oplus \ldots \oplus R_{N,k(N)-1} \oplus d \oplus R_{N,k(N)}] \qquad (16)$$

Here, P_o is the placement point for the first *akshara*, d is the inter-*akshara* distance, X^{gap} is the inter word distance, and $R_{a,b}$ is the joining rule for *akshara* $x_{a,b}$. The rendition process is detailed out in Figure 8 and Figure 9.

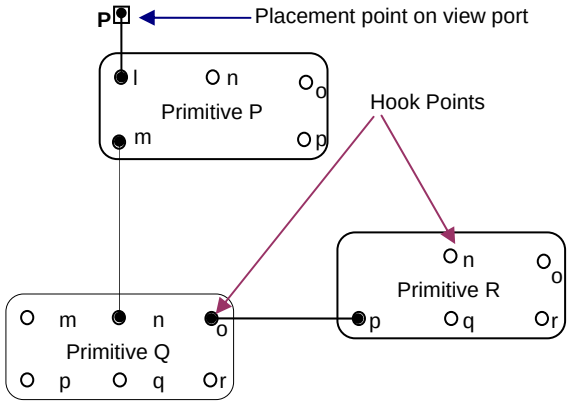


Figure 7: Schematic Representation of the Akshara Formation Process

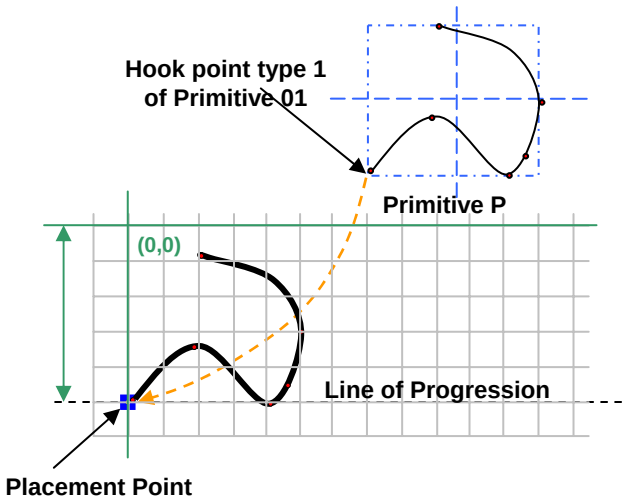


Figure 8: Placement of Primary Stroke

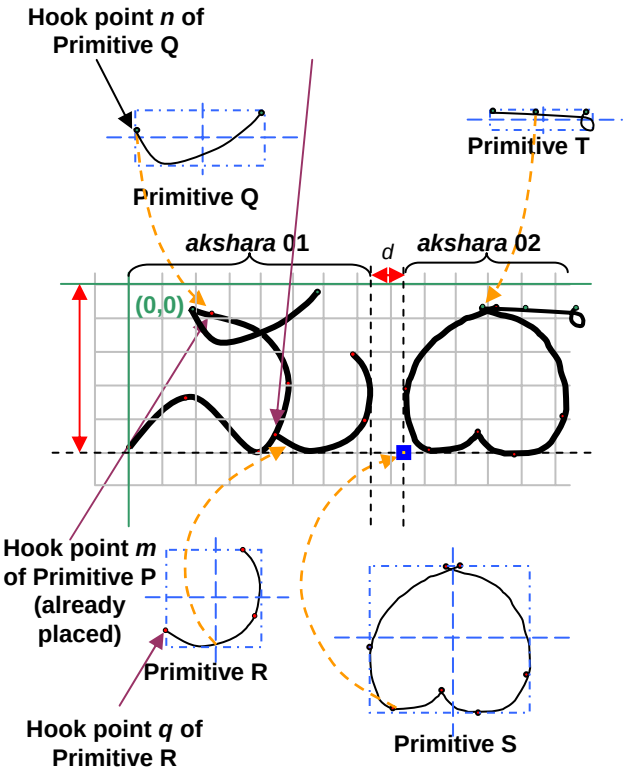


Figure 9: Placement of Subsequent Strokes

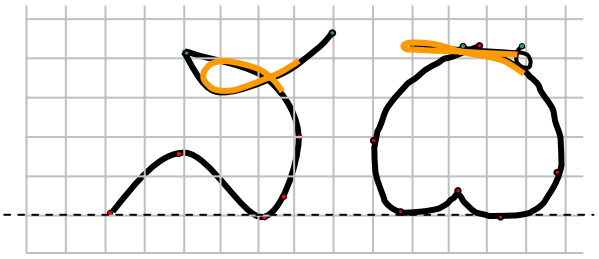


Figure 10: Generation of Transitions

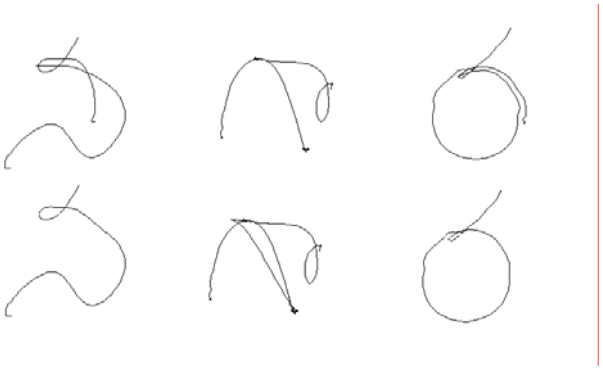


Figure 11: Examples of Forms Resulting due to Transitions

5. RESULTS

The synthesis engine was developed in C++. The input data was collected using an e-Pen, which is a pen motion capturing device. The generated output was in the form of .pgm and .jpg image files. The framework was evaluated for two scripts: Devanagri and Telugu. The primitives were decided by analyzing handwritten documents. The sample inputs and the outputs are shown in Figure 12 to Figure 17. A demonstrable system for rendering personalized message was also developed. The mobile phone component was written using BREW (Binary Runtime Environment for Wireless) SDK, while the server side script was in the form of a PHP page hosted by WAMP server. The PHP script, in turn, was used to run the script synthesis engine.

5.1.1 Language/Script: Telugu/Telugu

Input: `nannu cUDagAne Ame adhara maMdArAlu viccukuMTAyi`

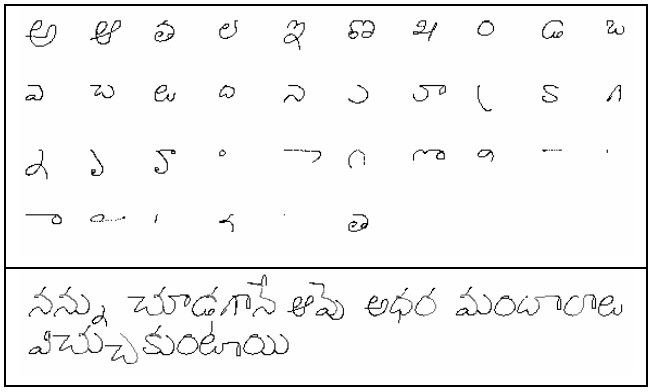


Figure 12: Samples and Output for Telugu (User TE01)

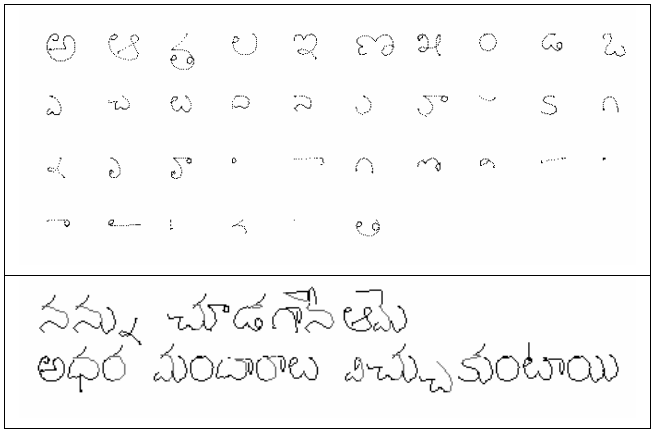


Figure 13: Samples and Output for Telugu (User TE02)

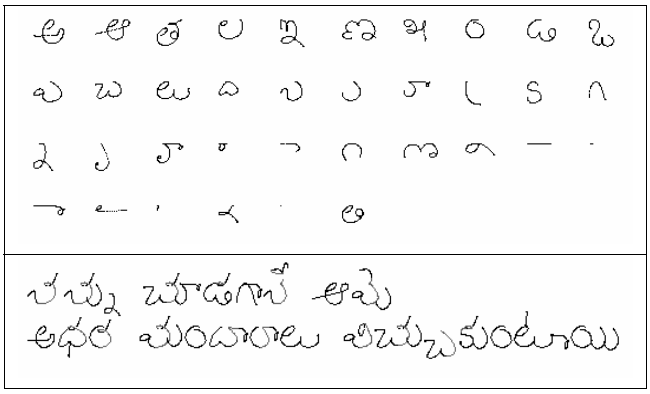


Figure 14: Samples and Output for Telugu (User TE03)

5.1.2 Language/Script: Marathi/Devanagri

Input: `garaja barasa re pAvasa rokhuna sagaLa SvAsa`

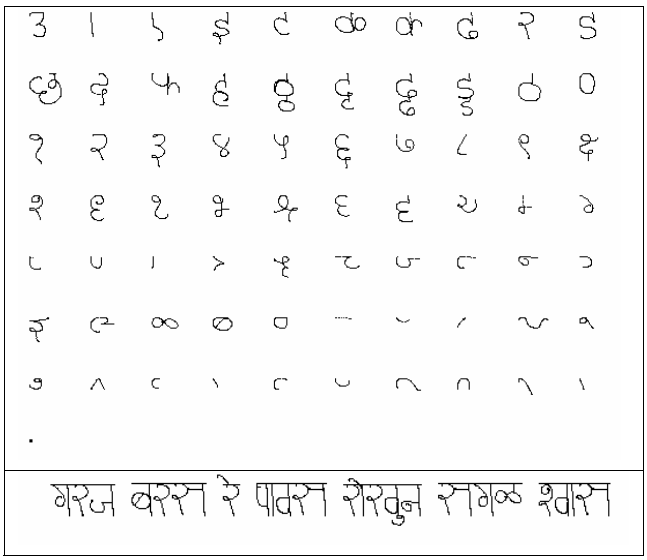
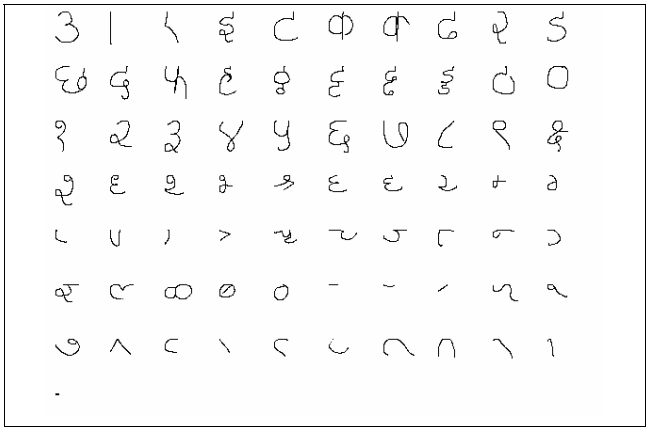


Figure 15: Samples and Output for Marathi (User MR01)



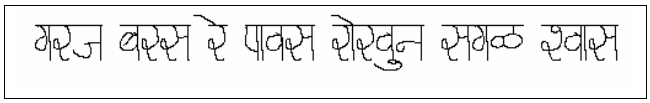


Figure 16: Samples and Output for Marathi (User MR02)

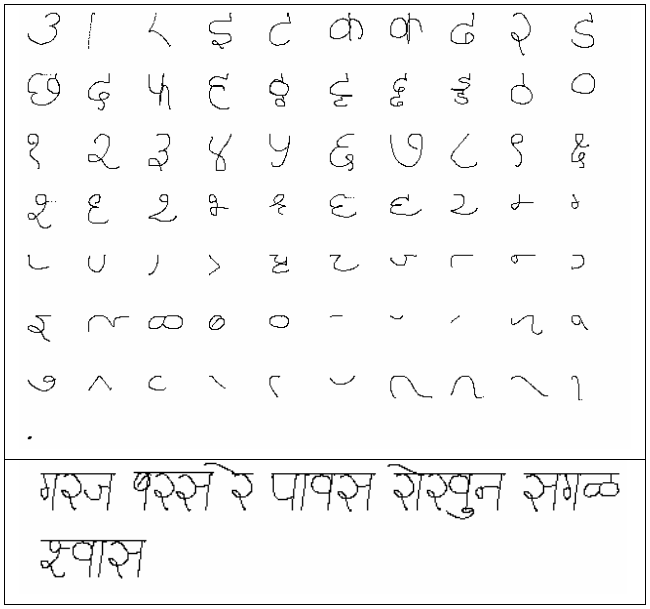


Figure 17: Samples and Output for Marathi (User MR03)

6. Conclusions

We proposed an architecture which enables exchanging SMS messages in a person's own handwriting in Indic script. We conceptualized and built a general framework that enables synthesis of any Indic script using a Roman input text. We further showed, for Telugu and Marathi how the proposed framework can be used for building a script synthesis engine which forms the core module of an SMS based personal messaging system for Indian languages. The above mentioned architecture and the framework helps in the localization efforts for Indian languages in the context of the fact that mobile has become a popular platform for people in India. Synthesis being an inverse problem of recognition, (the learning from) the proposed framework is being exploited by our group for Indic script recognition.

7. References

[1] H. Choi, S-J. Cho and J. H. Kim, Generation of Handwritten Characters with Bayesian network based On-line Handwriting Recognizers, Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR),2003

[2] P. B. Pal, Typesetting in Bengali script using TEX, TUGboat, Volume 23 No. 3/4, www.tug.org/TUGboat/Contents/contents23-3-4.html, 2002

[3] J. Wang, C. Wu1, Y-Q. Xu1, H-Y. Shum and L. Ji, Learning-based Cursive Handwriting Synthesis, Eighth

International Workshop on Frontiers in Handwriting Recognition (IWFHR), 2002

[4] L.V.S. Mukkavilli, TeluguTeX, www.cise.ufl.edu/~snsk/TeluguTex/TeluguTex.pdf, 1991

[5] I. Guyon, "Handwriting synthesis from handwritten glyphs", Proc. the Fifth International Workshop on Frontiers of Handwriting Recognition, Colchester, England, 1996.

[6] P.V.S. Rao and V. Ramasubramanian, Connected script synthesis by character concatenation – A Bezier curve formulation, Instt. Electronics & Telecom. Engrs., vol 37, 1991

[7] R. Ishida, An Introduction to Indic Scripts, http://people.w3.org/rishida/scripts/indic-overview/, 2003

[8] S. Ager, Omniglot Writing Systems and Languages of World, http://www.omniglot.com/writing/syllabic.htm, 1998

[9] J. Wells (contact), SAMPA computer readable phonetic alphabet, http://www.phon.ucl.ac.uk/home/sampa/home.htm, 2005

[10] J.M. Hollerbach, An Oscillation Theory of handwriting, http://dspace.mit.edu/handle/1721.1/6879, 1980

[11] Microsoft Typography, Developing OpenType Fonts for Indic, http://www.microsoft.com/typography/OpenType%20Dev/indic/intro.mspx, Scripts, 2001

[12] A. Chopde, ITRANS Encoding, http://www.aczoom.com/itrans/tblall/tblall.html, 2001

[13] R. Sproat, A Formal Computational Analysis of Indic Scripts, http://compling.ai.uiuc.edu/rws/newindex/indic.pdf

[14] C.V. Jawahar and A. Balasubramanian, Synthesis of Online Handwriting in Indian Languages, http://cvit.iiit.ac.in/papers/balu06OnlineSynthesis.pdf, 2006

[15] S.P. Mudur, N. Nayak, S. Shanbhag, R.K. Joshi, An Architecture for the Shaping of Indic Texts Computer and Graphics, Vol. 23, , No. 1, 1999

[16] CWIT, Indian Language SMS: Proposals for 3GPP Standards Update, Centre of Excellence in Wireless Technology, #152, CSD Building, ESB, IIT Madras Campus, Chennai, India, 2008

[17] D.N. Katre, Position Paper on "Cross-cultural Usability Issues of Bilingual (Hindi & English) Mobile Phones", Indo-Danish HCI Research Symposium, Department of Design, Indian Institute of Technology, Guwahati, India, 2006