DL Architecture for Indic Scripts

Suryaprakash Kompalli, Srirangaraj Setlur, and Venugopal Govindaraju

CEDAR,
UB Commons,
520 Lee Entrance, Suite 202,
Amherst, NY 14228
{kompalli,setlur,govind}@cedar.buffalo.edu
http://www.cedar.buffalo.edu/ilt

Abstract. In this study, we outline computational issues in the design of a Digital Library (DL) for Indic languages. The complicated character structure of Indic scripts entails novel OCR analysis techniques and user interface (UI) designs. This paper describes a multi-tier software architecture, which provides text and image processing tools as independent, reusable entities. Techniques for measuring and evaluating different stages of an Indic script recognition engine are outlined.

1 Introduction

DLs have made possible the global access of content, once limited by physical location, transportation and cooperation of artifact owners. Researchers have been showing interest in developing recognition and DL tools for Indic scripts [3, 5, 6, 9, 10, 17, 20]. Internationally acclaimed literary masterpieces, and centuries of cultural heritage lie hidden in these languages and scripts. However, this interest in OCR/DL systems for Indic scripts is not supported by a matching infrastructure of standardized data sets, or OCR analysis tools. Data access for non-Latin scripts like Indic is a frustrating experience for many users. In this paper, we examine design issues, and suggest techniques for developing Indic DL architectures. User-friendly data entry, multi-lingual data display, and storage are also outlined for Aryan and Dravidian scripts, with possible extensions for Arabic. Section 2 describes the characteristics of Indic scripts, and technological challenges posed by them. Section 3 demonstrates the architecture developed to support Indic DLs, with current and future applications suggested in section 4.

2 Indic Scripts

The Indian subcontinent has 18 scheduled (official) languages, which correspond to 11 scripts[1]. These scripts are divided into three categories: (i) Aryan: Devanagari, Bengali, Gujarati, Oriya, and Punjabi; (ii) Dravidian: Kannada, Malayalam, Tamil, and Telugu; and (iii) Arabic (Figure 1). All Indic scripts are inflective, i.e. elements from their alphabet set combine to form new characters by shape variation based on context.

[1] Source: *Central Institute of Indian Languages*

S. Marinai and A. Dengel (Eds.): DAS 2004, LNCS 3163, pp. 28–38, 2004.

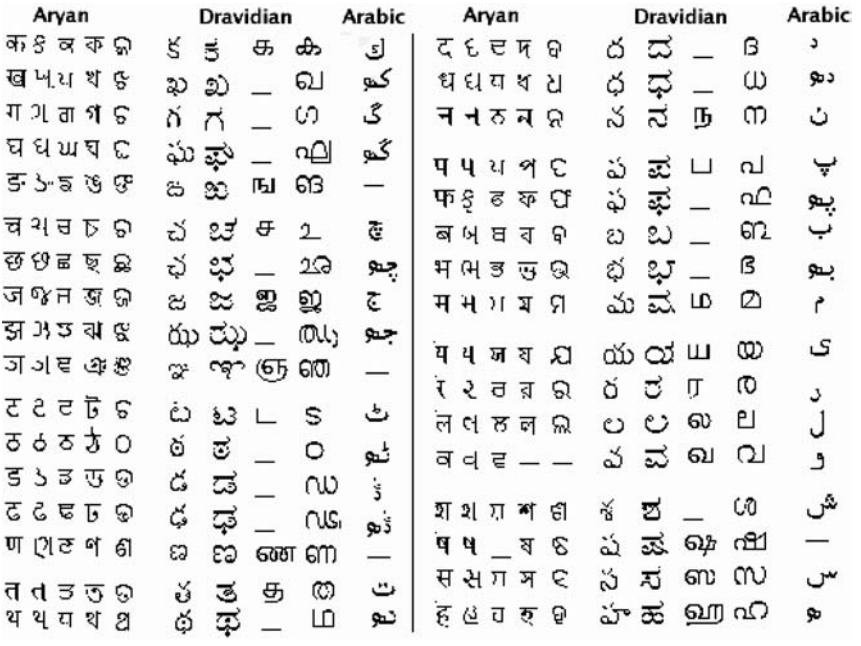


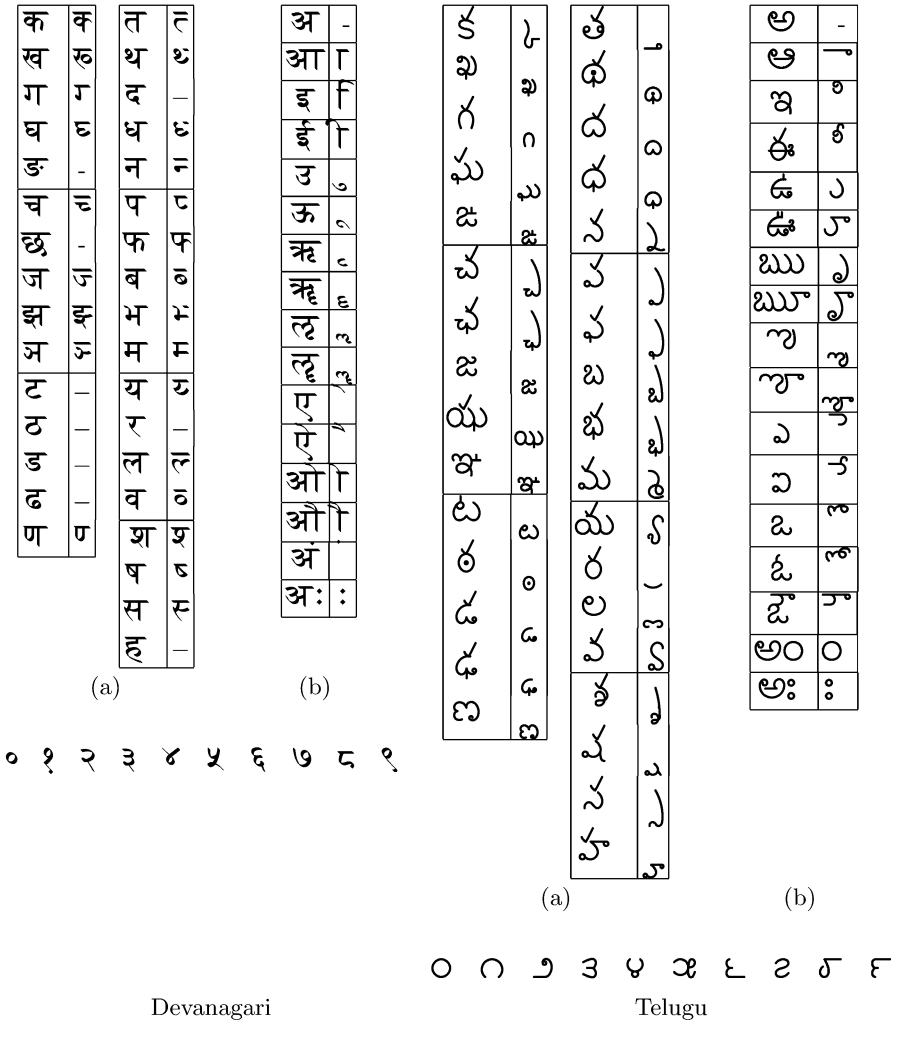
Fig. 1. Indic consonants: Aryan scripts - Devanagari, Bengali, Gujarati, Punjabi, and Oriya; Dravidian scripts - Tamil, Telugu, Kannada, and Malayalam, and Arabic script

Devanagari is a syllabic-alphabetic script derived from Brahmi, and provides written form to approximately forty eight languages including Hindi, Konkani, and Marathi [11, 13]. Each Devanagari character represents a *Syllable*: a unit of pronunciation that has a stop (coda) at the end. Forming *Syllabic* characters from *Alphabets* is a representative feature of *Syllabic-alphabetic* scripts, which include Devanagari, Telugu, Kannada, Bengali etc [13]. Devanagari is an inflectional script: its' base alphabet forms (Table 1) combine to give new characters. For example: The compound character kya (क्य) is made by combining forms of ka (क) and ya (य) (More examples in Table 2). Devanagari characters are joined together into words by a horizontal line, called the *Shirorekha*.

Telugu is a Syllabic-alphabetic script of Brahmic origin, and provides written form to three languages. In tune with other Syllabic-alphabetic scripts, Telugu characters are composed of syllable units ending with a stop, and contain conjunct alphabet forms. However, Telugu is a Dravidian script, and has a few distinctive characteristics: there is no *Shirorekha* connecting the characters of a word and the consonant half-forms of compound characters do not touch each other (Table 1).

The Devanagari and Telugu alphabet set (Table 1) shows that many consonants and vowels have half-forms and vowel modifiers. A Devanagari/ Telugu word (any Indic script word) can be split into characters (syllables), which are themselves a combination of consonants, vowels, half-forms, modifiers and special symbols. Using the alphabet set, and rules for syllabic-alphabetic

Table 1. Devanagari and Telugu character set (a): Consonants (left) and their half-forms (right), (b): Vowels (l) and Vowel modifiers (r), bottom: Numerals. Note: Special character forms are not shown



character formation, words are parsed into components using production rules called script writing grammar (Examples in Table 2).

2.1 Challenges and Solutions for Indic DL

Our initial goal for Indic DL has been to enable semi-automatic truthing[2] of Indic documents, create public domain OCR data sets, design tools for word and sub-

[2] *Truthing* refers to the collection of ground-truth data for training or testing learning algorithms.

Table 2. Top: Hierarchical splitting of characters into glyphs and components. Bottom: Separation of a few Devanagari words. The characters ग्या ज्यो स्तु री క్ష కొ are samples of *compound characters*, formed by combining one or more consonant(s) with a vowel modifier. Some characters (री) have strokes on the top, called *ascenders* and others (स्तु) have strokes at the bottom, called *descenders*

English	Devanagari			Telugu			Description
	Character	Glyphs	Component	Character	Glyphs	Component	
ba	ब	ब	ब	బ	బ	బ	components and
kha	ख	ख	ख	ఖ	ఖ	ఖ	characters are identical
ga	ग	ग ा	ग ा	గ	గ ా	గ ా	different character and glyph
kya	क्य	क य	क य	క్ష	క ్ష	క ్ష	similar glyph and component
kau	कौ	क ौ	क ◌ ा	కొ	క ొ	క ◌ ొ	

Word	Character	Glyph	Component
ग्यान	ग्या ⋈ न	ग + य + ◌ ा ⋈ न	ग + य + ◌ ा ⋈ न
ज्योती	ज्यो ⋈ ती	ज + य + ◌ ो ⋈ त + ◌ ी	ज + य + ◌ ् + ◌ ा ⋈ त + ◌ ि + ◌ ी
कस्तुरी	क ⋈ स्तु ⋈ री	क ⋈ स + त + ◌ ु ⋈ र + ◌ ी	क ⋈ स + त + ◌ ु ⋈ र + ◌ ि + ◌ ी

word querying, and provide multi-lingual annotations. Challenges encountered in the creation of these resources are outlined in this section.

Text Encoding: Though Indic scripts have a small alphabet set, each language has a multitude of character forms, resulting in complex shapes on digital output. A number of independent bodies have devised encoding schemes, with each providing a set of codes for the alphabet set, and a corresponding set of parsing rules to create character forms from sequences of alphabet codes [11, 21]. Our system uses Unicode[3], which provides a hexadecimal encoding scheme for all major scripts. Unicode has rules to generate inflectional characters with variations like compound forms, ascenders, descenders etc. Alternate schemes do not have the compatibility and global acceptance of Unicode [1, 8, 19, 22].

Multi-lingual Text Input: All modified Indic character forms cannot be accommodated on a QWERTY keyboard: Typing Indic script data necessitates a combination of keys to enter the entire repertoire of characters. The INSCRIPT standard [21] maps keys of a *QWERTY* keyboard onto Indic alphabets. Control keys (*alt, shift*) are used to input corresponding modified forms. Alphabets are grouped in phonetic order, with the keyboard layout designed for enhanced speed. Another technique, transliteration represents characters of an inflective script by phonetic equivalent Latin strings. For example: The Devanagari (Telugu) character क (క) would be entered using the Latin equivalent *ka* (Figure 4.1.a). Both systems rely on phonetic characteristics of Indic scripts.

[3] Unicode is an official implementation of ISO/IEC 10646.

Barring a few exceptions, different Indic scripts have similar INSCRIPT layouts and transliteration schemes. Tests conducted using 14,225 words from 34 documents reveal that on an average, the INSCRIPT layout needs 13% fewer key presses than transliteration. However, INSCRIPT keyboards have to be physically labeled before a user can easily associate keys with relevant characters. Typing multi-lingual documents on an unlabelled keyboard entails a steep learning curve. We use the transliteration scheme[4] for multi-lingual data input. A GUI keyboard is also provided to enter special characters (Figure 4.1.b).

Platform Independent Representation: Indic script data is stored in XML, a versatile and preferred scheme for DL projects [1, 2, 8]. The platform independent representation allows other researchers to reuse our data, and design in-house parsers for information extraction. The implementation provides converters to generate HTML content from XML truth files.

Multi-lingual Text Display, Annotation and Meta-data Addition: Unicode fonts and Java style attributes enable multi-lingual text/annotation display within the same text window (Figure 4.1.c). The system provides controls to select sections of transliteration text, and specify it's target script. Extensions to XML Data Type Definition provide the flexibility to include additional meta-data like author information, genre etc in truth files. The current implementation supports Arabic, Devanagari, English and Telugu.

Image Processing: Given that images are an important information source for DL systems, we incorporate data extraction from gray-scale documents. The system supports text-image separation, word boundaries estimation and OCR of Devanagari documents [14]. Applications for image analysis and querying are described in later sections of this paper.

3 DL Design

DL projects often reveal a combination of UI design, file systems and image processing tools. Kanungo et al. integrate page segmentation analysis [18] with a visualization toolkit in TrueViz [16]. Allen et al. [2] use a combination of OCR and meta-data extraction to generate XML representations of newspapers. ATLAS [7] outlines an annotation graph model for linguistic material (text, image and video). Couasnon et al. use an automated web-based system for collecting annotations of French archives [12].

These tools are oriented for use in a specific domain (OCR, Document Layout, Newspaper DL), or are targeted towards non-Inflective scripts similar to Latin. In our DL, the features outlined in section 2.1 are implemented as modular solutions independent from target applications. This section outlines the grouping of these features into three modules that are being used as building blocks in different applications.

[4] We use ITRANS transliteration. URL:http://www.aczone.com/itrans/

UI, Image and File Processing (UIP): The UIP contains routines that process image and truth files, convert files from one format to another, and create the UI. In effect, this module encapsulates Platform Independent Representation, and Image Processing operations of the DL.

Text Entry and Display (TexED): The TexED controls Text Encoding, Multi-lingual Text Input and Multi-lingual Display mechanisms outlined in Section 2.1. TexED's functional mapping can be shown as:

$$f_{TexED}(transliteration\_text, GUI\_keyboard\_input, language) \Rightarrow (transliteration\_text, Indic\_script, system\_messages).$$

The domain (transliteration text, GUI keyboard input, and target Indic script) and output (transliteration text, target Indic script generated by parsing input transliteration, user prompts) of this mapping is defined by TexEd. It is possible to design multiple applications interfaced with a common TexED. For example, the OCR engine and annotation tool described in Section 4 use the same TexED for providing text input and display. The TexED currently supports Devanagari, Telugu and English.

Text Processing (TexP): Routines in the TexP unit are used to test linguistic post-processing algorithms. Currently, character and word n-gram calculation is implemented in the TexP unit.

4 Applications

Several applications have been developed using the UIP, TexED and TexP outlined in section 3. We continuously upgrade the DL design based on research inputs and have listed a few immediate concerns in this section.

4.1 Data Collection for Inflective Scripts

This architecture supports data collection for Indic script OCR [15]. Indic OCR necessitates splitting words into smaller parts, though opinions vary on the level of detail (Table 2 shows different levels of separation). Statistical [5, 14, 17] techniques extract representative feature vectors from character images and apply pattern classification techniques on these features. To reduce the number of classes, structural information is used to split each character into smaller components; leading to higher granularity. Structural techniques [9, 10] classify characters on the basis of geometric attributes, relying on coarse characteristics at first, and classifying sub-groups using detail features. In this case, any loss of structural information is avoided by retaining the entire character (leading to coarse granularity). Providing data sets with different levels of character splitting allows robust testing of different OCRs: Structural classifiers could use words and characters, which have not been segmented into smaller parts; whereas statistical techniques may be tested with alphabet and component images.

Our data sets are available as XML files containing links to the document image (300 dpi, gray-scale, tiff files), bounding-box and Unicode values of

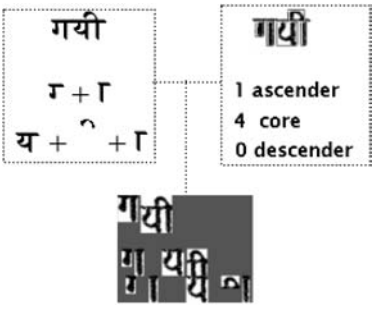


Fig. 2. Parsing a word on query: Extracting component truth (Left), Extracting component boundaries (right), Combining the two results to obtain component images (bottom)

words, and linguistic data. A Devanagari query system has been provided to extract character images at multiple levels of granularities (Figure 3.2). The query system parses each Unicode word according to the script writing grammar (Table 2), identifies each component, and stores them as component truth. Character segmentation algorithms [14] are used to compute the list of component boundaries in each word image. Component truth and component boundaries are combined (Figure 2) to obtain results of each query. The query system allows users to prune results, extract features, store the images (query results) into a directory, or pass extracted components to a recognizer. Recognition results are displayed on the query panel using color codes.

To ensure data accuracy, we use two verification steps: we examine the truth value and OCR segmentation results of each word in a single interface, and flag truthing or segmentation errors(Figure 3.1). On an average, each truth file contains 445 words and needs 22 minutes for verification. Creation of initial truth is more expensive and involves the typing of transliteration text with occasional correction of word boundaries (average 69 minutes for a 400 word document). Verification of 1800 words revealed 2 truthing errors(less than 0.01%). Words flagged in the verification process are observed in a tree view structure to identify reasons for segmentation failure and correct truthing errors, if any. Currently, we have around 50,000 machine printed Devanagari word images and plan to add another 150,000 words. We are also gathering handwritten Arabic documents for a database of handwritten words. The data and associated querying tools are available for free download from our web-site[5].

4.2 OCR Analysis

The Devanagari query system is integrated with a tree view for OCR analysis (Figure 3.2, 3.3). Errors in query output indicate possible failures of the segmentation algorithm. If reasons for an error are not apparent, the tree

[5] URL: http://www.cedar.buffalo.edu/ilt

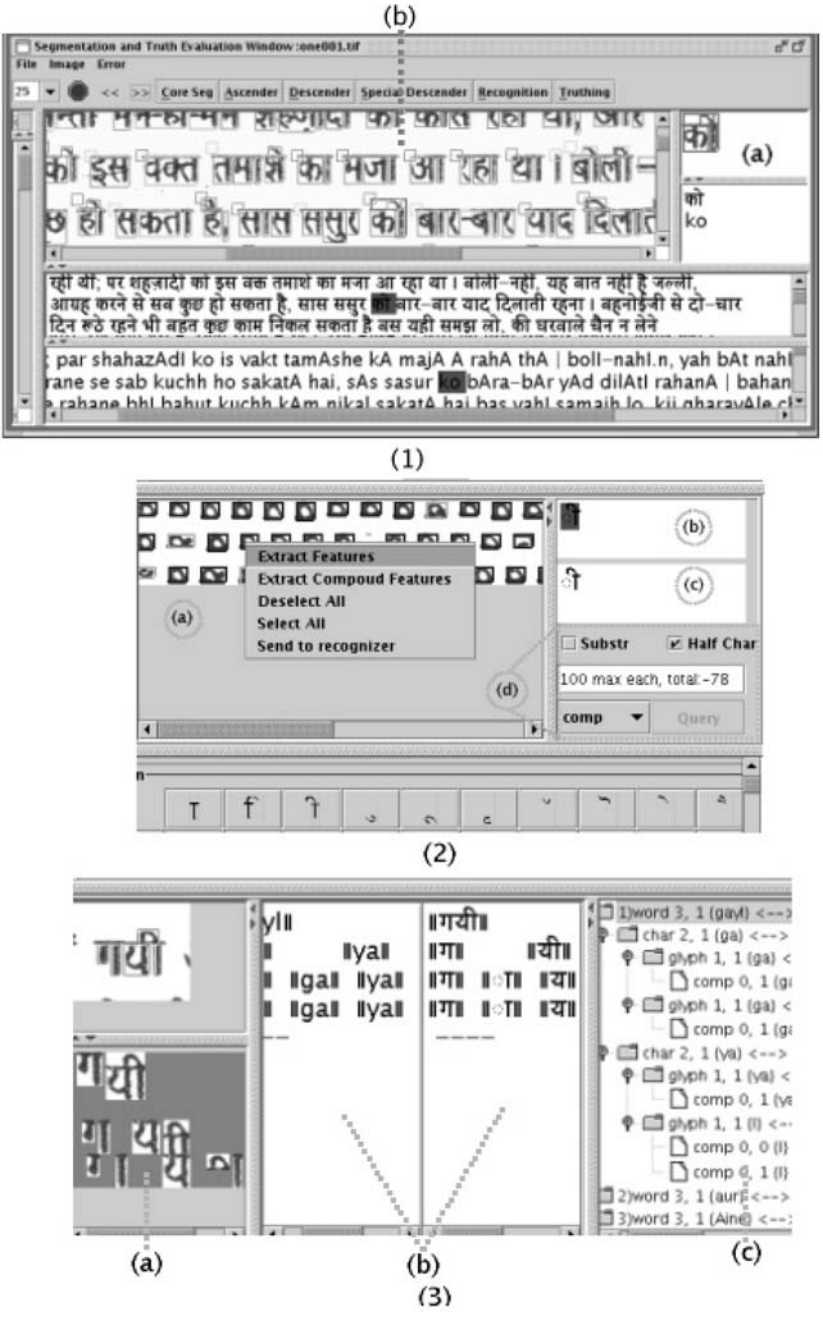
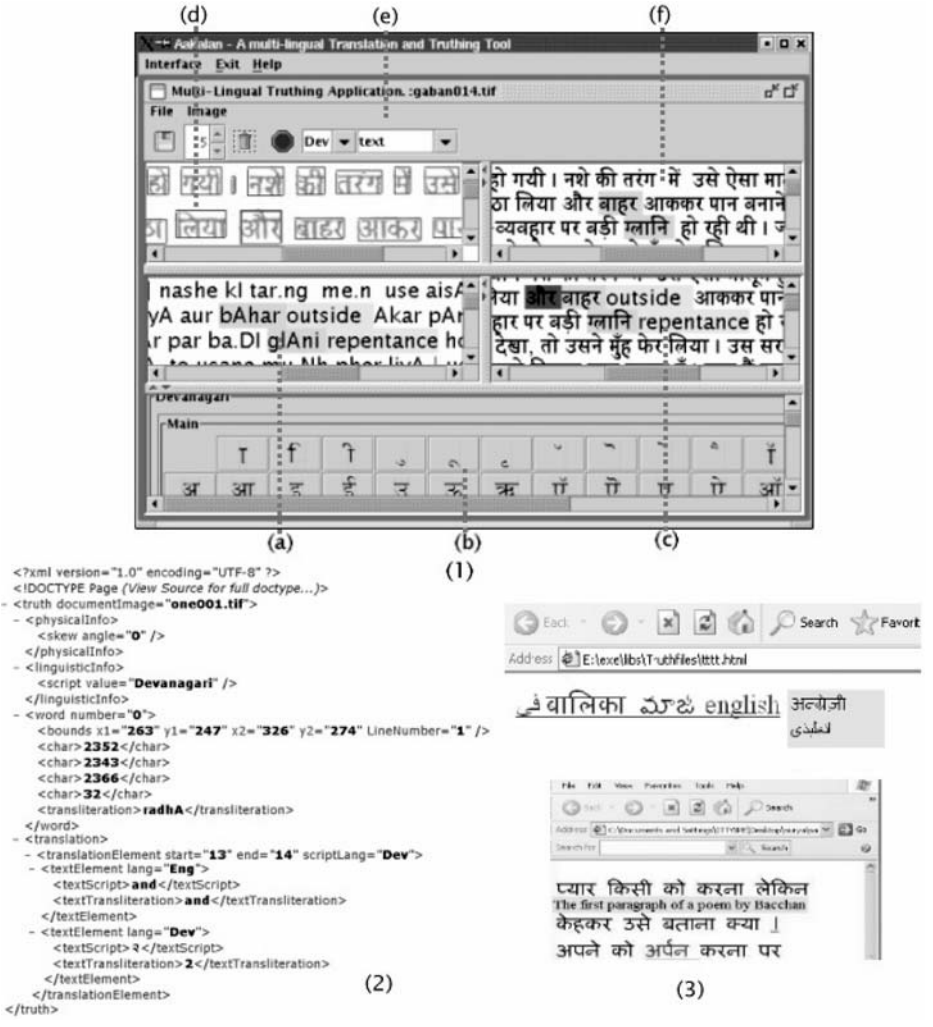


Fig. 3. 1: Segmentation and truth evaluation; (a) Word truth and segmentation results, (b) Document view. 2: Query Panel; (a): Query results, (b,c): Transliteration and display for query input, (d): Query selection panel. 3: Script Analysis window; (a): Breakup of image into components, (b): Breakup of image transliteration and truth, (c): Tree view of document




```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE Page (View Source for full doctype...)>
- <truth documentImage="one001.tif">
  - <physicalInfo>
      <skew angle="0" />
    </physicalInfo>
  - <linguisticInfo>
      <script value="Devanagari" />
    </linguisticInfo>
  - <word number="0">
      <bounds x1="263" y1="247" x2="326" y2="274" LineNumber="1" />
      <char>2352</char>
      <char>2343</char>
      <char>2366</char>
      <char>32</char>
      <transliteration>radhA</transliteration>
    </word>
  - <translation>
    - <translationElement start="13" end="14" scriptLang="Dev">
      - <textElement lang="Eng">
          <textScript>and</textScript>
          <textTransliteration>and</textTransliteration>
        </textElement>
      - <textElement lang="Dev">
          <textScript>२</textScript>
          <textTransliteration>2</textTransliteration>
        </textElement>
      </translationElement>
</truth>
```

Fig. 4. 1: Truthing and annotation interface; (a):Transliteration input, (b): GUI keyboard, (c): Annotation and script words, (d): Image with word boundaries, (e): Tool-bar for language selection, (f): Document Truth. 2: XML Truth of a document. 3: Annotations represented in HTML

structure is examined for a detailed view of word segmentation. This combination of a tree view and a query system allows users to test both segmentation and recognition results. Provision is being made to plug-in and test user-defined segmentation and recognition algorithms.

4.3 Annotation

Using the TexED and UIP modules, we have designed an interface to annotate passages of text in one language with meaning/description in other languages.

The UI allows users to input multi-lingual data with a common tool, and an unmodified QWERTY keyboard. Text and annotations are rendered in appropriate scripts, providing users prompt and natural feedback. Annotated documents can be saved as HTML with annotations being displayed when the mouse hovers over relevant text (Figure 4.3).

Annotation features of the tool are being used by the LiTgloss project[6] at the University at Buffalo. The project provides a collection of literary or cultural masterpieces, written in languages other than English, and annotated to facilitate comprehension by English-speaking readers.

5 Discussion and Future Work

This work examines prevalent text entry and display techniques and outlines an optimal use of technology to support Indic script DLs. We provide multi-granular data sets for Devanagari OCR analysis. Indic scripts have similar script writing grammars, and efforts are on to design OCR evaluation and querying systems for Telugu. Given the inflective nature of Indic scripts, it is likely that certain OCR errors (recognition) affect the "readability" of an output document to a lesser degree than segmentation errors. Suitable error metrics for these variations are being explored.

In future versions of our dataset, we plan to include information like document skew, document degradation parameters etc with our OCR data set. Analyzing artificial degradation algorithms used in Latin script documents [4] and developing similar techniques for inflective scripts is also being considered.

Acknowledgments

The authors would like to thank Professor Maureen Jameson for her interest in the annotation project. We also thank Swapnil Khedekar, Ramanaprasad Vemulapati, Faisal Farooq and Sumit Manocha for their invaluable suggestions.

References

1. The xml version of the tei guidelines. 24th February, 2004. URL: http://www.tei-c.org/P4X/CH.html.
2. R. B. Allen and J. Schalow. Metadata and data structures for the historical newspaper digital library. In *Proceedings of the 8th international conference on Information and knowledge management*, pages 147–153, 1999.
3. T. Ashwin and P. Sastry. A font and size independent ocr system for printed kannada documents using support vector machines. *Sadhana*, 27:35–58, February 2002.
4. H. Baird and T. K. Ho. Large-scale simulation studies in image pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(10):1067–1079, 1997.

[6] LiTgloss: http://wings.buffalo.edu/litgloss/

5. V. Bansal. Integrating knowledge sources in devanagari text recognition. *IEEE Transactions on Systems, Man and Cybernetics Part A*, 30(4):500–505, 2000.
6. I. Bazzi, R. Schwartz, and J. Makhoul. An omnifont open-vocabulary ocr system for english and arabic. *IEEE Pattern Analysis and Machine Intelligence*, 21(6):495–504, June 1999.
7. S. Bird, D. Day, J. Garofolo, J. Henderson, C. Laprun, and M. Liberman. Atlas: A flexible and extensible architecture for linguistic annotation. In *Proceedings of the Second International Language Resources and Evaluation Conference*, pages 1699–1706, 2000.
8. T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible markup language (xml) 1.0 (second edition), 2001.
9. B. Chaudhuri and U. Pal. An ocr system to read two indian language scripts: Bangla and devanagari. In *Proceedings of the 4th International Conference on Document Analysis and Recognition*, pages 1011–1015, 1997.
10. B. Chaudhuri, U. Pal, and M. Mitra. Automatic recognition of printed oriya script. In *Proceedings of the 6th International Conference on Document Analysis and Recognition*, pages 795–799, 2001.
11. U. Consortium. *The Unicode Standard Version 4.0*. Addison-Wesley, 2003.
12. B. Couasnon, J. Camillerapp, and I. Leplumey. Making handwritten archives documents accessible to public with a generic system of document image analysis. In *Proceedings of the 1st International Workshop on Document Image Analysis for Libraries. (DIAL 2004)*, pages 270–277, 2004.
13. P. T. Daniels and W. Bright. *The World's Writing Systems*. Oxford University Press, March 1996.
14. V. Govindaraju, S. Khedekar, S. Kompalli, F. Farooq, S. Setlur, and V. Prasad. Tools for enabling digital access to multilingual indic documents. In *Proceedings of the 1st International Workshop on Document Image Analysis for Libraries. (DIAL 2004)*, pages 122–133, 2004.
15. S. Kompalli, S. Setlur, V. Govindaraju, and R. Vemulapati. Creation of data resources and design of an evaluation test bed for devanagari script recognition. In *Proceedings of the 13th International Workshop on Research Issues on Data Engineering: Multi-lingual Information Management*, pages 55–61, 2003.
16. C. Lee and T. Kanungo. The architecture of trueviz:a groundtruth/metadata editing and visualizing toolkit. *PR*, 36(3):811–825, March 2003.
17. H. Ma and D. Doermann. Adaptive hindi ocr using generalized hausdorff image comparison. *ACM Transactions on Asian Language Information Processing*, 26(2):198–213, 2003.
18. S. Mao and T. Kanungo. Software architecture of pset: A page segmentation evaluation toolkit. *International Journal on Document Analysis and Recognition (IJDAR)*, 4(3):205–217, 2002.
19. C. Microsoft. Windows glyph processing. Date: 24th, URL: http://www.microsoft.com/typography/developers/opentype/default.htm, February 2004.
20. A. Negi, C. Bhagvati, and B. Krishna. An ocr system for telugu. In *Proceedings of the 6th International Conference on Document Analysis and Recognition*, pages 1110–1114, 2001.
21. B. of Indian Standards. Indian script code for information interchange, 1999.
22. I. Sun Microsystems. Solaris 9 operating system features and benefits - compatibility. Date: 24th, URL: http://wwws.sun.com/software/solaris/sparc/solaris9_features_compatibility.html, February 2004.