

FREEHAND SKETCH RECOGNITION FOR COMPUTER-ASSISTED
LANGUAGE LEARNING OF WRITTEN EAST ASIAN LANGUAGES

A Thesis

by

PAUL PIULA TAELE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2010

Major Subject: Computer Science

FREEHAND SKETCH RECOGNITION FOR COMPUTER-ASSISTED
LANGUAGE LEARNING OF WRITTEN EAST ASIAN LANGUAGES

A Thesis

by

PAUL PIULA TAELE

Submitted to the office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee, Tracy Hammond
Committee Members, Yoonsuck Choe
Jun Kameoka
Head of Department, Valerie Taylor

December 2010

Major Subject: Computer Science

ABSTRACT

Freehand Sketch Recognition for Computer-Assisted
Language Learning of Written East Asian Languages. (December 2010)

Paul Piula Tael, B.S., The University of Texas at Austin

Chair of Advisory Committee: Dr. Tracy Hammond

One of the challenges students face in studying an East Asian (EA) language (e.g., Chinese, Japanese, and Korean) as a second language is mastering their selected language's written component. This is especially true for students with native fluency of English and deficient written fluency of another EA language. In order to alleviate the steep learning curve inherent in the properties of EA languages' complicated writing scripts, language instructors conventionally introduce various written techniques such as stroke order and direction to allow students to study writing scripts in a systematic fashion. Yet, despite the advantages gained from written technique instruction, the physical presence of the language instructor in conventional instruction is still highly desirable during the learning process; not only does it allow instructors to offer valuable real-time critique and feedback interaction on students' writings, but it also allows instructors to correct students' bad writing habits that would impede mastery of the written language if not caught early in the learning process.

The current generation of computer-assisted language learning (CALL) applications specific to written EA languages have therefore strived to incorporate writing-capable modalities in order to allow students to emulate their studies outside the

classroom setting. Several factors such as constrained writing styles, and weak feedback and assessment capabilities limit these existing applications and their employed techniques from closely mimicking the benefits that language instructors continue to offer. In this thesis, I describe my geometric-based sketch recognition approach to several writing scripts in the EA languages while addressing the issues that plague existing CALL applications and the handwriting recognition techniques that they utilize. The approach takes advantage of A Language to Describe, Display, and Editing in Sketch Recognition (LADDER) framework to provide users with valuable feedback and assessment that not only recognizes the visual correctness of students' written EA Language writings, but also critiques the technical correctness of their stroke order and direction. Furthermore, my approach provides recognition independent of writing style that allows students to learn with natural writing through size- and amount-independence, thus bridging the gap between beginner applications that only recognize single-square input and expert tools that lack written technique critique.

DEDICATION

I would like to dedicate this thesis to the members of my loving family: my mother Tiana for her valuable support, my father Tuputala for continually rooting for me, my sister Tina for always making me laugh, my brother Ne'e for his brotherly love, and my grandmother Sose for her immense kindness and for always doing my laundry.

ACKNOWLEDGEMENTS

My greatest thanks to the members of the Sketch Recognition Lab for their continued support and help in the research work covered in this thesis. This thesis would not have been possible without their support. In addition, I would like to give extra thanks to my advisor Dr. Tracy Hammond, as well as to my committee members Dr. Yoonsuck Choe and Dr. Jun Kameoka for their valuable sage advice.

TABLE OF CONTENTS

	Page
ABSTRACT	iii
DEDICATION.....	v
ACKNOWLEDGEMENTS	vi
TABLE OF CONTENTS	vii
LIST OF FIGURES	ix
LIST OF TABLES.....	xii
1. INTRODUCTION	1
2. TRADITIONAL INSTRUCTIONAL METHODS	4
2.1 Difficulties in Learning the Written Component	4
2.2 Stroke Order and Direction Instruction	6
2.3 Limitations of Traditional Instruction for the Written Component.....	7
3. RELATED TECHNIQUES AND SYSTEMS	9
3.1 Neural Networks	10
3.2 Hidden Markov Models.....	11
3.3 Previous Works in Computer-Assisted Language Learning.....	12
3.4 Corner-Finding Algorithms	17
3.5 LADDER Sketching Language.....	19
4. OVERVIEW OF VISUAL STRUCTURE ASSESSMENT.....	22
4.1 Relevant LADDER Specifications.....	25
4.2 Components	25

	Page
4.3 Constraints	30
4.4 Aliases	34
5. METHODOLOGY OF CONSTRUCTING SHAPE DEFINITIONS	38
5.1 Single Simple Shape.....	39
5.2 Single Compound Shape	43
5.3 Multiple Shapes.....	44
5.4 Handling Special Cases	47
6. OVERVIEW OF WRITTEN TECHNIQUE ASSESSMENT	54
6.1 Limitations of Assessing Written Technique in LADDER	54
6.2 Strokes and Primitive Shapes	56
6.3 Customizing Aliases for Simple Symbols	57
6.4 Customizing Aliases for Compound Symbols.....	59
6.5 Assessing the Written Technique Using Aliases	61
7. IMPLEMENTED APPLICATIONS AND EVALUATION	65
7.1 Hashigo: A CALL System for Handwritten Japanese Kanji.....	65
7.2 LAMPS: A CALL System for Handwritten Mandarin Phonetic Symbols I	71
8. SUMMARY.....	77
8.1 Expanding on This Methodology.....	78
REFERENCES	80
VITA	84

LIST OF FIGURES

	Page
Figure 1. The three primitive geometric components used in the methodology for written EA languages: (a) lines, (b) curves, and (c) ellipses.	27
Figure 2. Examples of the utilized geometric components: (a) primitives, (b) simple components, and (c) compound components.....	27
Figure 3. A side-by-side comparison of the original component labels and the corresponding aliases for an example Chinese character: (a) the original component labels, and (b) the aliases.	35
Figure 4. A shape description for the Chinese character <i>ten</i>	41
Figure 5. A shape description for the Chinese character <i>mouth</i>	42
Figure 6. A shape description for the Chinese character <i>ancient</i>	44
Figure 7. A shape description for the Chinese characters <i>Japan</i>	47
Figure 8. The three single straight line symbols in MPS1: (left) The symbol for the phonetic <i>i</i> sound, (middle) the symbol for the rising tone, (right) the symbol for the falling tone.....	49
Figure 9. The symbol for the phonetic <i>i</i> sound physically contained within the symbol for the phonetic <i>f</i> sound in MPS1.	50
Figure 10. A visual representation of the modified MPS1 methodology for the case of the phonetic <i>f</i> sound symbol.	51
Figure 11. The circular shaped <i>ieung</i> symbol in an example hangul letter.....	52
Figure 12. The bounds of an ellipse, including the highlighted bounding points that serve as default aliases for ellipse components in LADDER.	53

Figure 13. Label comparisons for the Chinese character <i>mouth</i> : (a) Enumerated labels for temporal order of strokes in conventional EA language instruction, and (b) enumerated aliases for temporal order of primitive line components for the methodology.	57
Figure 14. Partial shape description for the Chinese character <i>ten</i> which focuses primarily on the components and also the aliases related to stroke order and direction.	59
Figure 15. Stroke order and direction labels for a compound and two simple symbols: (a) Stroke order and stroke direction labels for the Chinese character <i>ten</i> , (b) stroke order and partial stroke direction labels for the Chinese character <i>mouth</i> , and (c) stroke order labels for the Chinese character <i>ancient</i>	60
Figure 16. An overview of the Hashigo GUI, incorporating four of the key features in the methodology: free-sketch input, paper-like interface, digital capabilities, and emulated teacher feedback.	66
Figure 17. The Hashigo selection window for choosing the lesson type and kanji or element set.	66
Figure 18. The instruction window for the Learn Mode in Hashigo for an individual symbol.	67
Figure 19. The result window for the Review mode in Hashigo for an individual symbol.	68
Figure 20. The final progress report window after completing a lesson in Hashigo.	69
Figure 21. A screenshot of LAMPS during regular operation.	74
Figure 22. A screenshot of LAMPS when the user writes a symbol in a visually correct sequence of symbols with an incorrect written technique: (a) drawing panel, (b) buttons to run assessment and clear panel, (c) prompt with next MPS1 symbol to draw, and (d) result.	75

Figure 23. Samples of user-sketched MPS1 symbols accurately recognized as having: (a) incorrect visual structure and (b) correct visual structure with incorrect written technique..... 76

LIST OF TABLES

	Page
Table 1. Ideal shapes for the LADDER sketching language and the challenges for written symbols in EA languages.....	23
Table 2. Orientation Constraints: checks whether a line is a slope or an anti-diagonal; they are always unary.	31
Table 3. Point Relationship Constraints: Compares the center, endpoint, or bounding position of two shapes.	31
Table 4. Position Constraints: Compares the position of two shapes relative to each other.....	32
Table 5. Proximity Constraints: Checks for the closeness proximity of one shape to another by some relative threshold value.	33
Table 6. Length Constraints: Compares the length of two lines relative to each other...	33
Table 7. Logical Constraint: Involves negating a constraint or operating disjunction on two constraints. By default, all constraints in LADDER are mutually conjunctive.....	33
Table 8. Comparisons between naming schemes for original labels and aliases.	34
Table 9. Accommodating multiple feedbacks using solely LADDER and its <i>drawOrder</i> constraint for a single symbol.	56
Table 10. Assessing the written technique of three different writing styles for the Chinese character <i>ten</i> based on their custom aliases for stroke order and direction.	64

1. INTRODUCTION

It should not come as a surprise that the English language differs much more greatly from EA languages such as Chinese (e.g., Mandarin, Cantonese), Japanese, and Korean than from other European languages such as Spanish, French, and German. In fact, this very sentiment is shared by the United States' federal government, which reports that for native English users learning a foreign language, it takes up to three times longer to reach proficiency for an EA language compared to a European language [1]. This holds especially true for the written component of EA languages, where the reading and writing of the more complicated writing scripts is “a labor-intensive endeavor” that requires that language students with native English fluency expend significant amounts of “time, patience, discipline and perseverance” to achieve native fluency [1].

In order to help students overcome the difficulties in studying written EA languages, language programs traditionally introduce various written techniques in the form of stroke order and direction as a way to ease the learning process and to provide a more systematic way for students to master their language of study's associated writing scripts [2, 3, 4, 5]. Furthermore, written technique instruction is greatly stressed early on in the learning process for these writing scripts in order to discourage the development of bad writing habits [6]; without correcting these bad writing habits early on, not only can they become more difficult to correct later in their language studies, but they can also impede the pace of their studies in the long term. Despite the advantages gained

This thesis follows the style of *IEEE Transactions on Visualization and Computer Graphics*.

through teaching written EA languages through written technique instruction, a limiting factor in how it is presently taught is that it requires that teachers actively participate in monitoring the students' writing in order to provide written technique assessment; simply evaluating the final result of students' writings in the course greatly restricts language instructors to evaluating the correctness of the writings' visual structure (i.e., appearance).

This thesis describes a freehand sketch recognition approach for use in computer-assisted language learning (CALL) applications specific to teaching written EA languages. The approach enables CALL applications to allow students to obtain the kind of feedback on their visual structure and written technique that human language instructors naturally provide, therefore not only allowing students to emulate the type of writing study practices found in EA language courses, but also freeing instructors to devote additional time on other equally important aspects of the languages of study. Furthermore, the approach described in this thesis is not constrained in terms of size and amount; that is, students' are not required to adhere to either writing in a restricted space or using only single symbol-input for evaluation. As a result, the primary contributions of this thesis are:

- Automated feedback and assessment of students' visual structure and written technique: students can receive automated feedback and assessment on their visual structure and written technique of their handwriting for written EA language scripts, much like what human language instructors already offer;

- Freehand writing environment: students can maintain a natural freehand writing environment that allows them to write multiple symbols without restriction on the size of those symbols, much like how writing is done naturally done on paper.

2. TRADITIONAL INSTRUCTIONAL METHODS

In learning written EA languages, practice through writing the symbols associated with the language of study is an essential condition. Furthermore, not only is writing practice in itself a prerequisite in the learning process, but it is also of vital importance for other reasons: through writing practice of their language of study, students are able to “improve the aesthetic appearance of their writing and acquire a ‘natural feel’ for the flow” of the symbols in those scripts “that cannot be achieved simply by remembering them” [7]. For this reason, the language curriculum guides students to initially hone their skills in writing the symbols using grid sheets [2, 3, 5, 8, 9], which are sheets of paper typically ruled into squares of an inch or so on each side for students to practice writing the individual symbols [9]. In addition to grid sheets permitting students to rehearse their writing of the symbols in an orderly fashion, they can also provide students with an opportunity to perfect the proportions of the symbols in a model square space before moving on to writing in a more natural writing environment. This is because the inherent written properties of EA languages demand that the symbols, regardless of their simplicity or complexity, should be written so that they occupy a consistent amount of square space [9].

2.1 Difficulties in Learning the Written Component

Although the act of writing plays an integral role in students learning the symbols of their language of study, merely having language students brute force their way through repetitious writing in order to master the written component is unrealistic within

a typical American classroom setting; unlike EA learners who spend countless hours practicing the writing for primary language acquisition as their first language, students with a native English background and a lack of mastery in an EA language distinct from their language of study do not have such a luxury [1]. Furthermore, solely relying on instruction of the written component through rote memorization is insufficient, as language instructors of EA languages have come to understand that requiring students to memorize excessive amounts of symbols in order to achieve written fluency is an unreasonable expectation [1]. In fact, the primary obstacles that challenge EA language students with native English fluency in learning the written component – especially for the more complex writing scripts of EA languages – include:

- vast symbol sets that can number in the thousands,
- complicated visual structures involving a numerous range of strokes that can exceed thirty,
- a high similarity between symbols within the writing script that can cause “shape collisions” during the memorization process, and
- a wide variation in visual appearance due to divergent writing styles [10].

It is because of the reasons above which language students of EA languages must experience a steep learning curve and make a long-term investment in their language of study in order to achieve sufficient reading and writing fluency. Not only is learning how to write these symbols considered a huge hurdle for many students, but it is one of the most difficult tasks in learning EA languages in general [4]. The problem is further

compounded by the fact that students who study the more complex writing scripts must have working knowledge of no less than two thousand “graphic symbols” (e.g., Chinese characters) before they can effectively communicate with native writers in those languages [11].

2.2 Stroke Order and Direction Instruction

With the complexities inherent in the written component of EA languages, one technique that has proven effective to second language users (e.g., American students) – as well as being commonly taught to first language users (e.g., Chinese students) [6] – is written technique of the stroke order and direction (SOD) kind. Historically, SOD instruction places special emphasis in teaching the symbols by the written stroke according to a particular sequence [3, 4]. A subset of the major benefits to students that are exposed to SOD instruction includes the following:

- renders the symbols to be drawn in the optimal number of strokes with no wasted movement [3]
- helps keep the symbols written uniform in size [3]
- ensures that “muscle memory” is developed for writing the symbols accurately [4]
- allows to be used as one of several alternatives to reference the symbols in dictionaries [12]

Moreover, certain elements within symbols (e.g., radicals) of the more complex writing scripts in EA languages, which are instrumental in building up those symbols, are

written first; students are more likely to end up with nicely shaped symbols following the correct stroke order [3].

Due to the importance of written technique instruction, instructors greatly stress the practice early to their students not only so that they develop the “muscle memory” needed to effectively write the symbols [4], but to also discourage the development of bad learning habits that impede the pace of the learning process [6]. Major consequences of deviating from the correct written technique not only includes students writing the symbols with an altered shape [3], but also introduces the more devastating scenario of deviations occurring in students’ writings for the simpler symbols in the early stages of learning, where errors would then propagate to the more complex symbols that incorporate those simpler elements [6]. Therefore, there is a strong motivation for instructors to employ written technique instruction in EA language programs early in the process, so that bad writing habits that may hinder effective memorization may be eliminated.

2.3 Limitations of Traditional Instruction for the Written Component

The current application of written technique instruction dominantly comes in the form of paper exercises, which is supplied in supplemental workbooks and related formats for novice-level EA language textbooks. This form explicitly teaches written technique by displaying to students an example symbol, whose strokes are then numbered in the order in which the strokes should be written [4]. Although workbooks are effective in allowing students to physically perform actual writing during the

learning process, these tools alone are only effective to instructors in terms of critiquing the visual structure of students' writing; determining correctness of the written technique is not as straightforward without direct observation. One obvious reason is that instructors would evaluate the students' writings on paper, which is a static medium that does not provide dynamic information like the strokes' temporal information to explicitly evaluate for written technique correction [13, 14]. Teachers could indirectly determine written technique correctness based on the consequences of incorrect SOD, such as incorrect proportions [3], but instructors cannot respond with absolute certainty whether such consequences are the result of incorrect written technique, or if they are instead the result of incorrect visual structure independent of written technique. This issue can be resolved with the aid of instructors physically monitoring students' writing, but this solution itself comes with additional costs: not only is such an assessment time-consuming, but it is also unrealistic to execute in the classroom setting as the number of students increase [15, 16].

3. RELATED TECHNIQUES AND SYSTEMS

Given the limitations of paper-based workbooks for SOD instruction, intelligent user interfaces that use pattern recognition techniques specific to written EA languages provides a viable direction. In fact, pattern recognition algorithms for recognizing handwritten EA languages have not only existed for several decades [17], but have also been used in systems for the instruction of written EA languages [18]. These recognition systems in general have historically been distinguished into two different classes [19]:

- **Online systems.** Handwriting data is captured during the writing process, which makes available the information on the ordering of the strokes.
- **Offline Systems.** Recognition takes place on a static image captured once the writing process is over.

Of the two recognition system classes, online recognition is the more appealing of the two because of its ability to retain the temporal information of the strokes that could potentially be used to assess the correctness of students' SOD. In addition, two of the most popular conventional techniques for handwriting recognition in domains such as EA languages are hidden Markov models and neural networks [17, 19]. While both approaches are inherently distinct, online EA language handwriting recognition systems that employ either of these techniques can achieve high accuracy [17]. Both techniques are introduced below with explanations of their limitations in written EA language instruction.

3.1 Neural Networks

Some of the advantages of systems utilizing neural networks (NNs) for handwriting recognition of EA languages include very high recognition rates while maintaining low false recognition rates [20], the ability to support a wide range of writing styles [21], and favorable adaptability to any Chinese character feature [20]. In fact, NNs serve as the backbone for handwriting recognizers such as Input Method Editors (IMEs) for EA language in the latest versions of Microsoft's Windows operating system, whose implementation functions similarly to other NN implementations in that recognition is based on various features from users' digital handwritten input [21]. These advantages are especially appealing to the recognition systems' target users whom are native or expert writers of these EA languages, since accuracy rates do not suffer when, for example, users write symbols with an alternative SOD or with a non-standard number of strokes.

The strengths of NNs stem from their inherent optical character structure, which recognizes handwriting solely based on their visual structure [20]; in other words, the timing and ordering of the points from the digital strokes are disregarded since these techniques rely on some form of template-matching. From a pedagogical perspective, these strengths become weaknesses for assessing the correctness of students' written technique for their handwritten EA language symbols, since the information discarded from NNs are the very information used to allow for the assessment. This means that systems employing NNs will have difficulty recognizing whether a students' handwritten input whose visual structure is correct may or may not also have correct

written technique, a situation similarly faced by language instructors whom are asked to provide written technique assessment based solely on completed writings.

3.2 Hidden Markov Models

Systems that utilize hidden Markov models (HMMs) differ from their NN-based counterparts in that HMM-based systems take into account how users write in the recognition process. While HMMs do not perform as well to NNs when similar features are applied [19], HMM-based systems still produce high recognition rates [17] and are advantageous in that they can be compacted for use in smaller computers such as mobile devices [22]. The general steps that HMM-based systems use to classify the handwritten EA language symbols are as follows [17]:

- 1) Sample the points from the handwritten data.
- 2) Extract the features or segment the lines from the sampled points.
- 3) Codify the strokes directly, such as providing indexing labels.
- 4) Assign probabilities to the strokes, much like how HMMs are typically employed.
- 5) Determine how those features or lines interrelate.
- 6) Determine the hierarchical structure (e.g., composition of simpler subcomponents of symbols, if any).

The main criticism of recognition systems that employ HMMs specifically for written technique instruction is that the SOD information extracted from the handwritten data is used primarily to aid in the handwriting recognition process. CALL applications

for written EA languages that rely solely on an HMM-based implementation have the significant consequence of not being able to provide feedback that can differentiate between handwritten input that is visually correct but technically (i.e., in terms of SOD) incorrect, and handwritten input that is both visually and technically correct. Furthermore, HMMs by design require a different model for each possible set of SODs that students may feasibly write. Otherwise, HMMs will misclassify some students' handwritten input that has unaccounted stroke order or direction possibilities, since these possibilities are assigned extremely low probabilities in the recognition process by default.

3.3 Previous Works in Computer-Assisted Language Learning

Existing computer-assisted language learning (CALL) tools aim to improve the language curriculum by augmenting conventional classroom practices with automated help, and one of the more established categories of written EA language-based CALL tools caters specifically to the Chinese character writing script [17]. Despite the script's name, Chinese characters are not only used entirely in written Chinese (i.e., the *hanzi* script), but they also see significant use in written Japanese (i.e., the *kanji* script) and limited use in written Korean (i.e., the *hanja* script) [12]. Moreover, this particular script has the properties of being complex and having highly variable visual structure in comparison to other EA writing scripts, while also being conventionally taught using SOD instruction. Due to properties such as these, CALL systems specific to the Chinese character writing script share very similar properties to those specific to the other EA

writing scripts. In other words, CALL systems for the Chinese character writing script can generalize to and are representative of CALL systems for the other EA writing scripts without much loss of generality.

Development of CALL systems for written Chinese characters have existed since the early part of the 1990s [23], and some CALL systems such as Online Chinese Flashcards and FlashcardsExchange provide digital versions of traditional flash cards [4]. Other CALL systems such as eStroke, Chinese Writing Master, and New Practical Chinese Reader go a step further from their paper-based counterparts by animating the model SOD of the characters [4].

While the above CALL systems aim to provide digital extensions of static paper-based tools, these CALL systems lack a sketching modality that incorporates artificial intelligence-based feedback in the learning process [11, 24]. Other types of “pen-less” systems utilize alternative audio or visual modalities that involve prompting the user to repeat or identify characters on the computer screen. While these systems expand on “flashcard”-based CALL systems that merely translate paper-based information into a digital format [14], the absence of a sketching modality contrasts with the explicit writing that is conventionally taught and used in the language curriculum.

Since conventional pattern techniques (e.g., HMMs, NNs) for recognizing written EA languages (e.g., Chinese) are limited in their ability to simultaneously assess both the visual structure and written technique of students’ written characters, researchers have devised alternative approaches to overcome these restrictions. One of the earliest research works from [25], and later improved upon in [26], utilized two

separate techniques for assessing the SOD of the prompted characters. Assessment of SOD correctness first involved defining a stroke as the endpoints of the lines that make up a stroke, and then critiquing the correctness of the sequence of spatial positions relative to the other strokes in the character. Assessment of stroke direction correctness, in comparison, first determined the direction of the lines in each stroke based on the temporal sequence of the endpoints, assigned them a numerical code that corresponds to one of the eight compass directions, and then analyzed the correctness of the sequence. This particular system was limited in that users needed to trace over the outline of the characters. In other words, the correctness of the visual structure in the students' handwriting is never assessed since no actual handwriting recognition occurs.

Subsequent research works from Chen [15, 16] provided more freedom in how users write the characters. Their method in assessing SOD involved grouping all possible lines into six different slopes, and then critiquing the SOD based on the temporal sequence written by the user. This work advances the previous research work from [25, 26] in that handwriting recognition exists, but restrictions still exist in the handling of the visual structure assessment. The system assumed the entire drawing area of the character was dedicated to one character, and the correctness of the character was based on all the features within this coordinate space. While this is sufficient for novice-level courses that introduce characters and symbols of EA writing scripts, the inabilities of this research to handle size independence and multiple characters and symbols mean that assessment cannot be handled in more natural writing situations like for writing phrases or sentences in free-sketch writing areas similar to paper.

Another recent work by Qi [27] proposed a solution to the size independence issue by first defining a bounding box for handwritten input of characters, then splitting the input into a three-by-three grid, and then comparing the features from the pixels in those grid blocks to possible candidate characters, splitting the input into a three-by-three grid, and then comparing the features from the pixels in those grid blocks to possible candidate characters. While the size independence issue was addressed, the system assumed that the strokes originated from a single character; the paper also did not provide information on how it could handle recognition of multiple characters within the same writing space. Another concern was that the system always made the assumption that the written input lied within a quadrate block. This would lead to the consequence of failing to account for single-line symbols that exist in a number of EA writing scripts, which plays a much more significant role when it is part of grouped symbols (e.g., vocabulary phrase, phonetic pronunciation) that students could feasibly be prompted to provide. An additional concern with the approach is that the written technique assessment is handled separately in a separate system, which can possibly lead to incompatible scenarios where the written technique assessment from the separate system outputs the input as being correct for visually incorrect characters.

One of the latest pen-enabled CALL systems for teaching the characters with written technique assessment comes from Tian, et al. [28]. In order to teach students the correct SOD, this particular system uses the \$1 recognizer, which is an easy-to-implement algorithm for recognizing user-defined sketch gestures [29]. Due to the \$1 recognizer's inherent template-matching nature, recognition is handled by having

students' strokes matched to the model characters due a distance measurement. While the system provides written technique assessment, the amount of assessment that is provided is imprecise; that is, the correctness of the SOD is determined by a threshold of the number of incorrect strokes instead of whether the SOD is exactly correct or not. The consequence is that if a student provides an incorrect SOD, the system will still consider the character to be correct as long as the number of incorrect strokes is below the threshold. In terms of the visual structure, a direct implementation of the \$1 recognizer would be problematic for matching the length of strokes to determine correct proportionality, particularly for short strokes [29]. The system addresses this issue by weighting the length of the strokes, but its dependence on the \$1 recognizer limits assessment to single-characters within a constrained box.

Alternative approaches provided by the CALL systems Hashigo [30] and LAMPS [31] for the EA writing scripts of Japanese kanji and Mandarin Phonetic Symbols I, respectively, also similarly assess both the visual structure and written technique of students' writings. Both Hashgio and LAMPS adapt free-sketch recognition techniques capable of recognizing unconstrained writing with reasonable accuracy, while also addressing the lingering issues of the previously mentioned prior research work and applications of multi-symbol input and input size independence. The research work from Hashigo and LAMPS serve as the basis of this thesis, and the content detailed in this thesis generalizes and elaborates further from the research work of those two systems.

3.4 Corner-Finding Algorithms

The core ideas behind constructing a visual structure and written technique assessment-capable system for written EA language instruction derive from the sketch recognition literature, specifically research that focuses on geometric-based recognition. One of two contributions from geometric-based recognition that is highly relevant for developing such a capable system and is heavily utilized for the methodology is corner-finding algorithms (i.e., Sezgin and PaleoSketch).

A valuable technique in the methodology is the use of corner-finding algorithms on captured data that digitally represents the written input by the users (e.g., students). Prior to executing these algorithms, raw data is first collected from users on pen-capable computers (e.g., Tablet PCs) through a stylus that is used to input their writing; the data is then stored in memory and later represented back to users as a given set of pixels. By treating this set of pixels as points in Cartesian space, the advantages of employing the different corner-finding algorithms can be exploited for processing these sequence of points and later approximated back as basic geometric primitive shapes (e.g., lines, curves, arcs, ellipses) [32, 33]. One assumption that is taken advantage of is visually approximating the strokes as a set of primitives. This particular assumption allows for the exploitation of stroke processing algorithms that are capable of fragmenting the collected pixel points into elementary geometric shapes, which is later used for written recognition and subsequent feedback and assessment. Based on observations of students' writing habits being more careful during the learning process, this assumption generally holds well for most EA writing scripts.

To aid in the task of processing strokes into their representative geometric primitives, the corner-finding capabilities from the Sezgin [33] and PaleoSketch [32] algorithms were selected for their strengths in fragmenting the raw strokes into recognized geometric primitives. Since a stroke is defined in this thesis as being a temporal sequence of points collected on a computer, from the pen-down motion on the writing surface to the pen-up motion, the key idea shared by these two corner-finding algorithms is that the corresponding corners detected in a stroke serve as the endpoints of recognized geometric primitives.

Both the Sezgin and the PaleoSketch algorithms were utilized in order to take advantage of their respective strengths. The Sezgin algorithm's ability to detect corners for lines from strokes stems from the observation that people slow down during the formation of corners in their writing. Therefore, the algorithm relies on curvature and velocity data from the direction of the pen writing in order to make its selection of the stroke's corners. Alternatively, the PaleoSketch algorithm's ability to detect corners specifically for other geometric shapes (e.g., arcs, ellipses) uses the same concepts of computing the direction, velocity, curvature, and corner values from the Sezgin algorithm [32]. The PaleoSketch algorithm further expands on the Sezgin algorithm by calculating the normalized distance between direction extremes (NDDE) and direction change ratio (DCR), two additional features that have proven very useful in the algorithm's ability to recognize a larger set of geometric shapes.

The importance of processing the strokes from their explicit temporal sequence of points into their geometric primitives is stressed in the research work of this thesis,

because it enables implementations that employ this methodology to later achieve handwritten recognition of those geometric primitives from the handwritten input and independent of the size of the writing space. The result is that recognition occurs in a writing environment that more closely emulates writing naturally done on paper. With the geometric primitives recognized, the next important step is to recognize the interactions between those primitives. These interactions between the constraints help make it possible for the written input to be visually categorized to symbols from EA writing scripts. The tool we use to keep track of the primitives and their interactions are handled using a sketching language called **A Language to Describe, Display, and Editing in Sketch Recognition (LADDER)**, which is further elaborated in the next section.

3.5 LADDER Sketching Language

Once the strokes for the users' writing are processed into their geometric primitives, the groupings of those primitives are categorized using pattern recognition techniques. In order to provide this pattern recognition with reasonable accuracy, the LADDER sketching language [34] was employed to fulfill the methodology's sketch recognition needs. Since LADDER is a general purpose sketching language for describing how sketch diagrams for various domains are drawn, displayed, and edited, this second contribution of geometric-based recognition through the form of a sketching language was adopted to recognize symbols from the various EA writing scripts.

What differentiates the pattern recognition techniques on users' handwriting with the LADDER language from traditional pattern recognition techniques (e.g., neural networks, hidden Markov models) for the domain of written EA languages is the emphasis on recognizing the writing. Specifically, the methodology focuses more on recognizing users' handwriting based on whether it fulfills a set of requirements. Not only does this free systems that implement this methodology from using training data that restricts the recognition to existing training data from model users, but it is also similar to how language teachers determine whether students succeeded in writing the symbols for a particular EA writing script correctly by verifying if all the necessary visual structure requirements for those symbols have been met.

It should be noted that the methodology contrasts sharply with how alternative pattern recognition techniques are handled for recognizing users' handwriting from a pedagogical perspective. A major disadvantage of these alternative systems involves instances where a student may write a particular symbol visually incorrect to a slight degree (e.g., missing or extra strokes, sloppiness), yet still obtain a response from these systems that the input is correct (i.e., the system gives a false positive on this slightly incorrect input). This is one of the consequences of traditional pattern recognition techniques which inherently recognize written input based on the closest match in the training set. This greater leeway in recognition may be appropriate for native writers of specific EA writing scripts, since these writers would prefer writing symbols with the convenience of higher recognition over the perceived hassle of pedagogical-based feedback on a domain that they have already mastered. For students learning symbols

for their target EA writing script of study, this extra leeway in recognition is less suitable, since it would deteriorate students' learning of the writing script due to the system not correcting those slight mistakes.

The actual recognition of students' handwritten symbols using LADDER involves the use of shape descriptions, which are structures primarily containing geometric information for categorizing the handwritten input using the sketching language's syntax. The shape descriptions that are constructed in LADDER can be used to describe a wide variety of shapes such as the symbols from the various EA writing scripts. These shape descriptions consist of multiple specifications, and how these specifications are used in recognizing the visual structure and written technique of students' handwritten symbols in EA writing scripts are elaborated next.

4. OVERVIEW OF VISUAL STRUCTURE ASSESSMENT

Visual structure correctness of language students' handwritten symbols in an EA writing script is one important criterion that is necessary for mastering their target language of study. Existing CALL systems that employ pen-based input conventionally support automated assessment capabilities of the visual structure, but their support is largely limited to single input within a fixed writing space environment. Expanding this automated assessment to handle the type of writing that is naturally done on paper allows students to receive the same kind of valuable feedback without sacrificing writing environment realism. On the other hand, the consequence is that supporting this broader form of visual structure assessment creates additional challenges, since recognition not only includes classifying what the symbol is, but also includes determining which strokes belong to what symbol. In other words, the challenges for this broader form of visual structure assessment include existing and newer challenges:

- 1) **Classification.** Recognizing what symbol was written.
- 2) **Grouping.** Recognizing which strokes correspond to which symbol.
- 3) **Size independence.** Handling size variations of written symbols.

In order to expand visual structure assessment to include the kind of free-sketch input found in real world writing, the methodology adapts the LADDER sketching language for use in classifying students' symbols in the domain of written EA languages. As a sketching language, LADDER is capable of handling multiple domains and the wide array of shapes contained in them through the use of structured geometric

information called *shape descriptions*. Yet while the sketching language has successfully been used for recognizing shapes in engineering and visual design domains, LADDER was designed for recognizing shapes with specific properties in mind (Table 1).

The following subsections will first briefly introduce the relevant aspects of the LADDER sketching language to the methodology, and also describe how the methodology adapted LADDER to employ geometric-based recognition on written EA language symbols. Afterwards, the approach for construction shape descriptions specific to recognizing handwritten symbols of written EA languages will be presented.

Table 1. Ideal shapes for the LADDER sketching language and the challenges for written symbols in EA languages.

Property	Explanation	Challenge
Describable in a fixed graphical grammar.	Shapes are recognized, finitely enumerable geometric information. In other words, if a shape can be rigidly described geometrically, then it is possible for LADDER to recognize it.	LADDER was designed for shape recognition, which is distinct from the traditional handwriting recognition techniques employed on handwritten EA symbols.

Table 1 (cont.). Ideal shapes for the LADDER sketching language and the challenges for written symbols in EA languages.

Property	Explanation	Challenge
Solely composed of primitive constraints.	Shapes must either be a primitive geometric shape or a combination of them. If a shape or a particular part of it contains a non-primitive constraint, then it is not included in the shape description.	Some symbols in EA writing scripts are composed of components that may not easily be described geometrically. These parts may require that they be approximated as geometric primitives, possibly at the sacrifice of accuracy.
Few curves or trivial curves details.	Curves in general are much more difficult to describe geometrically, since they contain much more variations. Incorporating the necessary geometric information to capture this variety would greatly complicate constructing shape definitions in LADDER.	Some scripts in written EA languages consist of symbols that contain non-trivial curves. Symbols with these curves can exist in the same stroke as non-curves, which complicate the task for corner-finding algorithms.
Much regularity and few details.	Irregular shapes and shapes with numerous details become problematic with LADDER since this expands both the length and the logic of shape descriptions. Consequences include lengthier times to debug shape descriptions and increased recognition running time to check if constraints have been fulfilled.	Symbols in some written EA scripts are constructed in a hierarchy, which is a feature supported in LADDER. For the more complicated symbols, which may consist of quite a number of details (e.g., many strokes) and much irregularity (e.g., hierarchy of several layers), this may cause non-trivial running time issues.

4.1 Relevant LADDER Specifications

The structured geometric information that is shape descriptions (i.e., for LADDER to reference shapes from a wide array of domains) is broken up into multiple parts. The parts that are used as the building blocks of shape descriptions are as follows: *components*, *constraints*, *aliases*, *editing*, and *display*. Of the five specification parts, the first three (i.e., components, constraints, and aliases) play a significant role for the written structure assessment, while the last one (i.e., display) aids in providing visual feedback to the student. The following subsections summarize their purpose in LADDER and elaborate on how they are adapted in the methodology for recognizing symbols in written EA languages.

4.2 Components

The first part of the shape descriptions is the components section, which consists of a list of elements that a shape is built from. Components serve as the building blocks of shapes and are analogous to ingredients in a food recipe. Furthermore, components must first be defined before defining the rest of the specifications of the shape descriptions, since the rest of the shape descriptions are dependent on knowing the components to constrain on. These components can be categorized into three different categories (Figure 1):

- **Primitive geometric components (i.e., primitives).** By definition, these types of components are the most fundamental shapes for any domain. In other words, primitives cannot be further broken down to smaller components, since they

serve as the base components for shapes in the domain. The primitives relevant to the symbols in written EA languages for this methodology are: *lines*, *curves*, and *ellipses* (Figure 2.a).

- **Simple components.** These components are related to primitives in that they are a combination of primitives (Figure 2.b). In other words, a simple component is entirely built of only primitive parts that are not already themselves simple components. If primitives are the building blocks of simple components, then these components are the building blocks of the next category of components called compound components.
- **Compound components.** These components differ from simple components in that they are a combination of smaller components; they can either be built from simple components, simpler compound components, or a mixture of the two (Figure 2.c). Compound components relevant in written EA languages for this methodology include but are not limited to: *symbols* from EA writing scripts (e.g., Chinese characters), *radicals* (i.e., subcomponents of Chinese characters), and other subparts of written EA language symbols.

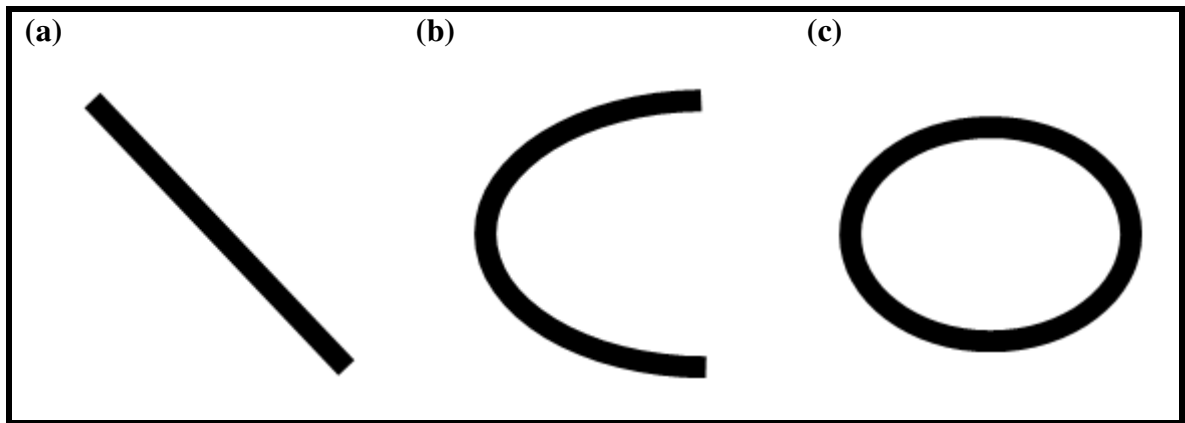


Figure 1. The three primitive geometric components used in the methodology for written EA languages: (a) lines, (b) curves, and (c) ellipses.

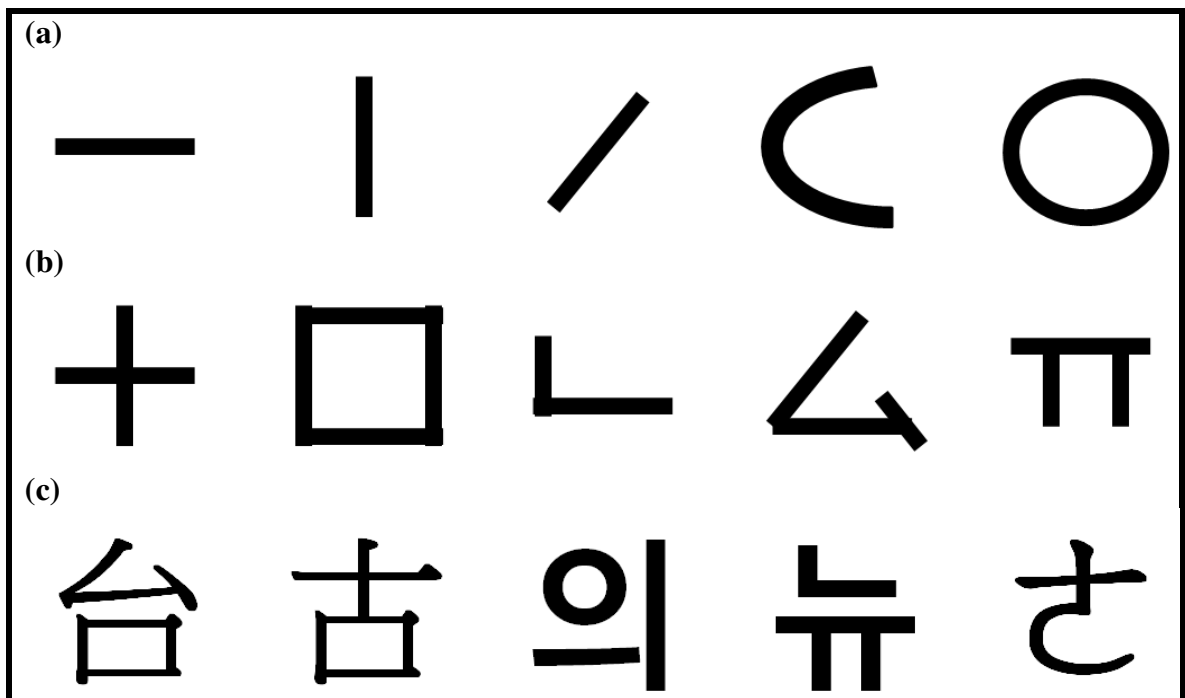


Figure 2. Examples of the utilized geometric components: (a) primitives, (b) simple components, and (c) compound components.

The visual structures of the different symbols in the various writing scripts of EA languages are diverse in nature, and the question that needed to be addressed for the methodology was whether the selected three primitives are sufficient to capture these visual structures while retaining reasonable recognition rates. After empirical observations, the following conclusions were derived based on the symbols that are commonly taught in introductory EA language classes:

- **Many strokes in written EA language symbols can be sufficiently approximated with line primitives.** One of the techniques used in HMMs for recognizing written EA language symbols, especially for Chinese characters, is to approximate them as a collection of lines. This strategy is adapted for the geometric-based methodology discussed in this thesis for symbols that possess a dominantly line-based visual structure.
- **Curves with large degree of “bending” are approximated as curve primitives.** Strokes that visually resemble curves pose a challenge compared to lines, since curves present much more variety and are geometrically more complex to define. Due to this, curves that have a high degree of curvature and cannot be reliably recognized as a sequence of line primitives are instead treated as curve primitives.
- **The circular subparts of symbols are treated as ellipse primitives.** Much like how circles are treated as special cases of ellipses in geometry, sketched circles that make up a subpart of certain written EA language symbols are recognized as ellipse primitives.

In addition to listing the components in the shape description for a particular written symbol, one can also assign unique labels for these symbols, much like how variables can be assigned names in conventional programming languages. For this methodology, special care is taken to how the components are labeled in providing a systematic naming scheme. This becomes more relevant for the alias specification, but in the mean time, the initial naming scheme for the following components is as follows:

- **Line components.** Labels for lines are given based on both orientation and relative location within the symbol. For example, if there exist a horizontal line located on the left side of a multi-stroke symbol, then that line is labeled as *leftVertLine*.
- **Curve components.** Labels for lines are given based on relative location within the symbol. If the curve within the symbol is unambiguous (e.g., there is only one curve within the symbol), then it is labeled simply as *curve*.
- **Ellipse components.** Similar to curves, labels for ellipses are given based on relative location within the symbol, and are similarly labeled simply as *ellipse* if the primitive component is unambiguous within the symbol.
- **Simple/Compound shape components.** Labels for single and compound shapes (i.e, the written EA language symbols) are given based on one of three naming schemes:
 - **Enumerated name.** If a particular symbol is part of a list of symbols that are being taught or tested on, then the symbol is assigned an enumeration

that matches the one given in the corresponding language textbook that contains the symbol.

- **English translation.** If the symbol is not given an enumerated symbol due to not fulfilling the previous conditions for one (e.g., it is an unnamed subcomponent, it is a review symbol from a previous symbol), then it is labeled by its English translation.
- **Romanization equivalent.** Given the nature of EA languages, there might not be a simple direct translation of a symbol (e.g., it is used in a grammatical structure, it has a complicated translation). In this case, it is labeled by its Romanization equivalent.

4.3 Constraints

Following components is the second specification called constraints, which is defined as the geometric relationships between the components. Resorting to the food recipe analogy once again, if components serve as the ingredients, then constraints are the cooking instructions. In other words, after the components are checked in the sketched input to determine if they exist, the constraints are then checked for the correctness of their relationship behavior. These kinds of relationships between the components can either be unary, binary, or ternary constraints.

The LADDER sketching language already includes a library of constraints for use in shape descriptions to recognize written input from a variety of domains. Of these existing constraints, a subset of those constraints were found to be highly useful for

creating shape descriptions specific to recognizing written East Asian language symbols. These available LADDER constraints that were employed are grouped into the following categories and described in Table 2 through Table 7.

Table 2. Orientation Constraints: checks whether a line is a slope or an anti-diagonal; they are always unary. E.g., checks if a particular line has a positive slope.





Vertical	The line is vertical.	
Horizontal	The line is horizontal.	
NegSlope	The line has a negative slope.	
PosSlope	The line has a positive slope.	

Table 3. Point Relationship Constraints: Compares the center, endpoint, or bounding position of two shapes. E.g., checks if the center of one line is left of the center of another line.

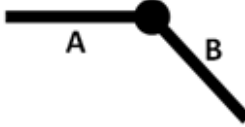
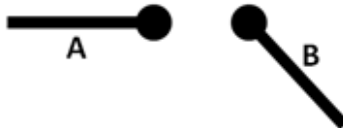

Coincident	A point from Line A is coincident to a point from Line B.	
SameX	A point from Line A has the same x-coordinate as a point from Line B.	
SameY	A point from Line A has the same y-coordinate as a point from Line B.	

Table 4. Position Constraints: Compares the position of two shapes relative to each other.
E.g., checks if a particular line is left of a particular circle.

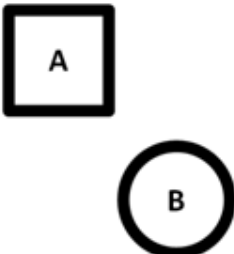
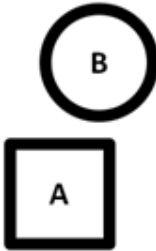
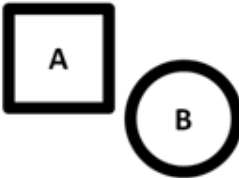
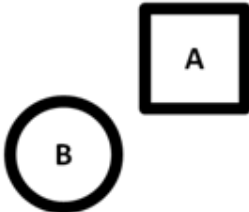
Above	Shape A is above Shape B.	
Below	Shape A is below Shape B.	
LeftOf	Shape A is to the left of Shape B.	
RightOf	Shape A is to the right of Shape B.	

Table 5. Proximity Constraints: Checks for the closeness proximity of one shape to another by some relative threshold value. E.g., checks if a particular shape is near another particular shape.

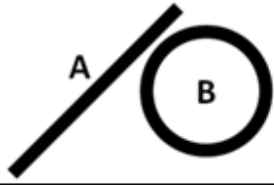

Near	Shape A is near Shape B.	
Closer	Shape A is closer to Shape B than to Shape C.	

Table 6. Length Constraints: Compares the length of two lines relative to each other. E.g., checks if one line is longer than another line.


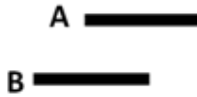


Longer	Line A is longer than Line B.	
EqualLength	Line A and Line B are of equal length.	

Table 7. Logical Constraint: Involves negating a constraint or operating disjunction on two constraints. By default, all constraints in LADDER are mutually conjunctive. E.g., checks if a line does not have a positive slope.

Not	The negation of a constraint. E.g., The line is not horizontal.	
Or	The disjunction of two constraints. E.g., The line is positive-sloped or negative-sloped.	

4.4 Aliases

The third specification is aliases, which is conventionally used in LADDER to simplify other elements in the description. As a feature in LADDER, aliases provide a mechanism to assign additional alternate labels to existing component names. One of the concrete benefits of using aliases is their ability to provide more intuitive names to existing components or the subparts (e.g., inner components, points) within those components. In regard to this methodology, aliases are given to primitive components (i.e., lines, curves, and ellipses) based on their stroke order enumeration, and optionally given to non-primitive components based on naming schemes that were not initially used in the components specification of the shape description (Table 8).

Table 8. Comparisons between naming schemes for original labels and aliases.

Component Type	Original Label Naming Scheme	Alias Naming Scheme
Line	Relative location. Orientation type.	Stroke order enumeration.
Curve	Component type. Relative location (optional).	Stroke order enumeration.
Ellipse	Component type. Relative location (optional).	Stroke order enumeration.
Simple/Compound Symbol	Enumerated name or English translation or Romanized equivalent.	Naming scheme not chosen in original label (optional).

Aliases not only serve as a convenience in constructing shape descriptions for a variety of domains in LADDER, but they also serve a dual-purpose specifically for the domain of written East Asian languages in the methodology. That is, for the aliases

applied to the line, curve, and ellipse components in the shape descriptions, they are also specifically referenced for assessing the correctness of students' written technique. This important secondary feature of aliases is further elaborated in the section dedicated to the handling of written technique, but a comparison of the original component labels and their matching aliases can be found in Figure 3.

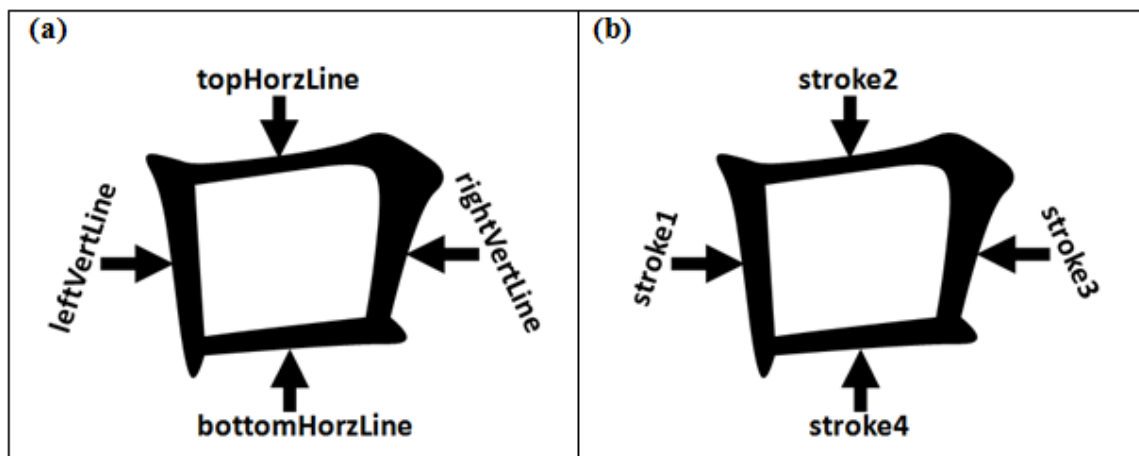


Figure 3. A side-by-side comparison of the original component labels and the corresponding aliases for an example Chinese character: (a) the original component labels, and (b) the aliases.

4.5 Display

The last relevant specification for the methodology is the display specification, which is defined as methods that indicate what to display when the object (i.e. the sketch) is recognized. The display specification contains various methods related to how the input strokes are displayed back to the user, and while this specification does not directly affect the classification of either the visual structure or the written technique of the EA language symbol input, it provides the capability for one form of explicit feedback of

students' written input through visual cue interactions given on those input strokes. In this particular methodology, the display specification was employed in the following fashion:

- **Beautification disabled.** By default, the LADDER sketching language enables the beautification of strokes, which is the removal of mess and clutter from the original sketches such as those found in natural writing. With beautification, the strokes are visually altered to more visually precise shapes, such as straighter lines and more consistent curves. The methodology does not enable beautification, but instead maintains the look of the original strokes in order to maintain consistency of what users normally see in natural writing. In addition, it was observed that when the strokes are beautified, the beautified strokes are displaced from the original position of the original strokes. When two strokes connected at the endpoint are drawn separately (i.e., separately sketched with the lifting of the stylus), users connected the beautified strokes when beautification was enabled. This writing behavior caused recognition problems, since the corner-finding and grouping algorithms used in LADDER rely on the positions of the original strokes; when users connect the strokes on the beautified strokes instead of the original strokes, the recognition does not treat the strokes as being connected at the endpoints. Therefore, since beautification indirectly affects recognition of the written input, it is disabled in the methodology.

- **Coloring the strokes.** One of the two visual cues employed in the methodology to provide users on their sketches is changing the colors of the sketched strokes after a particular symbol has been recognized. This option of coloring the strokes when it is recognized by LADDER allows students to receive visual structure feedback by informing them that their symbol is visually correct. The stroke coloring method is more suited for instructing or reviewing the symbols, while is recommended to be disabled for testing. In this methodology, if the stroke coloring option is enabled, then unrecognized strokes are left as the default blue color, recognized subcomponents of a symbol typically change to red strokes, and the completed recognized symbols are changed once again to dark gray strokes.
- **Supplementary text output.** In addition to coloring the strokes, the sketches can be augmented with surrounding supplementary text on the drawing panel after a symbol or a subcomponent of it has been recognized. This can be used for multiple purposes, such as one more form of visual aid to the student and also as a convenient visual cue for debugging the correctness of a particular shape description for symbols. The supplementary text in the display specification can be placed in a variety of locations on and around the text.

5. METHODOLOGY OF CONSTRUCTING SHAPE DEFINITIONS

Much like how there is freedom of style in writing a piece of code for a particular high-level programming language, there are analogously numerous variations in constructing shape descriptions in the LADDER sketching language. Despite this flexibility, it is advantageous to have an efficient type of convention for designers in constructing shape definitions, similar to how existing style guides and coding conventions are provided for coders of a particular programming language. One reason is that having a convention allows shape descriptions to be constructed in a systematic and formatted methodology; not only does this reduce the complexity of constructing shape descriptions for the designer, but it also eases the debugging of shape descriptions such as when a chosen constraint performs poorly in recognition.

For the case of symbols in written EA languages, establishing a shape construction convention is even more important due to the complexity of the symbols and the similarities between them. For this methodology, a convention was introduced for the sake of creating shape descriptions that were robust enough to handle the diversity of written EA language symbols while also keeping the order of those shape descriptions manageable in terms of ease of readability. This convention can basically handle shape descriptions for most cases, and further modifications can be done to handle special cases for symbols in certain written EA language scripts.

5.1 Single Simple Shape

A single simple shape can be described straightforwardly in this thesis as an individual shape in LADDER that is built entirely of primitive shapes. Due to this, geometric constraints that are used in the shape descriptions of single simple shapes are frequently more simplified since they only interact with the endpoints and boundaries of the primitive shapes in LADDER. Despite these shape descriptions relying only on the physical properties of primitive shapes as opposed to also including those from more complex shapes, the shape descriptions for these single simple shapes are non-trivial since their correctness impacts the correctness of shape descriptions for more complex shapes that utilize single simple shapes.

While there is flexibility in how the constraints can be listed in the shape descriptions such as those for single simple shapes, for the case of shape descriptions specific to symbols of written EA languages, an ordered format style was used to order the constraints so that readability and debugging capabilities can be improved. This is also done because each line component in LADDER has endpoints and midpoints assigned $p1$, $p2$, and $center$, respectively; since the assignment of endpoints $p1$ and $p2$ in each line component changes depending on how the line is drawn when context is not provided, those endpoints are explicitly assigned their placement relative to each other in a systematic fashion. That is, the $p1$ endpoints of each line component is assigned as being left relative to their corresponding $p2$ endpoints for all non-vertical lines, and assigned as being above relative to their corresponding $p2$ endpoints for vertical lines. The order of constraint groups in the format style is summarized below.

- 1) **Line orientations.** Based on all the line components that make up the shape, these lines are constrained based on their orientation.
- 2) **Endpoint ordering.** After the lines are constrained by orientation, their endpoints are constrained based on their relative location from one another.
- 3) **Spatial relationships.** As opposed to the previous two constraint groups, these constraints consist of how the components spatially relate to other components. In other words, the spatial relationships group consists of the rest of the constraints that make up the constraints portion of the shape descriptions.

Listing aliases also provide an important contribution in building shape descriptions for single simple shapes. One advantage is through ease of use; that is, aliases allow a designer to reference a particular part of a shape (e.g., *vertLine.pl*) with an easier-to-understand label (e.g., *leftPoint*). Another advantage is through practicality; that is, the only way for more complex shapes in LADDER to utilize a specific component from a simpler shape is by explicitly referencing it through its alias. This is done for the sake of computation, since the computational time to allow designers to possibly directly access every possible combination of subcomponent when constructing more complex shapes becomes exponentially large. From the standpoint of single simple shapes though, since these shapes are constructed solely using primitive shapes, the value of aliases does not seem immediately apparent. It is still important to label specific parts of these shapes as aliases when these shapes are used to build compound shapes, since this simplifies the process of constructing compound shapes composed of single shapes. This will be made more readily apparent in the next subsection.

	<u>Name:</u>	
	TenKanji	
	<u>Components:</u>	
	Line	horzLine
	Line	vertLine
<u>Constraints:</u>		
Horizontal	horzLine	
Vertical	vertLine	
LeftOf	horzLine.p1	horzLine.p2
Above	vertLine.p1	vertLine.p2
EqualLength	horzLine	vertLine
SameX	horzLine.center	vertLine.center
SameY	horzLine.center	vertLine.center
<u>Aliases:</u>		
Point	horzLine.p1	leftPoint
Point	horzLine.p2	rightPoint
Point	vertLine.p2	bottomPoint
...		

Figure 4. A shape description for the Chinese character *ten*.

To illustrate the methodology of constructing shape descriptions for single simple shapes, example shape descriptions for two specific Chinese characters – *ten* and *mouth* – are introduced in Figure 4 and Figure 5, respectively. These two Chinese characters are not only composed entirely of primitive shapes, or more specifically lines, but they are also simple single shapes that are commonly used in more complex EA symbols. In fact, these two Chinese characters will be combined in the next section in order to describe the methodology for constructing single compound shapes.

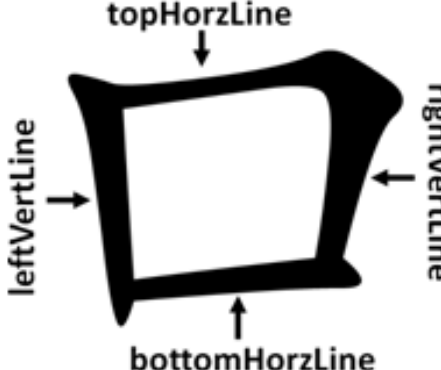
	<p>Name:</p> <p>MouthKanji</p>																																										
	<p>Components:</p> <table> <tr> <td>Line</td> <td>leftHorzLine</td> </tr> <tr> <td>Line</td> <td>rightHorzLine</td> </tr> <tr> <td>Line</td> <td>topVertLine</td> </tr> <tr> <td>Line</td> <td>bottomVertLine</td> </tr> </table>	Line	leftHorzLine	Line	rightHorzLine	Line	topVertLine	Line	bottomVertLine																																		
Line	leftHorzLine																																										
Line	rightHorzLine																																										
Line	topVertLine																																										
Line	bottomVertLine																																										
<p>Constraints:</p> <table> <tr> <td>Horizontal</td> <td>leftHorzLine</td> <td></td> </tr> <tr> <td>Horizontal</td> <td>rightHorzLine</td> <td></td> </tr> <tr> <td>Vertical</td> <td>topVertLine</td> <td></td> </tr> <tr> <td>Vertical</td> <td>bottomVertLine</td> <td></td> </tr> <tr> <td>Above</td> <td>leftVertLine.p1</td> <td>leftVertLine.p2</td> </tr> <tr> <td>LeftOf</td> <td>topHorzLine.p1</td> <td>topHorzLine.p2</td> </tr> <tr> <td>Above</td> <td>rightVertLine.p1</td> <td>rightVertLine.p2</td> </tr> <tr> <td>LeftOf</td> <td>bottomHorzLine.p1</td> <td>bottomHorzLine.p2</td> </tr> <tr> <td>Coincident</td> <td>leftVertLine.p1</td> <td>topHorzLine.p1</td> </tr> <tr> <td>Coincident</td> <td>topHorzLine.p2</td> <td>rightVertLine.p1</td> </tr> <tr> <td>Coincident</td> <td>rightVertLine.p2</td> <td>bottomHorzLine.p2</td> </tr> <tr> <td>Coincident</td> <td>leftVertLine.p2</td> <td>bottomHorzLine.p1</td> </tr> <tr> <td>EqualLength</td> <td>leftVertLine</td> <td>rightVertLine</td> </tr> <tr> <td>EqualLength</td> <td>topHorzLine</td> <td>bottomHorzLine</td> </tr> </table>		Horizontal	leftHorzLine		Horizontal	rightHorzLine		Vertical	topVertLine		Vertical	bottomVertLine		Above	leftVertLine.p1	leftVertLine.p2	LeftOf	topHorzLine.p1	topHorzLine.p2	Above	rightVertLine.p1	rightVertLine.p2	LeftOf	bottomHorzLine.p1	bottomHorzLine.p2	Coincident	leftVertLine.p1	topHorzLine.p1	Coincident	topHorzLine.p2	rightVertLine.p1	Coincident	rightVertLine.p2	bottomHorzLine.p2	Coincident	leftVertLine.p2	bottomHorzLine.p1	EqualLength	leftVertLine	rightVertLine	EqualLength	topHorzLine	bottomHorzLine
Horizontal	leftHorzLine																																										
Horizontal	rightHorzLine																																										
Vertical	topVertLine																																										
Vertical	bottomVertLine																																										
Above	leftVertLine.p1	leftVertLine.p2																																									
LeftOf	topHorzLine.p1	topHorzLine.p2																																									
Above	rightVertLine.p1	rightVertLine.p2																																									
LeftOf	bottomHorzLine.p1	bottomHorzLine.p2																																									
Coincident	leftVertLine.p1	topHorzLine.p1																																									
Coincident	topHorzLine.p2	rightVertLine.p1																																									
Coincident	rightVertLine.p2	bottomHorzLine.p2																																									
Coincident	leftVertLine.p2	bottomHorzLine.p1																																									
EqualLength	leftVertLine	rightVertLine																																									
EqualLength	topHorzLine	bottomHorzLine																																									
<p>Aliases:</p> <table> <tr> <td>Point</td> <td>topHorzLine.p1</td> <td>topLeftPoint</td> </tr> <tr> <td>Point</td> <td>topHorzLine.center</td> <td>topPoint</td> </tr> <tr> <td>Point</td> <td>topHorzLine.p2</td> <td>topRightPoint</td> </tr> <tr> <td>Point</td> <td>bottomHorzLine.center</td> <td>bottomPoint</td> </tr> <tr> <td>...</td> <td></td> <td></td> </tr> </table>		Point	topHorzLine.p1	topLeftPoint	Point	topHorzLine.center	topPoint	Point	topHorzLine.p2	topRightPoint	Point	bottomHorzLine.center	bottomPoint	...																													
Point	topHorzLine.p1	topLeftPoint																																									
Point	topHorzLine.center	topPoint																																									
Point	topHorzLine.p2	topRightPoint																																									
Point	bottomHorzLine.center	bottomPoint																																									
...																																											

Figure 5. A shape description for the Chinese character *mouth*.

5.2 Single Compound Shape

Expanding beyond single simple shapes are single compound shapes; shapes of this kind are composed of at least two simpler shapes, one of which is of type single simple shape. Two obvious benefits in creating shape descriptions for a more complex Chinese character as a single compound shape is that it:

- 1) simplifies the logic of designing, and
- 2) reduces the computational time in recognizing that shape.

For the former, the designer can simply add an existing simple shape as one of the components into the shape description instead of re-writing that simple shape's shape description. For the latter, the computation time for recognizing these more complex shapes is reduced as a result of LADDER processing less primitive shapes.

The importance of creating alias labels for relevant parts of the single simple shapes in the previous section can now be realized for shape descriptions specific to single compound shapes. The reason is that it makes it easier, perhaps even feasible, to determine where in the shape description a particular single simple shape is relative to other inner shapes for the compound shape. In fact, creating shape descriptions for single compound shapes are simply an extension of the process in creating shape descriptions for single simple shapes. Furthermore, the user-created alias labels (e.g., labels *rightPoint* and *bottomRightPoint*) for simple shapes (e.g., a basic Chinese character) within single compound shapes are analogous to the default core components (e.g., the points *p1*, *p2*, and *center*) for primitive shapes (e.g., a line) within single simple

shapes. Figure 6 demonstrates one such shape description for a single compound shape composed of two single simple shapes.


	<p><u>Name:</u> AncientKanji</p> <p><u>Components:</u></p> <table> <tr> <td>TenKanji</td> <td>ten</td> </tr> <tr> <td>MouthKanji</td> <td>mouth</td> </tr> </table>	TenKanji	ten	MouthKanji	mouth								
TenKanji	ten												
MouthKanji	mouth												
<p><u>Constraints:</u></p> <table> <tr> <td>SameX</td> <td>ten.bottomPoint</td> <td>mouth.topPoint</td> </tr> <tr> <td>SameY</td> <td>ten.bottomPoint</td> <td>mouth.topPoint</td> </tr> <tr> <td>LeftOf</td> <td>ten.leftPoint</td> <td>mouth.topLeftPoint</td> </tr> <tr> <td>RightOf</td> <td>ten.rightPoint</td> <td>mouth.topRightPoint</td> </tr> </table>		SameX	ten.bottomPoint	mouth.topPoint	SameY	ten.bottomPoint	mouth.topPoint	LeftOf	ten.leftPoint	mouth.topLeftPoint	RightOf	ten.rightPoint	mouth.topRightPoint
SameX	ten.bottomPoint	mouth.topPoint											
SameY	ten.bottomPoint	mouth.topPoint											
LeftOf	ten.leftPoint	mouth.topLeftPoint											
RightOf	ten.rightPoint	mouth.topRightPoint											
<p><u>Aliases:</u></p> <table> <tr> <td>Point</td> <td>ten.topPoint</td> <td>topPoint</td> </tr> <tr> <td>Point</td> <td>ten.rightPoint</td> <td>rightPoint</td> </tr> <tr> <td>Point</td> <td>mouth.bottomPoint</td> <td>bottomPoint</td> </tr> <tr> <td>...</td> <td></td> <td></td> </tr> </table>		Point	ten.topPoint	topPoint	Point	ten.rightPoint	rightPoint	Point	mouth.bottomPoint	bottomPoint	...		
Point	ten.topPoint	topPoint											
Point	ten.rightPoint	rightPoint											
Point	mouth.bottomPoint	bottomPoint											
...													

Figure 6. A shape description for the Chinese character *ancient*.

5.3 Multiple Shapes

The next extension from recognizing single shapes – whether they are simple or compound – is recognizing more than one of them within the sketching area of a

particular CALL system. For written EA languages, this is very important since symbols are not written in isolation. In fact, many words in written EA languages require multiple symbols in order to achieve their meaning. Tackling the difficult challenge of recognizing multiple shapes absent of sketching constraint is significantly alleviated with LADDER, which is accomplished by first constructing a shape description containing the set of multiple shapes. In other words, the components section of the shape description lists the complete symbols that make up the target written EA language word.

While listing the complete symbols that make up the target word is a necessary condition in the shape description, their mere listing is not sufficient enough to complete the shape description. This can easily be seen by drawing these symbols with varying sizes at random locations of a CALL system's sketching area. Therefore, the designer must also include at least three additional conditions in the shape descriptions to provide sufficient recognition of multi-symbol words. These conditions come in the form of listing physical relationships amongst the symbols relative to each other.

- 1) **Relative position.** Unlike the written properties of European languages such as English, written EA languages are more flexible in that they can be written in multiple ways such as left-to-right, right-to-left, and top-to-bottom. Therefore, relative position can be taken into account through the constraints portion of the shape description. After the preferred writing direction is established, the designer should explicitly state where the symbols are physically located relative

to each other. This can be accomplished with the constraints *leftOf*, *rightOf*, *above*, and *below*.

- 2) **Relative size.** The symbols in written EA symbols are inherently contained within a block much like a bounding box of square proportions. Therefore, the shapes that make up a multi-shape written EA language word must have similar-sized bounding boxes. This property can be established by either matching the bounding points of the symbols to the same axis through the constraints *sameX* and *sameY*, or by ensuring that one symbol is contained entirely within the extreme bounding points of another slightly longer or slightly wider symbol.
- 3) **Relative closeness.** This last property exists to ensure that the shapes that make-up a multi-symbol word are grouped within a reasonable space. In other words, the property of relative closeness exists to make sure that symbols of one word in one part of the sketching area do not accidentally get incorrectly recognized with symbols of another word in another part of the sketching area. To achieve this property, the designer can make use of constraints such as the two-argument *near* and the three-argument *closerThan* constraint. The former constraint relies on an absolute pixel distance, while the latter constraint compares the pixel distances against two given shape components.

In essence, the shape descriptions for a multi-symbols word operate on complete symbols analogously to how shape descriptions for a single compound shape operate on simpler shapes and primitive shapes, as well as how shape descriptions for a single

simple shape operate on primitive shapes. A concrete example of a multi-symbol word's shape description can be seen in Figure 7.


		<u>Name:</u>
		JapanKanji
		<u>Components:</u>
DayKanji	day	
BookKanji	book	
<u>Constraints:</u>		
LeftOf	day.rightPoint	book.leftPoint
Not Above	day.topPoint	book.topPoint
Not Below	day.bottomPoint	book.bottomPoint
Near	day.rightPoint	book.leftPoint
<u>Aliases:</u>		
...		

Figure 7. A shape description for the Chinese characters *Japan*.

5.4 Handling Special Cases

The methodology for constructing shape definitions in this thesis so far generalizes reasonably well for the symbols of the various EA writing scripts, especially for polyline-heavy scripts such as Chinese characters. Some scripts though exhibit visual structure properties that may be more challenging to describe with the methodology. In order to address these challenging properties, the methodology was adapted to address the special cases inherent in the writing scripts of interest.

One such example exists within the combined forty-one phonetic and tonal symbols that make up Mandarin Phonetic Symbols I (MPS1), which are the three symbols in the set consisting of a single straight line: the symbol for the phonetic *i* sound, and the symbols for the rising and falling tones (Figure 8). Treated in isolation, these three symbols are trivial for a recognizer to correctly recognize due to their simple property of being a single stroke.

The problem becomes readily apparent when making use of LADDER for recognizing the symbols in MPS1, especially when a sequence of symbols contains one of the three single straight symbols that frequently occur in writing. This is because LADDER employs eager recognition on sketched input; in other words, the moment a user lifts the stylus from the writing surface of the computer screen, LADDER attempts to recognize the stroke after it has been converted to its primitive shape equivalent by matching it to existing shape descriptions in the domain. Therefore, given some symbol in MPS1 which consists entirely of lines, its corresponding shape description would therefore contain entirely of lines in the components section. The issue occurs when at least one of those lines happens to be visually equivalent to one of the three single straight line MPS1 symbols; with eager recognition in LADDER, the multi-line MPS1 symbol will therefore not achieve correct recognition since one of its component lines will prematurely be recognized as a single straight line MPS1 symbol. This is especially problematic since the primitive line equivalents of the three single straight line-based symbols occur so frequently within the multi-stroke MPS1 symbols. A visual example

of this issue which shows a multi-stroke MPS1 symbol containing a single straight line-based MPS1 symbol can be seen below in Figure 9.

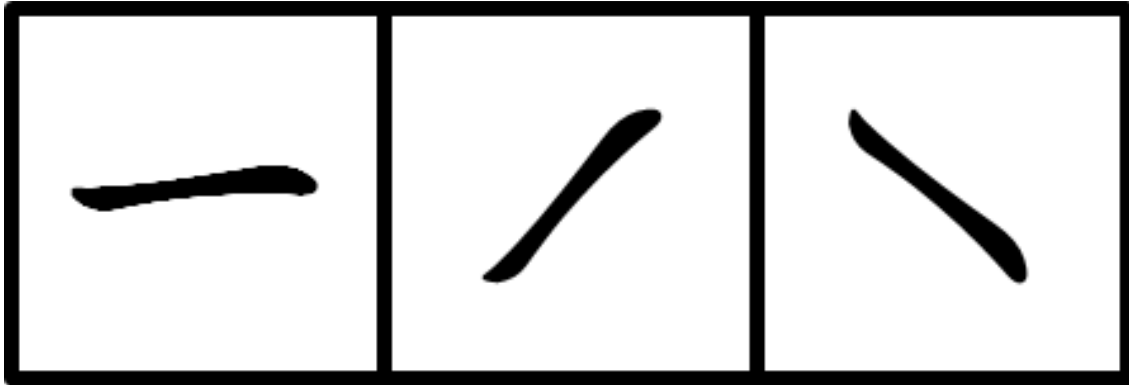


Figure 8. The three single straight line symbols in MPS1: (left) The symbol for the phonetic *i* sound, (middle) the symbol for the rising tone, (right) the symbol for the falling tone.

The problem becomes readily apparent when making use of LADDER for recognizing the symbols in MPS1, especially when a sequence of symbols contains one of the three single straight symbols that frequently occur in writing. This is because LADDER employs eager recognition on sketched input; in other words, the moment the user lifts the stylus from the writing surface of the computer screen, LADDER attempts to recognize the stroke after it has been converted to its primitive shape equivalent by matching it to existing shape descriptions in the domain. Therefore, given some symbol in MPS1 which consists entirely of lines, its corresponding shape description would therefore contain entirely of lines in the components section. The issue occurs when at least one of those lines happens to be visually equivalent to one of the three single straight line MPS1 symbols; with eager recognition in LADDER, the multi-line MPS1 symbol will therefore not achieve correct recognition since one of its component lines

will prematurely be recognized as a single straight line MPS1 symbol. This is especially problematic since the primitive line equivalents of the three single straight line-based symbols occur so frequently within the multi-stroke MPS1 symbols. A visual example of this issue which shows a multi-stroke MPS1 symbol containing a single straight line-based MPS1 symbol can be seen below in Figure 9.

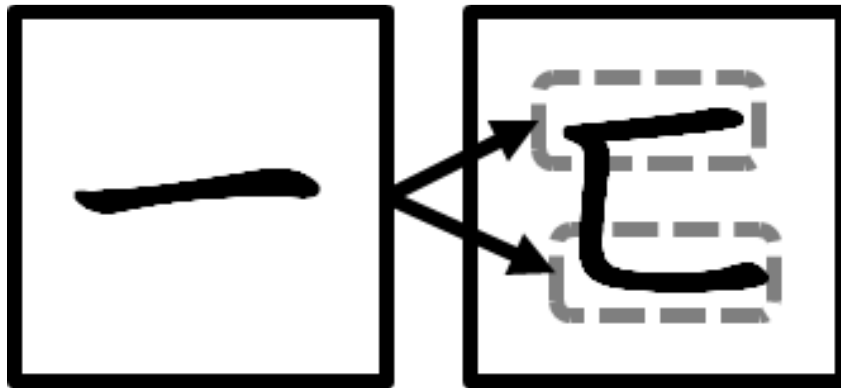


Figure 9. The symbol for the phonetic *i* sound physically contained within the symbol for the phonetic *f* sound in MPS1.

In order to resolve this issue for the domain of MPS1 symbols, the methodology needs to be modified as follows: if a line component within a multi-stroke MPS1 symbols is visually equivalent to an existing single straight line MPS1 symbol, swap that line component with the visually equivalent symbol, such as in Figure 10.

One important consequence regarding this modification to the methodology is that the designer loses the default aliases *p1*, *p2*, and *center* that are given to all primitive line components in LADDER. This consequence can be resolved by redefining those default aliases in those single straight line-based symbols' shape descriptions.

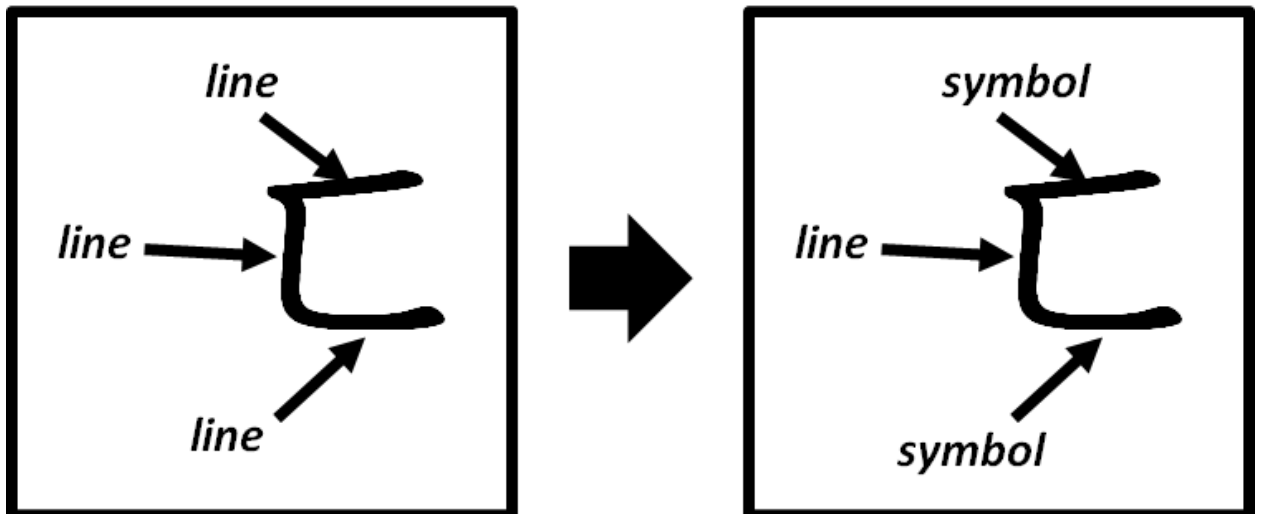


Figure 10. A visual representation of the modified MPS1 methodology for the case of the phonetic *f* sound symbol. The modified methodology swaps the conflicting line components with their visually equivalent MPS1 symbols.

For another EA writing script, the hangul writing script for the Korean language – unlike other EA writing scripts – incorporates one primitive shape that is not found in the other writing scripts: the circular-shaped *ieung* symbol (Figure 11). In the methodology described in this thesis, the assumption was that the symbols in the various scripts of written EA languages were composed of:

- straight lines represented as *line* components,
- curved lines represented as *curve* components, or
- curved lines which could be approximated as *line* components.

In the case of letters (i.e., symbols) in the hangul script, the circular subcomponent exhibits geometric properties that cannot sufficiently be satisfied by the above assumptions. This can be remedied by adding the ellipse primitive shape as a component to represent the circles that are in certain hangul letters.

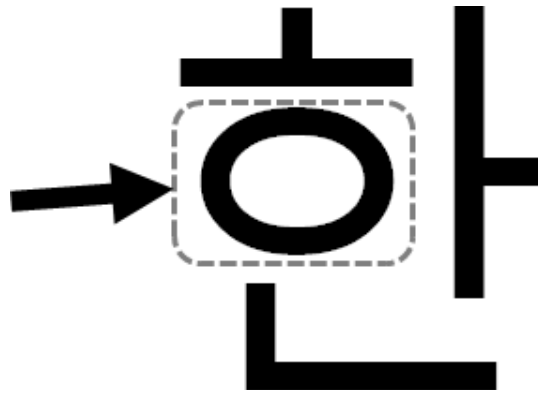


Figure 11. The circular shaped *ieung* symbol in an example hangul letter.

One important property that needs to be noted regarding the circular *ieung* symbol – which is used within certain hangul letters – is that the other polyline components used in those hangul letters lie externally from that circular shape; that is, the lines never intersect the circles. The need for modifying the methodology to thus accommodate ellipse components for representing the *ieung* symbol becomes apparent, since existing default aliases used in the line components are insufficient for constructing shape descriptions for those hangul letters.

The first motivation behind a modified methodology for written Korean relates to the default aliases of endpoints *p1* and *p2* line components, since line endpoints do not have a clear analog in ellipse components due to their closed-shape nature. The second reason relates to the default midpoint alias of *center*; unlike in line components, geometric constraints that interact with this default alias for ellipses must rely on other elliptical geometric properties in order to determine whether a particular part of the shape description refers to either the inside or to the outside of the circular symbol. In other words, utilizing ellipse components requires utilizing different default aliases in

order to determine whether a line component is properly external to an ellipse component.

The solution employed in the modified methodology for written Korean – specifically ellipse components – is to have the constraints interact with another set of default aliases responsible for the bounding points of the ellipse (Figure 12). These bounding points are points that lie specifically at relevant points located on the ellipse’s bounding box.

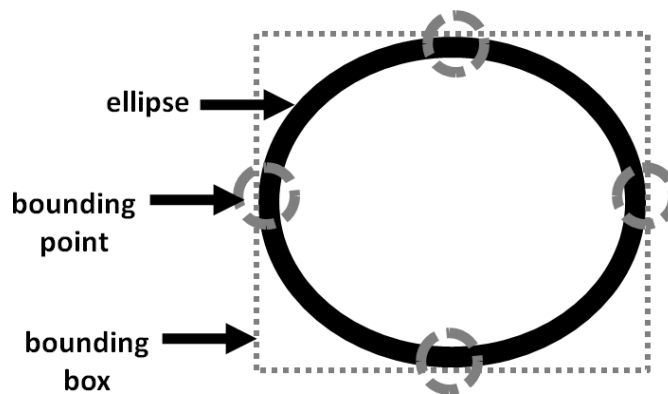


Figure 12. The bounds of an ellipse, including the highlighted bounding points that serve as default aliases for ellipse components in LADDER.

For this modified methodology, the default aliases of interest are the bounding points that lie on an eight-point compass rose of the ellipse component: *boundTopMiddle*, *boundTopRight*, *boundRightMiddle*, *boundBottomRight*, *boundBottomMiddle*, *boundBottomLeft*, and *boundLeftMiddle*, and *boundTopLeft*. With these bounding points, the designer can sufficiently constrain the physical location of an *ieung* symbol within a particular hangul letter relative to surrounding polylines in a LADDER shape description.

6. OVERVIEW OF WRITTEN TECHNIQUE ASSESSMENT

Much like visual technique assessment, assessing the written technique of students' written EA symbols is handled using LADDER. The difference though is that the capability to handle the written technique assessment – which involves a system checking for the correctness of the written symbols' stroke order and direction – requires additional actions that are not directly supported in LADDER. Therefore, supporting this assessment requires an expanded approach of matching part of the LADDER shape descriptions to the labeled segmented strokes' raw temporal information. In other words, the written technique assessment for this methodology occurs after the strokes are segmented and later works in conjunction with the same shape descriptions used in the visual structure assessment. The rest of this section will describe how written technique assessment is handled using this expanded approach.

6.1 Limitations of Assessing Written Technique in LADDER

Besides constraints that are based on the geometric properties of a sketch, LADDER also provides a constraint called *drawOrder* that can check whether correct stroke order was followed by comparing whether one shape was drawn before another. This same constraint can also be used to determine if a shape followed the correct stroke direction by comparing whether one endpoint of a shape was drawn before the other endpoint. From an initial observation, it would appear that LADDER is sufficiently capable of recognizing the correctness of stroke order and stroke direction, two key properties for assessing the written technique of students' written EA symbols. That is,

if a student draws a particular written EA symbol with both correct visual structure and written technique, then LADDER would be able to recognize that symbol as being visually and technically correct.

The limitation of relying on LADDER in assessing students' written technique occurs when the designer wishes for the system to also provide feedback for cases when the student draws a particular symbol visually correct but technically incorrect. If the system relies solely on the *drawOrder* constraint within the shape descriptions, then symbols will not be recognized by LADDER unless the visual technique and the written technique are both correct. In other words, even if the student draws the symbol visually correct, if the written technique is incorrect then the symbol will still be recognized as completely incorrect. This is because LADDER shape descriptions as dictated by the methodology only gives a single feedback of correct or incorrect when two separate feedbacks are needed for the visual structure and written technique. A possible solution to accommodate the needed variable feedback for separate assessment of the visual structure and written technique – while still completely relying on LADDER and its *drawOrder* constraint – can be found in Table 9.

Based on the information in Table 9, designers who wish to provide assessment for both visual structure and written technique entirely using LADDER will encounter issues for the case of when the written symbol has correct visual structure but incorrect written technique; specifically, the problem occurs in the designer needing to construct multiple shape descriptions for each possible incorrect stroke order and direction. While this accommodation does potentially address the issue in providing multiple feedbacks to

suit the situation, the solution is very repetitive and time-consuming on the part of the designer as the number of symbols to recognize and the number of strokes for each of those symbols increase in number.

Table 9. Accommodating multiple feedbacks using solely LADDER and its *drawOrder* constraint for a single symbol.

	Correct Written Technique	Incorrect Written Technique
Correct Visual Structure	Construct a single LADDER shape description that defines the correct visual structure and written technique properties.	Construct multiple shape descriptions that all define the correct visual structure properties, but take into account each possible incorrect stroke
Incorrect Visual Structure	Only provide feedback that the symbol has incorrect visual structure, since written technique is irrelevant for an incorrect symbol.	Only provide feedback that the symbol has incorrect visual structure, since written technique is irrelevant for an incorrect symbol.

6.2 Strokes and Primitive Shapes

Before describing how the correctness of stroke order and direction is checked, it is first important to differentiate between strokes, as it is used in the EA language curriculum; and primitive shapes, as it is used in LADDER. In a conventional EA language curriculum, a stroke is defined in the EA language curriculum as the mark that is made from the moment the writing device makes contact with the surface to the moment that the writing device is lifted from the surface, so stroke order consists of a temporal sequence of these marks. Therefore, stroke order and direction is based on the temporal ordering of the sequence of pen-down to pen-up motions.

In contrast, the methodology alternatively assesses the correctness of the stroke order and direction by the temporal ordering of the segmented primitive shapes when they were originally sketched. The concept of checking for correctness between how an instructor performs the task in the classroom and how a system that implements the methodology for a CALL application is conceptually similar; the difference is that the strokes as defined in the curriculum are treated as segmented primitive shapes in the methodology (Figure 13).

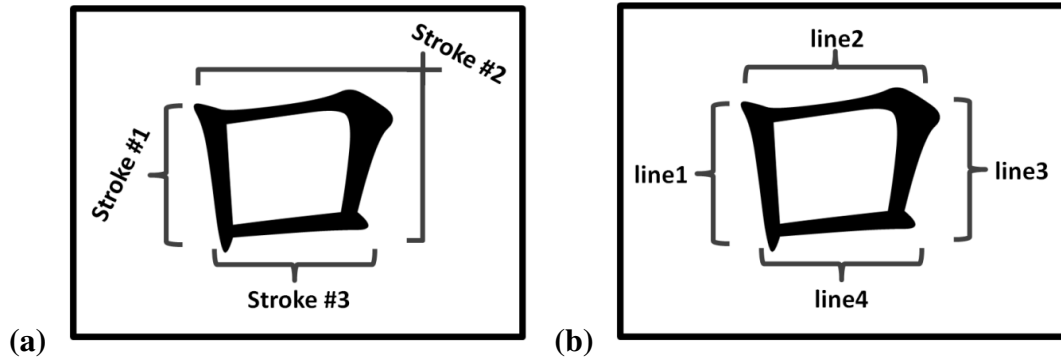


Figure 13. Label comparisons for the Chinese character *mouth*: (a) Enumerated labels for temporal order of strokes in conventional EA language instruction, and (b) enumerated aliases for temporal order of primitive line components for the methodology.

6.3 Customizing Aliases for Simple Symbols

As an alternative to the *drawConstraint* constraint in LADDER, the methodology relies heavily on LADDER's user-created aliases and their additional purpose as labels for segmented primitive shapes in order to handle stroke order and direction correctness. This use of aliases differs from how they are used in the visual structure assessment, where aliases serve as either a way for more complex written EA symbols to access

specific subcomponents of their inner simpler symbol components, or as more intuitive labels for designers to more easily construct shape descriptions. For simple symbols (i.e., symbols composed entirely of primitive shapes), the specific steps for exploiting aliases to handle stroke order correctness are as follows:

- 1) For each primitive shape in the components section of the shape description, create a new corresponding alias for that primitive shape.
- 2) For each new alias created, label that alias as *line#*, where # is the stroke order number of that primitive shape. For example, if a given line is the third stroke in the stroke order, then that line is enumerated as *line3*.

Similarly for direction correctness, the steps are as follows:

- 1) For each primitive shape in the components section of the shape description, create a new corresponding alias for both the starting and ending endpoints.
- 2) For each new alias created for the starting endpoint, label that alias as *start#*, where # is the stroke order number of that primitive shape for which that point belongs to. For example, if a given line is the third stroke in the stroke order, then its associated starting endpoint is enumerated as *start3*.
- 3) Similarly for the ending endpoint, label the alias as *end#*, analogous to what was done in the previous step but for ending endpoints.

An example of the above steps for aliases related to stroke order and direction can be found specifically for the Chinese character *ten* in Figure 14.

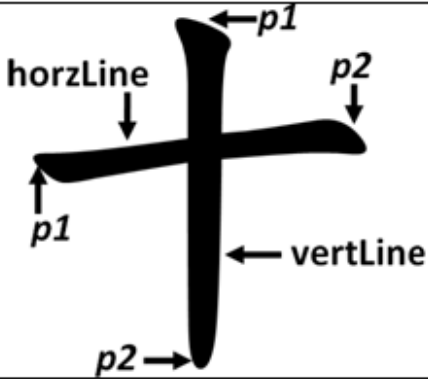
	<u>Name:</u> <u>MouthKanji</u>
	<u>Components:</u> Line <u>horzLine</u> Line <u>vertLine</u>
<u>Constraints:</u> ...	
<u>Aliases:</u> ... Line <u>horzLine</u> line1 Line <u>vertLine</u> line2 Point horzLine.p1 start1 Point horzLine.p2 end1 Point vertLine.p1 start2 Point vertLine.p2 end2	

Figure 14. Partial shape description for the Chinese character *ten* which focuses primarily on the components and also the aliases related to stroke order and direction.

6.4 Customizing Aliases for Compound Symbols

Customizing aliases for simple symbols is a relatively straightforward process, since aliases are created, labeled, and matched to their corresponding primitive shapes and endpoints within the target symbol (Figure 15.a and Figure 15.b).

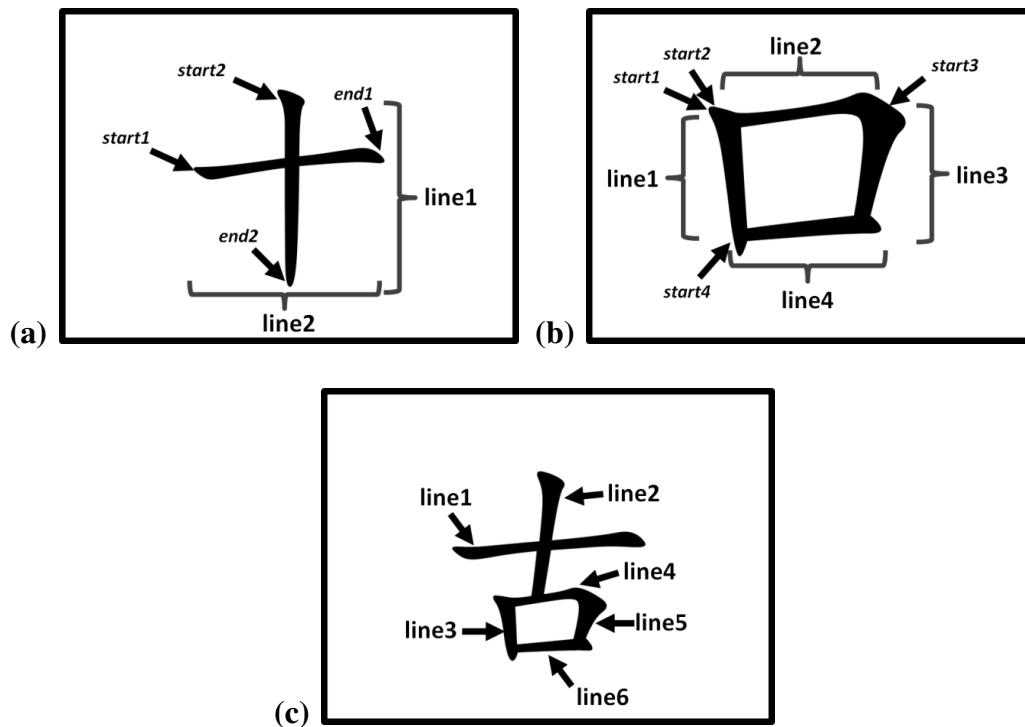


Figure 15. Stroke order and direction labels for a compound and two simple symbols: (a) Stroke order and stroke direction labels for the Chinese character *ten*, (b) stroke order and partial stroke direction labels for the Chinese character *mouth*, and (c) stroke order labels for the Chinese character *ancient*.

While this assumption holds true for simple shapes, aliases need to be customized in a different way when a written symbol is a compound symbol. The core reason is that customizing the aliases to written technique assessment requires that these aliases are tied to their corresponding primitive shapes, which is distinct from the user-created shapes found in compound shapes. The direct solution is to re-label the aliases from the simple symbols so that the stroke order and direction are maintained for the compound symbol (Figure 15.c).

6.5 Assessing the Written Technique Using Aliases

Once the aliases have been customized to their corresponding primitive shapes and their endpoints, assessing the correctness of the written EA symbol's stroke order and direction is a straightforward matter of retrieving the raw temporal values of the segmented strokes retrieved from those custom aliases. The following steps summarize the prior steps that have been undertaken before written technique assessment in this methodology occurs:

- 1) **Collect the data.** Prior to segmenting the strokes for use in LADDER, written data is collected from the stylus in the form of spatial (i.e., x- and y-coordinates) and temporal data.
- 2) **Segment the strokes.** The strokes are segmented using specialized algorithms into primitive shapes, which also contain the temporal data associated with those segmented spatial data.
- 3) **Create custom aliases.** Aliases are created for each primitive shape to denote their order and direction in the strokes.

Once custom aliases have been created, the next step is to perform the assessment on the written technique. This step assumes that the written EA symbol has already been successfully recognized as having correct visual structure. The reasoning behind this is that if a symbol is visually incorrect (e.g., the symbol was drawn correctly but was not the symbol that was prompted, the symbol's visual structure contains visual errors), then the drawing's written technique will not be relevant since it is not related to the

prompted symbol's correct visual structure. The steps below first describe how the custom aliases coordinate with the temporal data to assess the stroke order, subsequently followed by stroke direction assessment. These two steps assume that the visual structure is also correct, and additionally the check for correct stroke direction is optional if the stroke order check fails.

- **Checking for correct stroke order:**

- 1) The methodology first retrieves the customized aliases from the alias section of the written symbol's associated shape description, and then stores those aliases in a list.
- 2) Next, the aliases are used to reference their corresponding primitive shape from the segmented data.
- 3) Afterwards, the raw temporal data of the primitive shapes are extracted and used to order the corresponding aliases in temporal order.
- 4) Finally, since the aliases for stroke order begin with *line* and are enumerated, the enumerated value for each alias is checked for correct ascending order. If the numbers are ordered correctly in ascension, then the methodology denotes the stroke order as correct. Otherwise, it is incorrect.

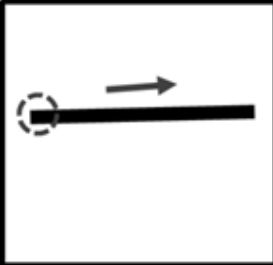
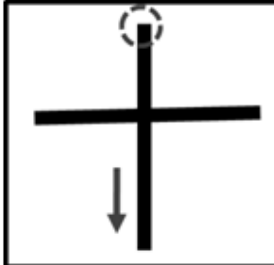

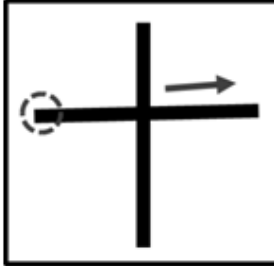
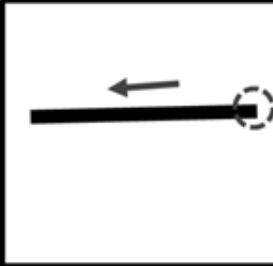
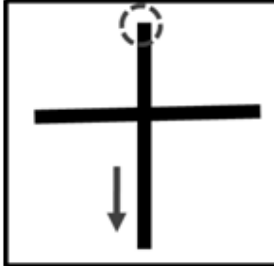
- **Checking for correct stroke direction:**

- 1) Referencing the list of primitive shapes extracted for the stroke direction check, first iterate through this list of primitive shapes and retrieve the first point sketched temporally for that primitive shape.

- 2) Next, compare each primitive shape's endpoints and determine which endpoint lies closest to the first initially sketched point of that primitive line. This step is necessary since the first point in the sequence of raw points for a primitive shape may be slightly different from the aliased endpoint of that primitive shape due to how segmentation is performed.
- 3) Lastly, retrieve the equivalent custom alias that corresponds to that initially sketched point. That custom alias is a label that may either begin with the label *start* or *end*. If it begins with *start*, then continue to the next primitive shape. Otherwise, that label starts with the label *end*, and therefore the result of the student's written EA symbol is incorrect stroke direction since the incorrect endpoint was sketched first.
- 4) If the entire list of primitive strokes has been iterated through, and if each custom alias specific to stroke direction assessment begins with the label *start*, then the result is that the stroke direction is correct.

After assessing the stroke order and direction using the above steps, results for both the current written technique and the previous visual structure can then be used for CALL applications to display the results as seen fit by the application designer. A visual example of the written technique assessment that utilizes the custom aliases can be found in Table 10.

Table 10. Assessing the written technique of three different writing styles for the Chinese character *ten* based on their custom aliases for stroke order and direction.

Stroke #1	Stroke #2	Result																
 <table border="1" data-bbox="289 674 607 779"> <thead> <tr> <th>Order</th> <th>Direction</th> </tr> </thead> <tbody> <tr> <td>line1</td> <td>start1</td> </tr> <tr> <td></td> <td></td> </tr> </tbody> </table>	Order	Direction	line1	start1			 <table border="1" data-bbox="651 674 969 779"> <thead> <tr> <th>Order</th> <th>Direction</th> </tr> </thead> <tbody> <tr> <td>line1</td> <td>start2</td> </tr> <tr> <td>line2</td> <td>start2</td> </tr> </tbody> </table>	Order	Direction	line1	start2	line2	start2	<table border="1" data-bbox="1008 541 1388 617"> <tbody> <tr> <td><input checked="" type="checkbox"/></td> <td>Stroke Order</td> </tr> <tr> <td><input checked="" type="checkbox"/></td> <td>Stroke Direction</td> </tr> </tbody> </table>	<input checked="" type="checkbox"/>	Stroke Order	<input checked="" type="checkbox"/>	Stroke Direction
Order	Direction																	
line1	start1																	
Order	Direction																	
line1	start2																	
line2	start2																	
<input checked="" type="checkbox"/>	Stroke Order																	
<input checked="" type="checkbox"/>	Stroke Direction																	
 <table border="1" data-bbox="289 1092 607 1197"> <thead> <tr> <th>Order</th> <th>Direction</th> </tr> </thead> <tbody> <tr> <td>line2</td> <td>start2</td> </tr> <tr> <td></td> <td></td> </tr> </tbody> </table>	Order	Direction	line2	start2			 <table border="1" data-bbox="651 1092 969 1197"> <thead> <tr> <th>Order</th> <th>Direction</th> </tr> </thead> <tbody> <tr> <td>line2</td> <td>start2</td> </tr> <tr> <td>line1</td> <td>start1</td> </tr> </tbody> </table>	Order	Direction	line2	start2	line1	start1	<table border="1" data-bbox="1008 957 1388 1033"> <tbody> <tr> <td><input checked="" type="checkbox"/></td> <td>Stroke Order</td> </tr> <tr> <td><input checked="" type="checkbox"/></td> <td>Stroke Direction</td> </tr> </tbody> </table>	<input checked="" type="checkbox"/>	Stroke Order	<input checked="" type="checkbox"/>	Stroke Direction
Order	Direction																	
line2	start2																	
Order	Direction																	
line2	start2																	
line1	start1																	
<input checked="" type="checkbox"/>	Stroke Order																	
<input checked="" type="checkbox"/>	Stroke Direction																	
 <table border="1" data-bbox="289 1507 607 1612"> <thead> <tr> <th>Order</th> <th>Direction</th> </tr> </thead> <tbody> <tr> <td>line1</td> <td>end2</td> </tr> <tr> <td></td> <td></td> </tr> </tbody> </table>	Order	Direction	line1	end2			 <table border="1" data-bbox="651 1507 969 1612"> <thead> <tr> <th>Order</th> <th>Direction</th> </tr> </thead> <tbody> <tr> <td>line1</td> <td>end2</td> </tr> <tr> <td>line2</td> <td>start1</td> </tr> </tbody> </table>	Order	Direction	line1	end2	line2	start1	<table border="1" data-bbox="1008 1373 1388 1449"> <tbody> <tr> <td><input checked="" type="checkbox"/></td> <td>Stroke Order</td> </tr> <tr> <td><input checked="" type="checkbox"/></td> <td>Stroke Direction</td> </tr> </tbody> </table>	<input checked="" type="checkbox"/>	Stroke Order	<input checked="" type="checkbox"/>	Stroke Direction
Order	Direction																	
line1	end2																	
Order	Direction																	
line1	end2																	
line2	start1																	
<input checked="" type="checkbox"/>	Stroke Order																	
<input checked="" type="checkbox"/>	Stroke Direction																	

7. IMPLEMENTED APPLICATIONS AND EVALUATION

The primary motivation in creating the methodology is so that it could be implemented into a CALL system tailored specifically for written EA languages. In this section, CALL systems were implemented for the following two distinct writing scripts to showcase the range and depth that the methodology can achieve:

- **The *kanji* script.** The non-phonetic Chinese characters specific to written Japanese.
- **The *Mandarin Phonetic Symbols I (MPSI)* script.** The phonetic symbols specific to representing the Mandarin sounds in written Chinese.

The rest of this section will introduce the capabilities and evaluate the effectiveness of those different CALL systems.

7.1 Hashigo: A CALL System for Handwritten Japanese Kanji

The Hashigo system [30], which was the first completed system to implement the methodology, is a CALL system developed specifically for the instruction of the Japanese kanji script (Figure 16). In addition, Hashigo's interface successfully adopts key features in the methodology, specifically: free-sketch input, paper-like interface, digital capabilities, and emulated teacher feedback (Figure 17). This fully operational learning tool – which provides a graphical-user interface (GUI) over the recognition provided in the methodology – follows the instructional techniques established in a Japanese language textbook [7] by prompting users to sketch the kanji and their elements

(i.e., the corresponding simpler kanji contained within them) that is introduced in that textbook's chapters.

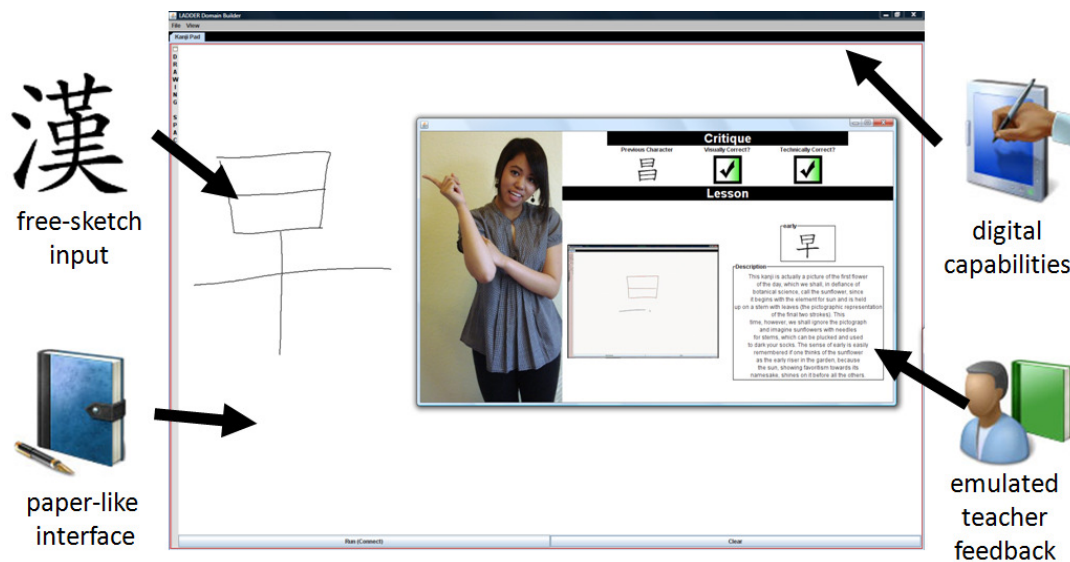


Figure 16. An overview of the Hashigo GUI, incorporating four of the key features in the methodology: free-sketch input, paper-like interface, digital capabilities, and emulated teacher feedback.

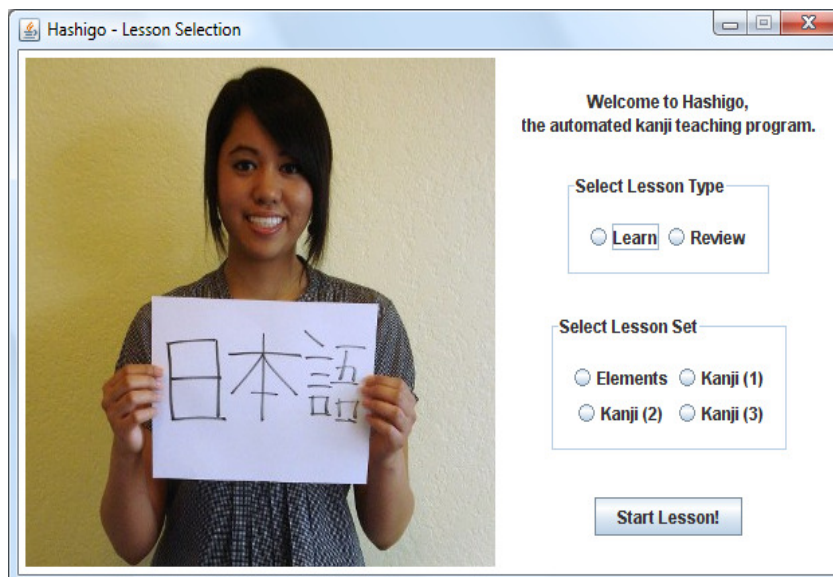


Figure 17. The Hashigo selection window for choosing the lesson type and kanji or element set.

In the Hashigo system, a type of review setup was created for three of the chapters in the source textbook, but the system can easily be expanded to include additional chapters in future iterations. Upon usage of the CALL application, users are initially given the following two choices (Figure 17):

- **Learn mode:** Before drawing a new kanji or element, the user is shown an animation of how to draw it, a textual hint to help in memorization, and an assessment of the visual structure and written technique of the previous kanji or element prompted, if there was one (Figure 18).

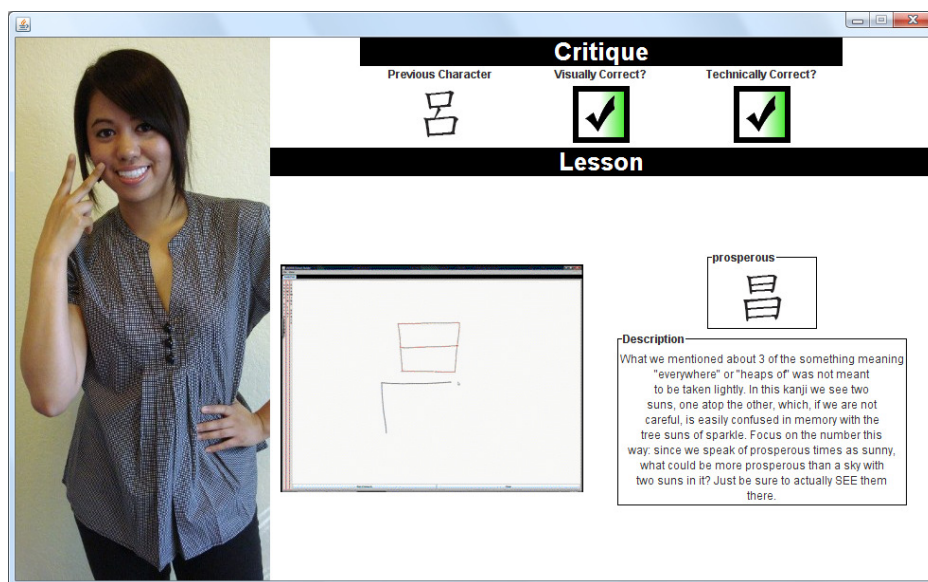


Figure 18. The instruction window for the Learn Mode in Hashigo for an individual symbol.

- **Review mode:** Unlike in Learn mode, this mode initially prompts the user to draw a particular kanji or element given its English translation. After the user

confirms completion of the sketch, the application then provides one of three possible feedbacks based on the user's performance:

- 1) **Correct visual structure and written technique.** The user is congratulated for achieving correctness on the kanji or element.
- 2) **Correct visual structure, incorrect written technique.** The user is informed on the correct visual structure. In addition, the user also receives feedback on what aspect of the written technique was incorrect, as well as a reminder animation of how it is drawn (Figure 19).
- 3) **Incorrect visual structure.** The user is informed on the incorrect visual structure and written technique, and receives remedial feedback in the form of a reminder animation and textual hint.

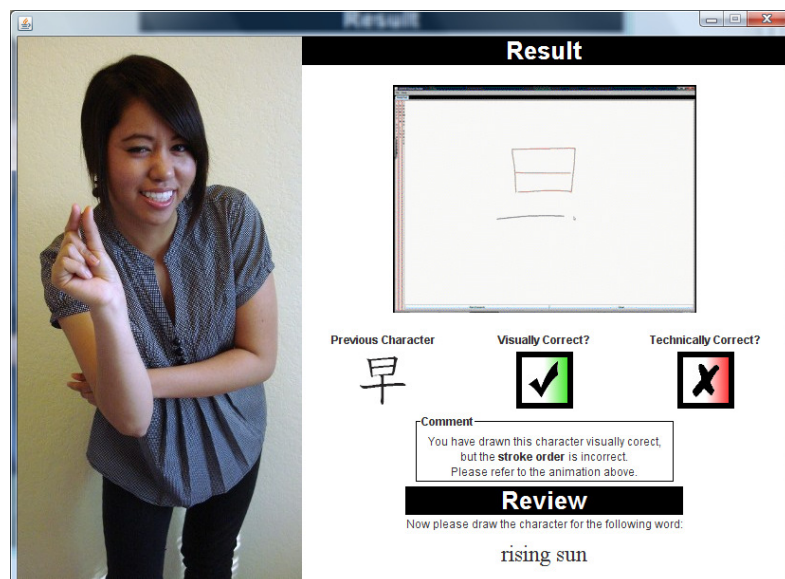


Figure 19. The result window for the Review mode in Hashigo for an individual symbol.

Upon completion of a single lesson, regardless of whether the user selected Learn or Review mode, the user receives a progress report card that reviews the kanji or elements tested, as well as the user's performance in drawing them (Figure 20).



Figure 20. The final progress report window after completing a lesson in Hashigo.

In order to gauge the technical performance of Hashigo as a CALL system for handwritten kanji, a series of three user studies were performed. The first evaluation focused on its visual structure assessment capabilities, where a user study comprised of eleven international graduate students from Texas A&M University proficient in kanji were asked to write a total of nineteen kanji from a specific chapter twice. Since model kanji to be used for teaching students the correct way to write was desired for the system, the only requirement given to the participants was that they write the kanji as if though they were teaching someone not familiar with them. The result of this user study was that the system correctly classified 92.9% of the provided kanji. The entire data from the user study was later used to tweak the shape descriptions so that natural handwritten

kanji were reflected in the system. Accuracies from existing online kanji recognizers in [17] ranged 85% to 95%; therefore, the accuracy given by Hashigo is comparable to those recognizers when recognizing expert users' handwritten kanji.

The second evaluation focused on the written technique capabilities in Hashigo, which involved determining whether the system could properly differentiate written technique factors like a human instructor. The corresponding user study consisted of five non-East Asian students from the graduate school at Texas A&M University with no prior knowledge of kanji writing. An initial user study was run on the participants by asking them to write seven prompted kanji from a given kanji lesson, providing them with no further instruction on how to draw these kanji other than their visual structure. When this initial handwritten data was provided to the system, Hashigo generated 98.6% accuracy on the visual structure. This rise in accuracy was attributed to the higher care that novice participants took in drawing the kanji exactly as presented, in contrast to their expert counterparts, whom may have taken less care and whose previous writing habits may have biased their visual structure. In terms of written technique recognition, it was first noted that all novice participants only gave correct written technique for 5.7% of the visually-correct recognized kanji, which solidified the necessity of a sketch-based CALL system for teaching correct written technique. Secondly, the system perfectly differentiated those kanji with correct written technique from incorrect ones; that is, Hashigo achieved 100% accuracy for written technique correctness.

Lastly, a user study was created to evaluate the viability of Hashigo as a learning tool. The same novice users from the previous user study were asked to use Hashigo

three times (i.e., preview, learn, and review) for a given lesson. After their third use of Hashigo, the final user study was conducted by collecting handwriting samples from the participants to gauge their kanji comprehension performance. After running through this last set of data, the novice users scored 100% accuracy on visual correctness and 97.1% accuracy on written technique correctness. This is a significant improvement of 5.7% in written technique correctness by the same user participants prior to using Hashigo.

7.2 LAMPS: A CALL System for Handwritten Mandarin Phonetic Symbols I

A similar system to Hashigo was developed exclusively for Mandarin Phonetic Symbols called **L**anguage **A**ssistance for **M**andarin **P**honetic **S**ymbols I (LAMPS) [31] (Figure 21). The latest iteration of the system tests students on their knowledge of MPS1 symbols based on the vocabulary that is covered in the first chapter of [5] a textbook used by several language programs in Taiwan.

At the start of running the system, the user is prompted with two additional windows that appear on the right side of the screen (Figure 22). The top-right window informs the user of the next Chinese words to write the sequence of phonetic symbols for (Figure 22.c), while the bottom window provides a visual structure and written technique assessment for the previous sequence of phonetic symbols for the previous Chinese word (Figure 22.d). This latter window provides the previously prompted word, the target pronunciation(s), and the assessment. If the user wrote a certain symbol incorrect, LAMPS also provides a visual guide on how to correctly write that symbol.

Similar to the evaluation conducted on the Hashigo system, the LAMPS system was evaluated on its effectiveness as a CALL system for the instruction of MPS1. The first conducted user study evaluated the recognition rates of LAMPS for its visual structure assessment. Ideally, the system should sufficiently recognize the handwritten symbols of expert MPS1 writers based on the constructed shape descriptions, since the objective of the dynamic workbook interface for MPS1 is for students to eventually emulate the visual structure writing made by these native writers. For this user study, nine Taiwanese graduate students at Texas A&M University with proficient MPS1 writing knowledge were recruited, with the additional prerequisite that they write the symbols as if though they were teaching someone not familiar with them. This was desired since casual writing was not representative of the type of model writing that was desired to base the system's recognition on as a pedagogical tool. The users were each asked to write the MPS1 symbols twice, which were later evaluated for visual correctness on an all-or-nothing recognition metric; in other words, the correctness of a written symbol in LAMPS is counted only if the entire symbol is correctly recognized.

The result of this first user study was that LAMPS attained 95% accuracy on the visual structure of the study participants' handwritten input, where the expert writers wrote the correct symbol and that the misrecognized symbols were considered too sloppily drawn for LAMPS to recognize. Since the system performance of similar online handwritten recognizers in the domain achieved accuracy within the range of 85% to above 95% [17], the conclusion that was reached was that the system attained sufficient recognition comparable to other recognizers. Since the system also does not

rely on the use of training data to improve the recognition in LAMPS compared to traditional machine learning techniques (e.g., neural networks, hidden Markov models), the recognition in LAMPS was generalized further by tweaking the shape descriptions further to reflect the writing styles on the expert writers' model handwritten input .

LAMPS was then evaluated again to determine whether it would be able to adequately recognize the correctness of novice users' sketched MPS1 symbols based on the visual structure and stroke order. A second user was conducted to collect handwritten symbols from a second group, this time consisting of five American university students from Texas A&M University with no knowledge of East Asian writing. Like in the previous in the previous user study, each participant was asked to write each symbol twice.

Since these latter participants had no knowledge of MPS1, the symbols were visually prompted for them. The eventual result of this user study yielded 100% accuracy on visual structure recognition. This higher accuracy rate was attributed to the American students writing the symbols more carefully and with less variation than the native writers, which is the type of learning behavior that was expected from novice students learning MPS1. In addition, for each symbol from this set of collected handwritten data that was recognized correctly for visual correctness, the system was also able to correctly assess the written technique with an accuracy of 100%. Following the user study, the user study participants were then asked to make slight errors to both the visual structure and the written technique of the symbols, and the system also succeeded in identifying those errors during the assessment (Figure 23). The accuracy

of correct assessment on written technique for these symbols is comparable to that found in the Hashigo system described in the previous section.

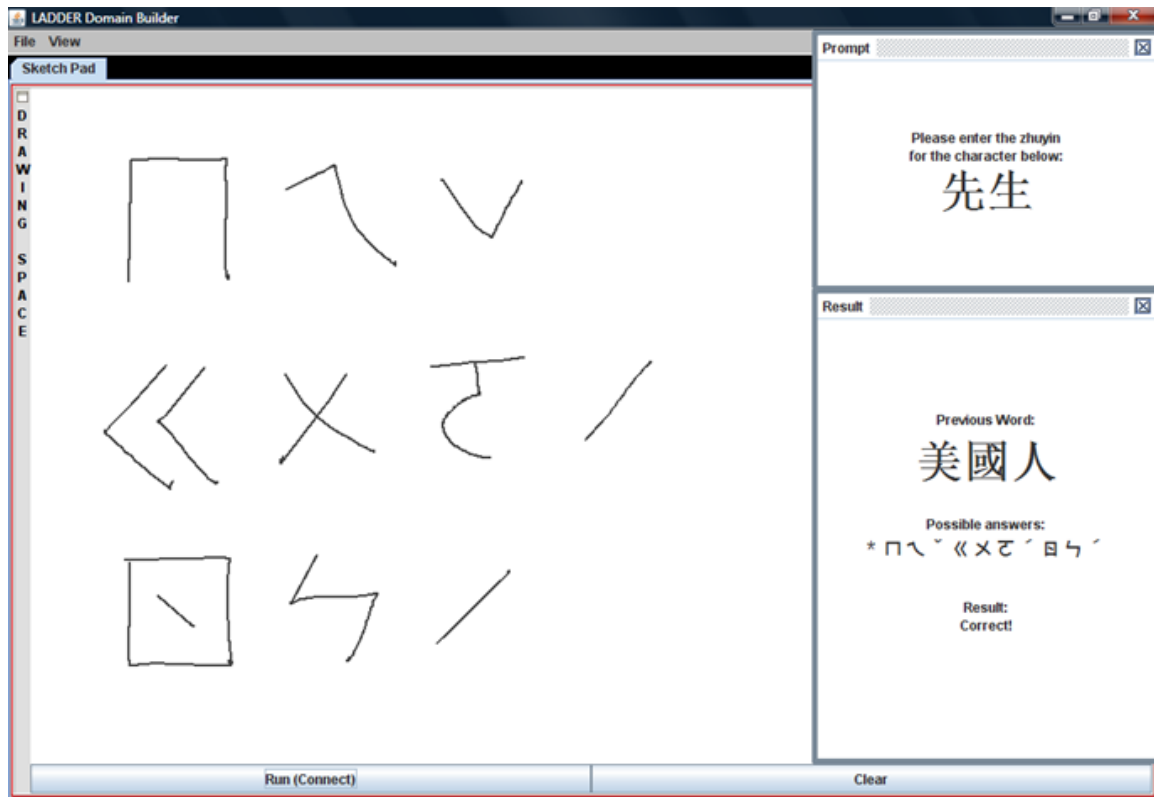


Figure 21. A screenshot of LAMPS during regular operation.

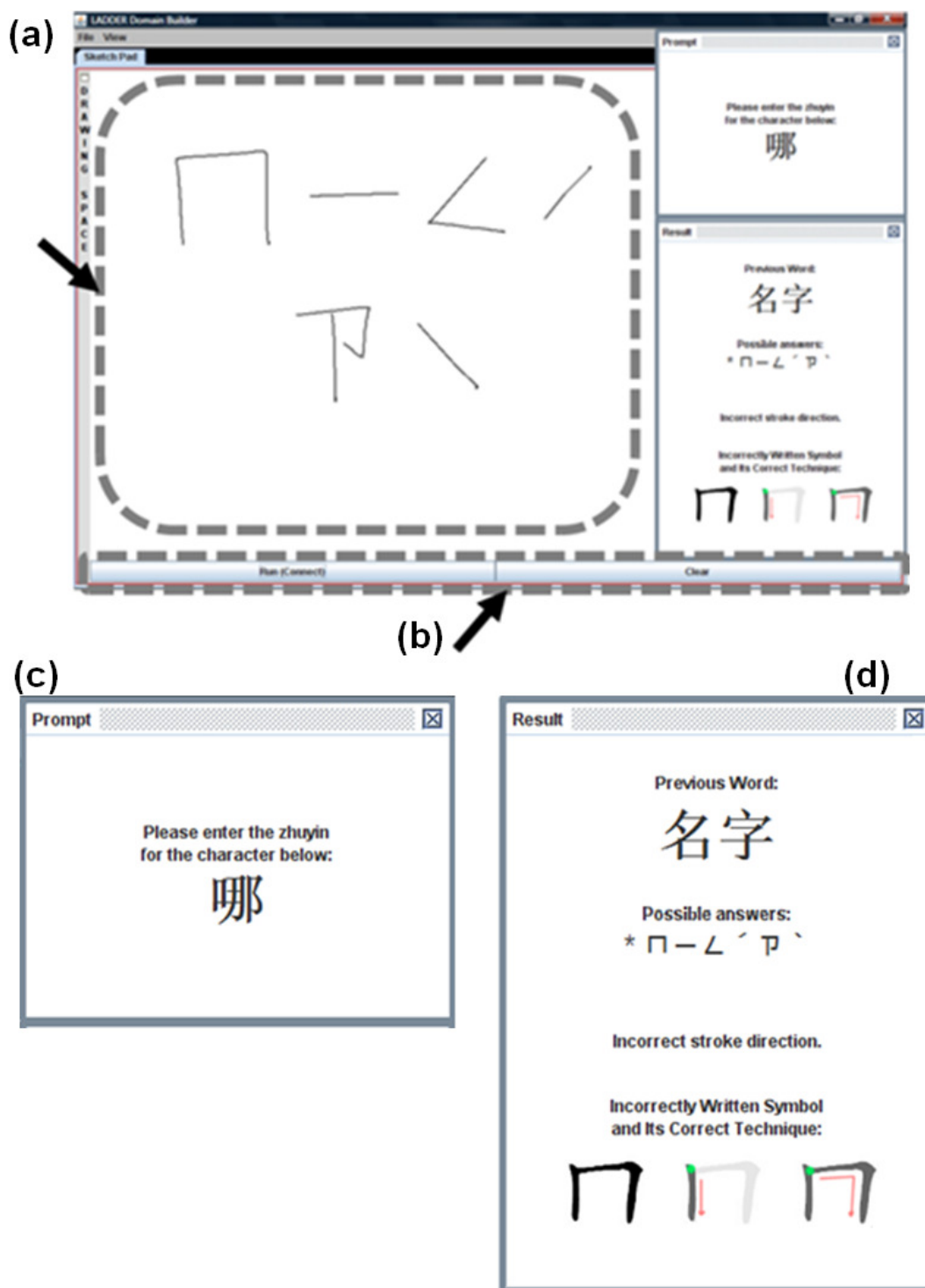


Figure 22. A screenshot of LAMPS when the user writes a symbol in a visually correct sequence of symbols with an incorrect written technique: (a) drawing panel, (b) buttons to run assessment and clear panel, (c) prompt with next MPS1 symbol to draw, and (d) result.

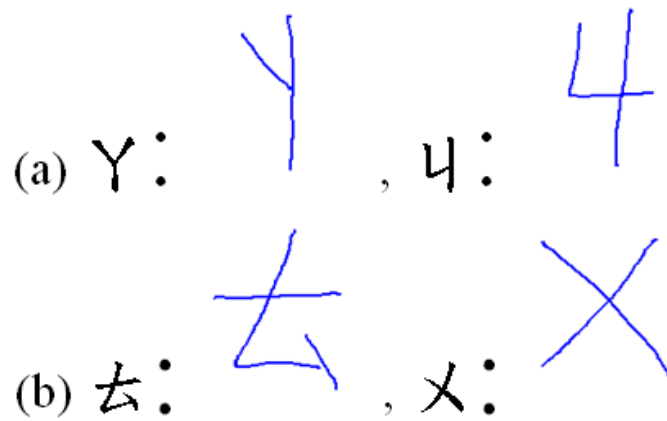


Figure 23. Samples of user-sketched MPS1 symbols accurately recognized as having: (a) incorrect visual structure and (b) correct visual structure with incorrect written technique.

8. SUMMARY

Recognizing language students' multi-symbol handwriting from written EA languages in a free-sketch, paper-and-pen-emulating environment is a challenging task. So is emulating human instructor feedback assessment on that sketched input's visual structure and written technique to be displayed back to the student. While existing techniques and algorithms have succeeded in partially or completely achieving those functionalities digitally, combining those concepts into a single CALL system has been difficult to realize. In this thesis, a methodology was described that allowed both tasks to be combined for use in innovative forms of CALL systems specific to written EA languages. The methodology first received the handwritten input from students and used cutting edge high-performing algorithms to segment those strokes into primitive shapes. Afterwards, an approach was devised using the LADDER sketching language and raw temporal data that allowed shape descriptions in LADDER to effectively recognize students' written EA symbols based on those primitive shapes. Lastly, the methodology used the recognition results derived from the employed sketch recognition tools and collected raw sketching data to assess the students' visual structure and written technique. In addition to devising the methodology, two systems were developed that implemented the methodology for use in two distinct EA writing scripts: Hashigo for Japanese kanji and LAMPS for Mandarin Phonetics Symbols I. After conducting user studies for both systems involving native and novice writers of those written EA scripts, the result was that both systems were able to achieve reasonable assessment accuracy.

8.1 Expanding on This Methodology

The version of the methodology described in this thesis is designed to handle students' handwritten input for writing scripts of EA languages. The current iteration can especially achieve recognition and provide assessment on input for writing scripts whose symbols can be described or reasonably approximated with primitive shapes (i.e., lines, curves, ellipses) that are available from the employed corner-finding algorithms. Of the writing scripts in the EA languages, one writing script that presents unique challenges to current corner-finding algorithms is the hiragana script of written Japanese. The reason is that the visual structures of many symbols in the hiragana script contain variable curves that are non-trivial to describe with the available primitive shapes at the designer's disposal. In order to accommodate the unique visual structures that these symbols possess, the employed corner-finding algorithms would need to be expanded such that they contain primitive shapes which could better describe those visual structures.

This methodology was also catered to handle symbols from a variety of writing scripts in the written EA languages of Chinese, Japanese, and Korean. Two such CALL systems incorporated the methodology which covered writing scripts from written Japanese and Chinese with reasonable results, and it would be desirable to further introduce additional fully-functional CALL systems for other writing scripts that this methodology can currently (e.g., the hangul script for written Korean, the katakana script for written Japanese) or will eventually (e.g., the hiragana script for written Japanese) support.

Lastly, the focus of the methodology was on the technical capabilities from the perspective of both human-computer interaction (e.g., paper-like interface input), artificial intelligence (e.g., written EA language recognition), and a hybrid of the two (e.g., free-sketch recognition). While the merits of the methodology's technical capabilities have already been evaluated, what has not been as fully evaluated are its pedagogical capabilities; in other words, the merits of CALL systems that incorporate the methodology in an EA language curriculum. Therefore, expanding the methodology to sufficiently address the pedagogical needs of an EA language curriculum is another desirable direction for this research. This may be accomplished by working closely with language instructors on what aspects of CALL systems would be desirable to further complement the instructors' lesson plans, and expanding these CALL systems to include additional content for longer-term instruction.

REFERENCES

- [1] M.E. Everson, "Literacy Development in Chinese as a Foreign Language," *Teaching Chinese as a Foreign Language: Theories and Applications*, M.E. Everson and Y. Xiao, eds., pp. 97-111, Cheng & Tsui, 2009.
- [2] E. Banno, Y. Ohno, Y. Sakane, and C. Shinagawa, *Genki I: An Integrated Course in Elementary Japanese I*, The Japan Times, 1999.
- [3] K. Takezaki and B. Godin, *An Introduction to Japanese Kanji Calligraphy*, B. Godin, ed., Tuttle Publishing, 2008.
- [4] T. Xie, and T. Yao, "Technology in Chinese Language Teaching and Learning," *Teaching Chinese as a Foreign Language: Theories and Applications*, M.E. Everson and Y. Xiao, eds., pp. 151-172, Cheng & Tsui, 2009.
- [5] National Taiwan Normal University, *Practical Audio-Visual Chinese 1 Textbook, Vol. 1*, Cheng Chung Book Company, Ltd., 2004.
- [6] W. McNaughton and L. Ying, *Reading & Writing Chinese: Traditional Character Edition*, Tuttle Publishing, 1999.
- [7] J. Heisig, *Remember the Kanji I: A Complete Course on How Not to Forget the Meaning and Writing of Japanese Characters*, University of Hawai'i Press, 2007.
- [8] D.O. Pyun and I.-S. Kim, *Colloquial Korean: The Complete Course for Beginners*, Routledge, 2010.
- [9] J. DeFrancis, *The Chinese Language: Fact and Fantasy*, University of Hawaii Press, 1984.
- [10] Y.-Y. Yang, "Adaptive Recognition of Chinese Characters: Imitation of Psychological Process in Machine Recognition," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 28, no. 3, pp. 253-265, 1998.
- [11] N. Lin, S. Kajita, and K. Mase, "A Multi-Modal Mobile Device for Learning Japanese Kanji Characters through Mnemonic Stories," *Proc. 9th International Conference on Multimodal Interfaces*, pp. 335-338, 2007.
- [12] K. Lunde, *CJKV Information Processing: Chinese, Japanese, Korean & Vietnamese Computing*, 2009.

- [13] C. L. Willis and L. Miertschin, "Tablet PC's as Instructional Tools or The Pen is Mightier than the 'Board'," *Proc. Fifth Conference on Information Technology Education*, pp. 153-159, 2004.
- [14] A. van Dam, S. Becker, and R. M. Simpson, "Next-Generation Educational Software: Why We Need It and a Research Agenda for Getting It," *EDUCAUSE Review*, vol. 40, no. 2, pp. 26-43, 2005.
- [15] G.-S. Chen, Y.-D. Jheng, and L.-F. Lin, "Computer-based Assessment for the Stroke Order of Chinese Characters Writing," *Proc. Second International Conference on Innovative Computing, Information and Control*, pp. 160-164, 2007.
- [16] G.-S. Chen, Y.-D. Jheng, H.-C. Yao, H.-C. Liu, "Stroke Order Computer-Based Assessment with Fuzzy Measure Scoring," *WSEAS Transactions on Information Science and Applications*, vol. 5, no. 2, pp. 62-68, 2008.
- [17] C.-L. Liu, S. Jaeger, and M. Nakagawa, "Online Recognition of Chinese Characters: The State-of-the-Art," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, pp. 198-213, 2004.
- [18] M. Nakagawa, K. Akiyama, T. Oguni, and N. Kato, "Handwriting-Based User Interfaces Employing On-Line Handwriting Recognition," *Advances in Handwriting Recognition (Series in Machine Perception and Artificial Intelligence Volume 34)*, S.-W. Lee, ed., pp. 578-587, World Scientific, 1999.
- [19] M.F. Zafar, D. Mohamad, and R.M. Othman, "On-line Handwritten Character Recognition: An Implementation of Counterpropagation Neural Net," *World Academy of Science Engineering and Technology*, vol. 10, pp. 232-237, 2005.
- [20] Y. Li, H. Yang, J. Xu, W. He and J. Fan, "Chinese Character Recognition Method Based on Multi-Features and Parallel Neural Network Computation", *Proc. Intelligent Computing 3rd international Conference on Advanced Intelligent Computing Theories and Applications*, pp. 1103-1111, 2007.
- [21] J. Pittman, "Handwriting Recognition: Tablet PC Text Input," *Computer*, vol. 40, no. 9, pp. 49-54, 2007.
- [22] L. Ma, Q. Huo, and Y. Shi, "A Study of Feature Design for Online Handwritten Chinese Character Recognition based on Continuous-Density Hidden Markov Models," *Proc. 10th International Conference on Document Analysis and Recognition*, pp. 526-530, 2009.

- [23] H.C. Lam, K.H. Pun, S.T. Leung, S.K. Tse, and W.W. Ki, "Computer-Assisted Learning for Learning Chinese Characters," *Communications of COLIPS - An International Journal of the Chinese and Oriental Languages*, vol. 3, pp. 31-44, 1993.
- [24] S. Fujita, K. Omae, C. Lin, Y. Ming, and S. Narita, "Kanji Learning System Based on the Letter Shape Recognition Method," *Proc. International Conference on Computers in Education*, pp. 85-89, 2002.
- [25] K. Sun, S. Hsu, and Y. Chen, "An Intelligent Tutoring System for Teaching the Stroke Orders of Chinese Characters on the Internet," *Proc. 1998 International Computer Symposium, Workshop on Computer Networks, Internet, and Multimedia*, pp. 199-206, 1998.
- [26] K. Sun and D. Feng, "A Distance Learning System for Teaching the Writing of Chinese Characters over the Internet," *International Journal of Distance Education Technologies*, vol. 2, pp. 52-66, 2004.
- [27] H.-N. Qi, "A Size-Independent Method for Chinese Character Writing Structure Assessment," *Proc. 2008 International Conference on Machine Learning and Cybernetics*, pp. 2754-2759, 2008.
- [28] F. Tian, F. Lv, J. Wang, H. Wang, W. Luo, M. Kam, V. Setlur, G. Dai, and J. Canny, "Let's Play Chinese Characters: Mobile Learning Approaches via Culturally Inspired Group Games," *Proc. 28th International Conference on Human Factors in Computing Systems*, pp. 1603-1612, 2010.
- [29] J.O. Wobbrock, A.D. Wilson, and Y. Li, "Gestures without Libraries, Toolkits or Training: A \$1 Recognizer for User Interface Prototypes," *Proc. 20th Annual ACM Symposium on User Interface Software and Technology*, pp. 169-168, 2007.
- [30] P. Taelle and T. Hammond, "Hashigo: A Next-Generation Sketch Interactive System for Japanese Kanji," *Proc. Twenty-First Innovative Applications of Artificial Intelligence Conference*, pp. 153-158, 2009.
- [31] P. Taelle and T. Hammond, "LAMPS: A Sketch Recognition-Based Teaching Tool for Mandarin Phonetic Symbols I," *Journal of Visual Languages and Computing*, vol. 21, no. 2, pp. 109-120, 2010.
- [32] B. Paulson and T. Hammond, "Paleosketch: Accurate Primitive Sketch Recognition and Beautification," *Proc. 13th International Conference on Intelligent User Interfaces*, pp. 1-10, 2008.

- [33] T. M. Sezgin, T. Stahovich, and R. Davis, "Sketch Based Interfaces: Early Processing for Sketch Understanding," *Proc. 2001 Workshop on Perceptive User Interfaces*, pp. 1-8, 2001.
- [34] T. Hammond, "LADDER: A Perceptually-Based Language to Simplify Sketch Recognition User Interfaces Development," PhD dissertation, Massachusetts Institute of Technology, 2007.

VITA

Paul Piula Taelé received his Bachelor of Science in computer science and mathematics from the University of Texas at Austin in May 2006. Following his undergraduate studies, he pursued Chinese Mandarin language studies at National Chengchi University (Taipei, Taiwan) in 2006 and 2007. He received his Master of Science in computer science from Texas A&M University in December 2010. His research and academic interests include artificial intelligence, human-computer interaction, and East Asian languages.

Mr. Taelé may be reached at Richardson Building, Room 912A, 3112 Texas A&M University, College Station, TX 7784. His e-mail is ptaelé@gmail.com.