TRANSLATION OF COMPUTER SCIENCE TEACHING MATERIALS INTO THE AMERICAN SIGN LANGUAGE FOR DEAF AND HARD OF HEARING STUDENTS

A Thesis

Presented to

The Faculty of the College of Graduate Studies

Lamar University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Prashant Kumawat

May 2011

TRANSLATION OF COMPUTER SCIENCE TEACHING MATERIALS INTO THE AMERICAN SIGN LANGUAGE FOR DEAF AND HARD OF HEARING STUDENTS

PRASHANT KUMAWAT

Approved:

[illegible]

Stefan Andrei
Supervising Professor

[illegible]

Lawrence J. Osborne
Committee Member

[illegible]

Hikyoo Koh
Committee Member

[illegible]

Lawrence J Osborne
Chair, Department of Computer Science

[illegible]

Brenda S Nichols
Dean, College of Arts and Science

[illegible]

Victor A. Zaloom
Interim Dean, College of Graduate Studies

Abstract

TRANSLATION OF COMPUTER SCIENCE TEACHING MATERIALS INTO THE AMERICAN SIGN LANGUAGE FOR DEAF AND HARD OF HEARING STUDENTS

PRASHANT KUMAWAT

A review of research on deaf students in higher education reveals a significant body of knowledge about the barriers these students face in gaining access to information in the classroom. There are many fewer potential solutions for these problems. Organizations such as 'The Shodor Education Foundation' have developed some signs in the American Signing Language (ASL) for technical languages, but still there are very few signs that are related to computer science terminologies. In addition, there is a dearth of research on the effectiveness of support services as interpreting, note taking, real-time captioning, and tutoring. There are some software tools developed based on animation of a signing avatar as a translator that helps students understanding of study material. However, these tools lack many requirements that are necessary for a student to understand computer science study material. There is no tool present that has a complete dictionary to translate computer science terms and many tools read text from file and directly translate the sentence word by word which is not grammatically correct in ASL.

The prime objective of this research project is to design a computer application that can read computer science teaching materials and translate them into grammatically correct ASL. Creation of computer science related terms is the second objective of this project. The final objective is to create a Graphical User Interface (GUI) that can display English teaching material and a signing avatar which will sign for each sentence in the teaching material simultaneously.

Acknowledgement

Foremost, I would like to express my sincere gratitude to my advisor, Dr. Stefan Andrei, for the continuous support of thesis research, for his patience, motivation, enthusiasm, immense knowledge and financial support. His guidance helped me in all the time of research and writing of this thesis. I would also like to thank my chair, Dr. Lawrence J. Osborne for his guidance and financial support for purchase of various tools required essentially for the completion of the application. I also want to thank my committee member, Dr. Hikyoo Koh, for his effort and continuous help throughout this process.

I would also like to thank the Department of Deaf Studies and Deaf Education for their significant support for completion of this research. I also want to thank my colleagues, Prem Tamang, Pratishara Maharjan, and Amit Mahindrakar, for their continuous helped in the completion of this research work and without their support this research project would have been incomplete.

Lastly, I want to thank my father, Rameshwar Kumawat, my mother, Santosh Kumawat, my brother, Vikram Jeet Kumawat and my sister-in-law, Anita Kumawat for their constant support of my dreams and ideas. I want to thank my family for all of the sacrifices that they have made on my behalf.

TABLE OF CONTENT

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

American Sign Language (ASL), for a time also called Ameslan, is a dominant sign language for deaf Americans, including deaf communities in the United States, in English-speaking parts of Canada, and in some regions of Mexico (Wikipedia 2011). ASL is a natural language and contains phonology, morphology, semantics, syntax, and pragmatics just like a spoken language (Valli and Lucas 2002). The grammatical structure of ASL is completely different from English. Information is encoded not in sound but with the shape and movement of hands and other parts of body and with facial expressions (Baker-Shenk and Cokley 2002). Learning ASL is the same as learning any other foreign language and should not be expected to have the same grammar rules as English. It is a common misconception that ASL is merely the finger spelling of English words. Using the manual alphabet to spell out entire words letter by letter is occasionally incorporated in ASL (Stewart, Stewart and Little 2007). ASL is continuously growing and adding new signs to keep up with new technologies. For example, there is a sign for 'Internet' (the L hands, touching at thumb tips, rotate up from palm down to palm forward) (Fourm 2009). One of the basic problems which the deaf and hard of hearing community faces is not getting enough exposure to science and engineering subjects (Marschark and Hauser 2008). For a deaf/hard of hearing student, to be a science or engineering major, it is required that faculties be fully trained to deliver their lectures in ASL or a translator is needed that can sign the lecture simultaneously with the faculty.

1.2 Motivation

Computers increasingly are prevalent in the classroom. Even with student laptops becoming the norm, some beneficial usage of this widespread technology is being overlooked. Speech recognition applications are maturing for deaf and hard of hearing students and possess the potential to provide real-time notes. This dependence on English captioning however does not ensure equal access, as the average deaf or hard of hearing high school graduates read below the fourth grade level (Lang 2002). Lectures delivered in universities and colleges and study materials are provided primarily in English. Since English is not the first language for deaf and hard of hearing students, it makes it very difficult for students to understand the subject (Sheryl 1997). A lot of terminologies in computer science and English are common but have a specific meaning when they are discussed in the computer science stream. For example, "long" in English would just mean lengthy or is extended in time or dimension, whereas in computer science it is a JAVA primitive type that is used to declare integers that are greater than the maximum value of integer declared by "int" primitive type. Few online and video databases are present. One is the Shodor Education Foundation (T. S. Education Foundation 2005) funded with support from National Science Foundation in 2005 has compiled various STEM (abbreviation for Science, Technology, Engineering and Mathematics) terms but highly specific signs for computer science stream are not present. To make computer science courses available to deaf and hard of hearing students, it is imperative for students to learn all the signs related to computer science terminologies (Robbins 1996). There are around 500 additional signs that are not currently present in either paper or online dictionaries. These factors cumulatively discourage deaf and hard of hearing students to take computer science as their major.

In order to attract deaf and hard of hearing students to computer science subjects there is a need to automate the process of translating lectures to ASL. With the help of the department of deaf studies and deaf education and other signing repositories this project intends to develop an extension of computer science American Sign Language Dictionary (ASLD), an online repository for computer science specific terminologies in ASL. This project also intends to develop a computer application that reads teaching materials, translates the English lectures into ASL, and displays the signs by using a signing avatar animation.

1.3 Problem Domain

With the radical changes in technology, software developers are trying to reduce the need for teachers to learn ASL or a translator for lectures (H|Anim 2000). Software developers around the world have built many applications that use a humanoid which can understand a written and spoken language and translate it to signing language by creating gestures using animation. For instance, “Sign Smith” from Vcom3D (VCom3D 2006) with VCommunicator Gesture Builder (VCom3D 2006), “SiSi(Say it|Sign it)” from IBM (IBM’s Extreme Blue Projects 2007), “SiGML” from eSign (eSign Project 2004) and

"TESSA" from Visicast (ViSiCAST 2000) are popular commercially used tools and all of these tools use a Signing Avatar, a computer modeled representation of a human being, to perform sign language gesture (H|Anim 2000). These tools use a text file or voice as input. However, none of these tools support the Microsoft Power Point slide or Microsoft Word as input files, which are often used by the teachers for teaching purposes. This project represents an effort to overcome some of the difficulties that come across converting teaching material into ASL.

The project proposes a software application which will eliminate the need of having a translator in teaching a computer science course. It will assist teaching of courses to deaf and hard of hearing students by translating the lectures from English to ASL. The project focuses on introducing computer science course related ASL signs, translation of English text contained in teaching materials to ASL text, and presenting an avatar to perform the Sign Language gestures corresponding to the translated text. With the cooperation of the Department of Deaf Studies and Deaf Education, signs for computer science specific terms will be created. This project also attempts to translate English into ASL using correct grammar rules by identifying correct grammatical structure of English sentences by using Stanford Natural Language Parser (The Stanford Natural Language Processing Group 2006). This project presents a much more elaborated algorithm for conversion of English grammar to ASL grammar. It also provides a graphical user interface that can display the teaching material, the ASL text and the signing avatar on one screen.

1.4 A Sample Approach for Conversion

English and ASL have completely different grammatical structures and rules of forming sentences. To convert a sentence from English to ASL different rules may be applied on the sentence depending upon the type of sentence (Stewart, Stewart and Little 2007). To develop an understanding of the relation between English and ASL, here is an example of a conversion of a computer science related sentence into an ASL sentence.

English sentence: What is size of integer array if size of integer is four?

For the above mentioned example the resultant tool must be able to apply following rules to the sentence:

- Eliminate Be-Verbs.
- Decide category of sentence.
- The above example is an information seeking question which has a topic and comment format.
- Wh-adverb is removed and added to the end of the sentence.
- Topic of sentence is placed at the beginning of the new ASL sentence.

American Sign Language sentence: size of integer is four, size of integer array, what?

Once a sentence is converted from English to ASL, a signing avatar can sign the sentence word by word such that the resultant animation delivers a meaningful sentence which is grammatically correct in ASL.

1.5 Scope and Limitation

Many professional tools are available in the market that can convert English to ASL, but there are no tools available for translating computer science related terminology to ASL. This application makes an effort to translate lectures of computer science courses into ASL. The gesture built for computer science related terms are validated by the Deaf Studies and Deaf Education Department. Some of the gestures are taken from the video tutorials provided by Rochester Institute of technology (Rochester Institute of Technology 2011). This software will attract deaf and hard of hearing graduate students for whom English is not the first language, and are interested in taking computer science as their major. The video tutorial also provides signs for science and mathematical fields. This software can be scaled to convert mathematical subjects as well.

Since this software uses a third party tool for the animation and signing, not all the features are available for the programmer. Changing facial expression according to the demand of the sentence is a very important part of ASL which is not provided with this application. Using third party tool also makes the process of translation much slower.

1.6 Structure of Subsequent Chapters

This report contains 6 more chapters. The chapter titles and a summary of each chapter are as follows:

Chapter 2: Methodology

This chapter provides the details about the technique used for the conversion of English to ASL.

Chapter 3: Parsing

This chapter discusses the details about the implementation of the parsing method of English sentences and also focuses on the data structures used to store parsed sentences.

Chapter 4: Conversion

This chapter provides the details about the rules used to convert English grammar to ASL grammar.

Chapter 5: Presentation

This chapter discusses the implementation of the front-end of the application created.

Chapter 6: Results

This chapter discusses the results that are gained after the end of the research.

Chapter 7: Conclusion and Future Work

This chapter provides the conclusion of the research and the potential future work in this field of study.

Chapter 2

Methodology

This chapter is mainly concerned with the analysis and design of the application prior to its implementation. It explains the requirement specification of the application and how the proposed application is going to provide a feasible solution for the problem domain. We will also discuss the categorical distribution of the words and the grammatical relationship between words for sentences in the English language which would provide us an understanding of relationships between the English sentence and the equivalent ASL sentence. This chapter also discusses the data structure used to represent an English sentence. Grammatical rules of formation of sentences in ASL are discussed to deliver a clear understanding of the relationship between ASL and English. We will then discuss various algorithms that will be used in the implementation of this project to convert English to ASL.

2.1 Requirement Specification

This project is intended to design an application that is able to read computer science teaching materials and convert them to American Signing Language which is comprehensible to deaf and hard of hearing students, and should be able to communicate to students visually by signing the text. The application should be able to read Microsoft© PowerPoint (used widely across the globe for delivering presentations) lecture notes into

English sentences. The English sentences must be then converted to ASL by applying the proper ASL grammar rules. The application must also have a graphic engine for a signing avatar and an American Sign Language Dictionary (ASLD) for the graphic engine. The graphic engine of application must be able to read the ASL text and should make use of the ASL dictionary to create signing gestures for English sentences in lecture notes. The application must be able to display a signing avatar with the PowerPoint presentation to make the text and the visual language available at the same time. New gestures related to computer science terminologies must be created which deliver a clear meaning of the words, as intended, to the deaf and hard of hearing students.

2.2 English Sentence Structure

2.2.1 Part-of-Speech Tagging and Type Dependencies

To translate an English sentence into an equivalent sentence with a different grammatical structure, we should first understand how a sentence in the English language can be decomposed into words that can fit into different categories, and then the words can be picked up to form a sentence with a different syntactical structure. The Part-of-Speech tagging (POS tagging or POST), also called grammatical or word category disambiguation, is the process of marking up the words in a text as corresponding to a particular part of speech, based on both its definition, as well as its context, i.e., relationship with adjacent and related words in a phrase, sentence, or paragraph . English words have been traditionally classified into eight lexical categories or parts of speech (and are still done so in most dictionaries) (The Stanford Natural Language Processing Group 2006):

- Noun: any abstract or concrete entity.
- Pronoun: any substitute of a noun or noun phrase.
- Adjective: any qualifier of a noun.
- Verb: any action or state of being.
- Adverb: any qualifier of an adjective, verb or other adverb.
- Preposition: any establisher of relation and syntactic context.
- Conjunction: any syntactic connector.
- Interjection: any emotional greeting.

For example, the sentence “Today is a wonderful day” can be represented as having the following parts of speech.

Today/NNP is/VBZ a/DT wonderful/JJ day/NN ./.

NNP: Proper noun, singular

VBZ: Verb, 3rd person singular present

DT: Determiner

JJ: Adjective

NN: Noun, singular or mass

. : Sentence-final punctuation

Apart from the categorical distribution, we also need to know the relationship between the words. The type dependency represents the binary grammatical relationship between two words in sentences. This application uses the Stanford type dependencies which contain 55 grammatical relationships (Mameffe and Manning 2010). For the above example, "Today is a wonderful day" the type dependencies are shown below.

nsubj(day-5, Today-1)

cop(day-5, is-2)

det(day-5, a-3)

amod(day-5, wonderful-4)

nsubj:	Nominal Subject
cop:	Copula
det:	Determiner
amod:	Adjectival Modifier

2.2.2 Treebank

Once the categories of words in the English language are decided, the next task would be to represent a sentence with its POS tags in a data structure. A Treebank, or parsed corpus, is a text corpus in which each sentence can be parsed, i.e., annotated with syntactic structure. The syntactic structure is commonly represented as a tree structure, hence, the name Treebank. Some treebanks follow a specific linguistic theory in syntactic annotation but more try to be less theory-specific (The Stanford Natural Language Processing Group 2006). However, two main groups can be distinguished: the treebanks that annotate phrase structure and those that annotate dependency structure (Pettibone 2002). It is important to clarify the difference between the formal representation and the data structure used. Treebank are necessarily constructed according to a particular grammar. The same grammar may be implemented by using different data structure (The Stanford Natural Language Processing Group 2006). For example, the syntactic analysis of "Today is a wonderful day" is shown in Figure 2.2, but can be represented by simple labeled brackets in a text file as shown in Figure 2.1.

```
(ROOT
  (S
    (NP      (NNP      Today))
    (VP      (VBZ      is)
           (NP      (DT      a)
                    (JJ      wonderful)
                    (NN      day)))
    (.     .)))
```

Figure 2.1 The Bracketed Tree Representation of Sentence with POS Tags.

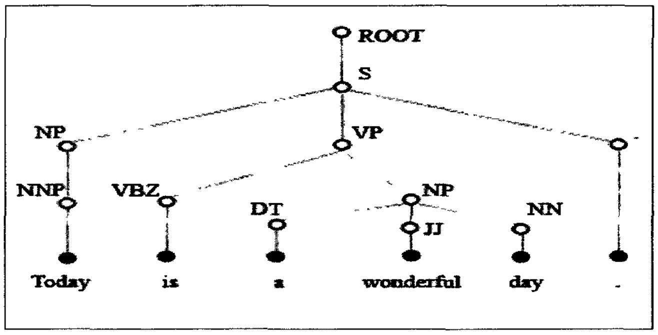


Figure 2.2 The Hierarchical Tree Representation of Sentence with POS Tags.

Listed below are the standard tags used in Penn Treebank. Each tag has its description associated with it (Marcus, Marcinkiewicz and Santorini 1993).

Table 2.1 Word Level POS Tags

Tag	**Description**
CC	Coordinating conjunction
CD	Cardinal Number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer

POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	To
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, Past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

2.3 ASL Grammar

Grammar is a set of rules for describing a language. These rules guide users in correct speaking or signing of a language. These rules are defined by the group of people who use the language. Just like any other spoken and written language, ASL grammar also have its own rules for phonology, morphology, syntax, and pragmatics (Valli and Lucas 2002). To make the signing by the avatar realistic to the deaf and hard of hearing students, the English sentences must be modified such that the resulting sentence should follow ASL grammar rules. There are fourteen grammar rules for ASL as listed below which are implemented by this application (Stewart, Stewart and Little 2007).

a. Topic/Comment: In a topic/comment sentence, the topic is described first followed by the comment.

   E.g.: "He won $3,000,000 and he is happy" will be translated to "He won 3 million dollars, he happy". In this example "He won 3 million dollars" is the topic and "he happy" is the comment.

   The basic idea which governs this rule is there must be a topic before there can be a comment about the topic.

b. Tense with time adverbs: The time adverb is placed at the beginning or near the beginning of a sentence.

   A. ASL: Last night, sunset beautiful. (English: The sunset was beautiful last night.)

   B. ASL: In-2-days, you go work. (English: You go to work in two days.)

   C. ASL: Me yesterday, stay home. (English: I stayed home yesterday.)

Placing a time adverb at the beginning of a sentence marks the tense of the sentence. Using the time adverb is the most common means of indicating tense. Unlike English, verb signs never undergo changes to indicate tense. Because there are no changes to a verb sign, the time that an action occurred must come before the verb sign.

c. Simple yes/no questions: In short sentence that ask a yes/no question, the order of sign is variable.

   A. ASL: You exercise want?
   B. ASL: You want exercise?
   C. ASL: Want exercise you?
   D. ASL: Exercise you want?

   In short, questions such as those in sentences a-d, the signer is asking simple yes/no question, and the correct English translation for all of them is "Do you want to exercise?"

d. Long yes/no questions: Long yes/no questions use the topic/question format. In a longer yes/no question, the first describes the topic and then places the sign that is asking the question at or near the end of sentence.

   E.g.: "Is that black cat climbing the tree yours?" can be translated as "Cat black tree climb, your?" in ASL.

   From the above example, we infer that first the topic is signed and then the question is asked.

e. Information seeking questions: Simple questions that ask for information have variable sentence structure and rely on non-manual signals to distinguish them from declarative sentences.

E.g.: “How long has she worked here?” will be translated to “She works here, how long?” and “Why did the city destroy the building?” will be translated to “City destroy building, why?”

In these sentences, the “WH” question sign or phrase comes at the end of question. It follows a topic/question format because a topic is described followed by a question about it.

f. Pronominalization: Pronouns are indicated by pointing to either (a) a person or thing that is present or (b) a place in the signing space that is used as a reference point for a person or a thing. Pointing is mostly done with the index finger. If the person or object is present, then the signer merely points to them and the pointing becomes the pronoun. In the absence of a person or object the singer use the signing space to insert reference point that will represent a specific person or object.

g. Rhetorical questions: In a rhetorical question, the signer asks a question and then answers it. There is no expectation that someone else will answer the question. Rhetorical questions often make use of sign for “WH” questions such as WHY and HOW. When ‘why’ is used, the proper translation will often include the conjunction “because.”

h. Ordering of simple sentence: In simple sentence the verb can be placed before or after the object of the sentence.

E.g.: "Me play game." Or Me game play" are both translation of the English sentence "I play a game."

i. Conditional sentence: In a conditional sentence, first the condition is described then the outcome of this condition is described. The condition can be clearly marked by use of sign SUPPOSE.

   E.g.: "I will have to leave if she sees me" will be translated to "Suppose she see me, me have to leave."

   The fingerspelling of I-F is also used to construct a conditional clause. I-F can be used interchangeably with the sign SUPPOSE. I-F is often used to give greater emphasis to a condition.

j. Negation: You can negate a thought by placing negative sign before the verb or by first describing a topic and then signing the appropriate negative sign or giving a negative head shake.

   E.g.: "I'm not watching the football game." can be translated to the ASL as "Me not watch football game." In this sentence, the thought is negated by placing a negative sign before the verb. The sign NOT negates the sign WATCH.

k. Be-verb Elimination: Be-verbs must be eliminated from the sentence.

   am, is, are, was, were, been, being, be are some examples of Be-verbs.

   E.g.: "ASL is a good language" will be signed as "language good, ASL"

l. Removing Articles: Articles such as 'a', 'an' and 'the' are removed from the sentence. In the above example, 'a' is removed from the sentence.

m. Shifting Adjectives: In ASL adjectives are placed after their corresponding noun. In the above example, 'good' is placed after 'language'.

n. Object-Subject-Verb: For lengthy sentences, in ASL, the verb must be placed after the subject and the object.

2.4 Algorithm

This operation is used to build the Semantic Parse Tree from the POS Tags. Each tag forms the nodes of the Semantic Parse Tree. This operation is useful to add new nodes and representing new words to make the context clear in ASL.

2.4.1 Process Flow

The conversion of English to ASL and the creation of the Signing Avatar are complicated procedures which involve several tools and algorithms. This process is divided into the following steps:

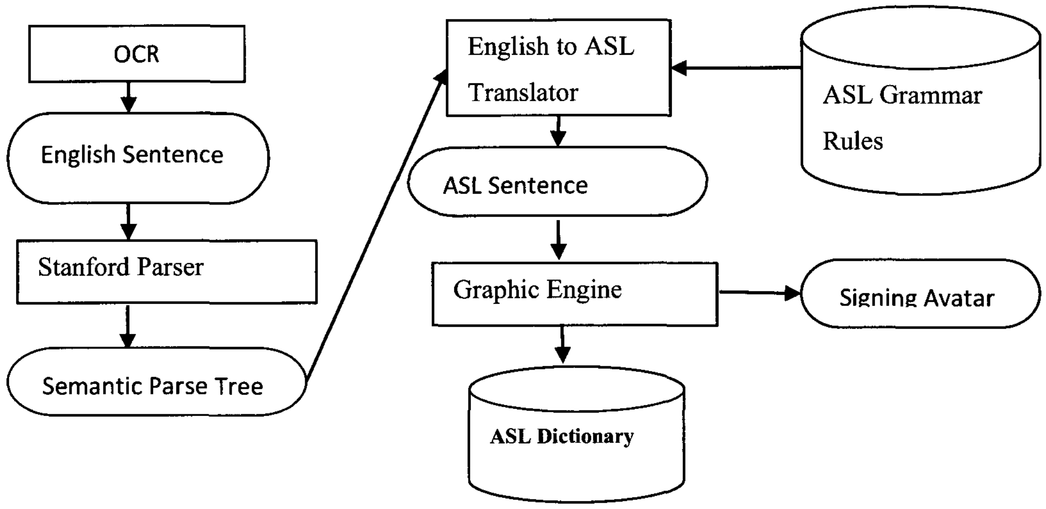


Figure 2.3 The Design View of the Software Project

2.4.2 Algorithm: ASL Translation

The input: An arbitrary English sentence

The output: The Equivalent Translated ASL Sentence

```
Procedure ASLTranslation (input: English_Sentence)

Begin:

    i)    Parse the English sentence using the Stanford Parser which gives the POS
          Tagset, Syntactic Tagset and Type Dependency as output.
    ii)   Build the Type Dependency List (TDL) from the given sets of the Type
          Dependencies.
    iii)  Generate the Semantic Parse Tree from the given set of POS and the
          Syntactic Tagset using Addition Operation.
    iv)   Sort the grammatical rules of ASL based on their priorities stored in the
          List. Each grammatical rule has its priority set based on its importance.
    v)    For each rules R  in the list Grammatical Rule List (GRL),
              a) Check if the rule applies to the sentence.
              b) Fetch the type dependency (TD), associated with the R.
              c) Based on the Rule R,
                     Either      Perform Rotation ()
                     Or          Perform Addition ()
                     Or          Perform Deletion ()
              d) Add the Non-manual Markers to the nodes in the Tree.
```

vi) Perform the Preorder Traversal of the final modified SPT. The ASL text is generated by concatenating all the strings at the leaf nodes of the Semantic Parse Tree.

End

2.4.3 Algorithm: Preorder Traversal

The following recursive algorithm is used to traverse the Semantic Parse Tree in preorder manner, i.e., visiting each root node first and then its child nodes from left to right.

```
Procedure preorder (input: SPT)
Begin:
    If SPT == null then return;
        visit (SPT);                        – visit/process the root
        For (each child with index i of the node SPT)
        preorder (SPT --> child[i]);        – traverse the child in the given List
                                            – from left to right
        Endfor
    End
```

2.4.4 Algorithm: Signing Avatar Generation

The following algorithm is the main algorithm of this project that generates the Signing Avatar animation videos from the English Sentences contained in Power Point slides.

The input: PowerPoint Slide containing English Sentence

The output: Signing Avatar animation videos

```
Procedure ASLTranslation (input: PowerPointSlide)
Begin:
        SlideList ← Get all the slides from PowerPoint using
                Aspose.slides.getSlides()
        For each slide with index i in SlideList
                SentenceList ← Get all the sentences in slide the
                        SlideList[i] using Aspose.slides.getText();
                For each sentence with index j in SentenceList
                        ASLText = ASLTranslation (SentenceList[j]);
                        Invoke AutoIt using JavaRunCommand;
                        Generate Signing Avatar from ASLText;
                        Export the SigningAvatar animation video to the folder;
                EndFor
        Endfor
End
```

Chapter 3

Parsing

This chapter basically deals with the detailed implementation method of getting text from the input source of the application and then parsing it into words that will fit a particular category. This chapter also discusses the data structure used to store the text and also the list of grammatical relationships between different words in a sentence. Different representation techniques are discussed for the hierarchal representation of the word tree. There are different types of tools and libraries that are used for completion of this application. This chapter only covers tools that are used for parsing of sentences. Other tools and libraries which are used are covered in subsequent chapters.

3.1 System Requirement

This section covers the tools and libraries required by the application used to parse an English sentence into a tree structure.

3.1.1 Tools and Libraries Used

The following tools and libraries are used in this application:

a. Aspose.Slides and Aspose.Words

The commercial tools Aspose.Slides© (Aspose 2011) and Aspose.Words© (Aspose 2011) from the Aspose company are used to interact with the Microsoft© Power Point slides and Microsoft© Word documents. Aspose.Slides provide the interface in the Java to manage texts, shapes, tables, animations, adding audio and video to slides, previewing slides, exporting slides to PDF format, etc. Similarly, Aspose.Words is a class library in Java that performs a great range of document processing tasks and supports DOC, RTF, HTML, Open Document, PDF, and other formats. This project uses these libraries to extract the English text from given lectures in Microsoft© Word and Power Point and displays it on the GUI of the software.

b. The Stanford Natural Language Parser

Stanford English Parse is a natural language parser that works out the grammatical structure of English sentences, for instance, which groups of words go together (as "phrases") and which words are the subject or the object of a verb. The parser is a Java implementation of a probabilistic PCFG and dependency parser for English, German, Chinese, and Arabic. The Stanford dependencies provide a representation of grammatical relations between words in a sentence. They have been designed to be easily understood and effectively used by people who want to extract textual relations. It uses knowledge of language gained from hand-parsed sentences to try to produce the most likely analysis of new sentences (The Stanford Natural Language Processing Group 2006).

3.2 Input

As per the requirement of the application, it must be able to read English text from teaching materials. The most commonly used application to deliver lectures is Microsoft PowerPoint. The Aspose.Slides library is used to retrieve the text from PowerPoint slides. Apart from retrieval of text application, it must also be able to gather the properties of text. The property of text must be preserved at the time of output generation as the output must resemble the input PowerPoint file. The Aspose Slides provide a class interface which can be used by other Java classes to get the text and the properties of the text. The Aspose Slides provide com.Aspose.Slides (Aspose 2011) class package with methods listed below in the table.

com.Aspose.Slides: This class is used to read the content from the Microsoft PowerPoint slides. This class provides the following package.

Table 3.1 Method Description of Java Class com.Aspose.Slides

getSlides ()	Returns the list of all slides in a PowerPoint presentation.
getSlideById(long id)	Returns the slide by Id.
getSlideByPosition(int position)	Returns the slide by Slide Position.
getSlideComments()	Returns the collection of slide comments.
getShapes()	Returns the shapes of a slide.
getTextFrame()	Returns the TextFrame object for a Shape.

getParagraphs()	Returns the list of all paragraphs in a frame.
getText()	Returns the plain text of a portion.
getFontColor()	Returns the color of a portion.
getFontHeight()	Returns the font height of a portion.
getFontIndex()	Returns the index of the used font in a Fonts collection.
isFontBold()	Determines whether the font is bold.

3.3 Parsing of English Sentence

This section explains the processing of an English sentence before it is converted to ASL using the grammar rules. It constitutes of the following subsections:

- Parsing and Categorical Distribution
- Semantic Parse Tree
- Type Dependency List

3.3.1 Parsing and Categorical Distribution

Once the sentence is retrieved from the PowerPoint slides, the next task is to separate all the words from the sentence and classify them into lexical categories. For this task we need a Natural Language Parser (NLP). This application uses the Stanford Natural Language Parser (The Stanford Natural Language Processing Group 2006) to parse words from a given sentences and classify them to appropriate POS tagset and syntactic tagset to represent the grammatical structure of the given sentence. The Stanford parser uses the Penn Treebank POS tagset for the representation of part of speech information and symbols of the English words. The Penn Treebank has 36 word level POS tags and 12 punctuation and currency level tags (Pettibone 2002). All the word level and punctuation tags are directly associated with each word. There are 21 phrase levels and 5 clause level tags that are parent tags for a phrase and clause, respectively (The Stanford Natural Language Processing Group 2006). A pictorial view of tagset hierarchy is explained in more detail in Semantic Parse Tree subsection. The Stanford parser also lists the Type Dependencies for a given sentence. The current representation of the Type Dependencies contains 52 grammatical relationships (Mameffe and Manning 2010). Let us consider the following computer science related English sentence to demonstrate the output of Stanford Parser:

"Java is an object-oriented programming language."

The word level POS tags are represented as an output by:

```
Java/NNP is/VBZ an/DT object-oriented/JJ programming/NN language/NN ./.
```

The POS tagset tree representation:

```
(ROOT
  (S
    (NP (NNP Java))
(VP (VBZ is)
  (NP (DT an) (JJ object-oriented) (NN programming) (NN language)))
(. .)))
```

The Type Dependencies are given by:

```
nsubj(language-6, Java-1)
cop(language-6, is-2)
det(language-6, an-3)
amod(language-6, object-oriented-4)
nn(language-6, programming-5)
```

The Stanford Parser provides edu.stanford.nlp package, and this project uses various classes in this package to generate the output of the sentences as shown above. The classes used by this application are as follows:

The edu.stanford.nlp.parser.lesparser.LexicalizedParser Class:

Table 3.2 Method Description of Class nlp.parser.lesparser.LexicalizedParser

apply(Object in)	Converts a Sentence/List/String into a Tree. If it cannot be parsed, it is made into a trivial tree in which each word is attached to a dummy tag ("X") and then to a start nonterminal (also "X").

The edu.stanford.nlp.trees.PennTreebankLanguagePack Class:

Table 3.3 Method Description of Class nlp.trees.PennTreebankLanguagePack

grammaticalStructureFactory()	This method returns a
	GrammaticalStructure suitable for this language/Treebank.

The edu.stanford.edu.tree.TreePrint Class:

Table 3.4 Method Description of Class edu.stanford.edu.tree.TreePrint

printTree(Tree t, PrintWriter pw)	This method prints the sentence along with POS Tagset, Syntactic Tagset and Tye Dependencies.

Parsing of English sentences is performed by EnglishParser class. The AsposePowerPointReader class, after getting the text, invokes the object of EnglishParser class to parse English sentences and generate Semantic Parse Tree and the Type-Dependency List for a particular sentence. The EnglishParser class invokes the object of

ASLGrammar class to generate an ASL sentence with correct grammatical syntax. The EnglishParser class provides the following interface.

Table 3.5 Method Description of Class EnglishParser

parse(String str)	This method takes English sentence as input and returns ASL sentence as output.
generateSemanticParseTree(Scanner s)	This method read the sentence from an input stream and creates a SemanticParse Tree.
buildTypeDependencyList(Scanner S)	This method read the sentence from an input stream and creates a Type-Dependency List.

3.3.2 Semantic Parse Tree

Once the sentence is parsed by the Stanford Parser it provides us with POS tags for each word and group of words. After the parsing, a data structure is required to store the words and the POS tags corresponding to each word. The Semantic Parse Tree is an Abstract Data Type (ADT) for representing POS tagset and Syntactic Tagset in the form of the Generic Tree. The EnglishParser class generates a semantic parse tree by parsing the output produced by the Stanford parser after parsing of the English sentence. The

Semantic tree is used by ASLGrammar Class to generate ASL from the tree structure by moving the tree branches and the nodes according to the ASL grammar rules. Each node of the semantic parse tree consists of a vector that can store the child nodes for a particular node. Each node consists of an index value. The index value is not unique but is used to decide if the node is a POS tag or a word from the given sentence. The index value of POS tags is -1 and for the leaf nodes the value is equal to the position number of the word in the sentence. The leaf node Vector is of size one because it contains a single word. The SemanticParseTree class generates ASL text after all the operations by ASLGrammar class have been performed on the tree structure. After applying the grammar rules, the tree structure is traversed and the leaf nodes are added to generate ASL string. The following methods provide interface to manipulate the tree nodes:

Table 3.6 Method Description of Class SemanticParseTree

searchNode_Forward(String nodeName, Node curNode, int leafIndex)	This helper method searches the given node recursively starting from the leftmost child on the basis of node name in the Tree and returns the pointer to the node if it succeeds.
searchNode_Reverse(String nodeName, Node curNode, int leafIndex)	This helper method searches the given node recursively starting from the rightmost child on the basis of node name in the Tree and returns the pointer to the node if it succeeds.
insertInTree(Node parentNode,	This private method is a helper method that connects

Node newNode)	the newly created node to its parent in the Semantic Parser Tree.
moveNodesInTree(String source, int src_index, String dest, int des_index)	This method helps to exchange the two subtrees in the given tree, i.e., order of the tree. Used to change the order of the given Sentence S+V+O to O+S+V.
rearrangeChildNodes(Vector list)	This method helps to rearrange the nodes in the given tree. It is used to move the adjectives before the noun
shiftNodesInTree(Vector list)	This method helps to shift the nodes in the given SPT, used to put the adverbs before the main verb.
removeNodes(String nodeName)	This method helps to prune the subtree or node if the name matches. Used for removing the Articles/Determiners, Auxiliary Verbs.
insertNonMannualMarker(String nodeName, NonMannualMarkers NMM)	This method adds the Non-manual Markers to the given Nodes.
generateASL(String _rule, int _options)	This method is used to apply particular rotation to Semantic Parse Tree as provided by the invoking object and traverse tree to generate ASL sentence.
generateStringTillNode(Node searchNode, Node curr)	This method is used to generate string till a particular node starting from the first leaf node.
generateStringFromNode(Node searchNode, Node curr)	This method is used to generate string starting from a given node till the last leaf node of semantic parse tree

3.3.3 Type Dependency List

The TypeDependency Class maintains a list in the form of a vector of objects of the class Dependency. The Type-Dependency List is created by EnglishParser class after the generation of Semantic Parse Tree. The Stanford Parser also produces the grammatical relationship between words in the form of sentences as an output of parsing of English sentences. Once output is produced, it is parsed to separate the words and the relationship between them. The object of class dependency contains the information regarding the grammatical relationship of words. Both words in a relation also contain an index value generated by Stanford Parser as a position in the type dependency tree structure. The Type-Dependency List is used by the ASLGrammar Class to determine the relationship between words in order to apply particular ASL grammar rule. The class TypeDependency consists of the following methods:

Table 3.7 Method Description of Class TypeDependency

addDependency(String relationshipType,String Iword,String IIword,int Iindex,int IIindex)	This function adds dependency to the Vector used to maintain list. Each dependency is added at the end of the list.
getDependency(String type)	This function returns a list in form of a Vector if any relationship in the list matches the type provided as parameter to the method

Chapter 4

Conversion

This chapter discusses the strategy used to convert an English sentence into an ASL sentence by applying proper grammatical rules of ASL. The chapter also provides a glance over the non-manual markers and their application. Finally the creation of new gestures for this application is discussed.

4.1 Required Tools

In this chapter, we discuss gesture creation for the new computer science terminologies. This application uses a commercial gesture building tool, VCommunicator Gesture Builder, provided by VCom3D (VCom3D 2006).

The Gesture Builder will allow users to create new gestures (or signs), including gestures that can be spatially inflected at run-time. A key feature of this tool is the use of Inverse Kinematics (IK) technology. This allows the user to focus on the hand position. Once the user selects a hand shape and positions the hand, the IK software automatically places the joints of the wrist, elbow, and shoulder in the correct position. This approach is fast and easy, and puts the power of creativity completely into the hands of the users (VCom3D 2006). This tool also supports the exporting the newly created gestures in action format which can be added to the Signing Dictionary being used by the SignSmith Studio (tool used for signing avatar animation which is discusses in next chapter).

4.2 ASL Grammar

The ASLGrammar class plays a vital role of converting an English sentence into a syntactically correct ASL sentence. The ASLGrammar class contains a reference to the Semantic Parse Tree of the English sentence and also to the Type-Dependency List as discussed in the previous chapter. It uses both data structures together to decide the category of a sentence and to find the appropriate set of rules that will be applied to the particular sentence. The ASLGrammar class stores various grammatical rules of ASL and provides methods to apply these rules.

Table 4.1 Method Description of ASLGrammar Class

ASLGrammar(SemanticParseTree _SPT, TypeDependency _TD)	This function is the constructor of ASLGramamr class and generates the vector of rules.
applyGrammarRules()	This function decides the category of the sentence and invokes appropriate methods to apply different rules according to the category of the sentence.
yesNoQuestionRule()	This method determines whether the sentence is a yes/no question or not. This function also determines if the question is long question or short.
applyYesNoRule(int yesNoValue)	This method rotates SPT according to the

	yesNoValue generated by yesNoQuestionRule() method.
applyTopicCommentRule()	This method rotates tree to put topic before comment as discusses further in ASL rule implementation section.
isInfoQuestion()	This method determines whether the sentence is an information seeking question.
isNegation()	This method determines if the sentence has a negation relation in it.

4.3 Implementation of ASL Rules

a. Topic/Comment Rule

A sentence with topic and comment is generally combined with the use of a conjunction. Once a conjunction is identified in a sentence, it is considered as having a topic and a comment. The applyTopicCommentRule() method in ASLGrammar class identifies the conjunction in the sentence by checking the POS tag of nodes in the semantic parse tree. A parent node of conjunction in a semantic parse tree must contain "IN" as the POS tag value. The generateASL() method in then called to rotate the semantic parse tree around the conjunction. After the rotation of tree, the topic precedes comment.

b. Short yes/no Question Rule

The isYesNoQuestion() method in ASLGrammar class is used to find weather the sentence belongs in a yes/no question category. isYesNoQuestion() method consists of four word banks according to their POS tags (MD, VBP, VBZ and VBD). If words such as ‘can’, ‘could’, ‘is’, ‘are’, ‘do’ etc. are the first words of a sentence and ends with a question mark then it is a yes/no question. Upon identification, this method returns the POS tag of the starting word which is then removed from the semantic parse tree by applyYesNoRule() method. If the length of the sentence is shorter than sixwords, it falls under short yes/no question category.

c. Long yes/no Question Rule

After applying the isYesNoQuestion() method and removing the starting word from the sentence, the length of the sentence is checked, and if it is greater than a certain value, it is marked as a long sentence. The average length for a short sentence is considered to be six. If the length of sentence is greater than six, the topic/comment rule is applied to the sentence as discussed above.

d. Information Seeking Question

All information seeking questions start with wh-adverb. The isInfoQuestion() method works similar to that of isYesNoQuestion() method. It consists of three word banks according to their POS tags (WRB, WP and WDT). isInfoQuestion() returns the POS tag of the word. The word with identified POS tag is then removed and if the length of the sentence is greater than six, the topic/comment rule is applied, otherwise it remains the same. After applying topic comment rule the removed word is then added at the end of the ASL sentence.

e. Negation Rule

This rule makes use of the Type-Dependency List to determine whether a negation relation exists in a sentence. If there is a negative relationship present, in the Type-Dependency List of the sentence then this method invokes the negateTree() method on the Semantic Parse Tree and results in moving the negation word at the end of the tree. The negation is determined by "neg" in the Type-Dependency List.

f. Pronominalization

The pronominalization rule requires the signer to indicate pronouns by pointing to either (a) a person or thing that it represents (b) or a place in the signing space that is used as reference point. The SignSmith studio has inbuilt gestures for pronouns and when a pronoun is encountered, the SignSmith studio uses the inbuilt gestures for pointing. The SignSmith studio does not allow users to add custom pointing gestures.

g. Tense with Time Adverb:

The class ASLGrammar consists of a time adverb bank which contains time adverbs (Kelly and Kelly 2010). The applyTimeAdverb() method scans the sentences for time adverbs. If a time adverb is found in the sentence it is moved and placed at the beginning of the sentence followed by a comma. For example, in the sentence "I stayed home yesterday", the word yesterday is moved inside the tree such that it becomes the first node and a comma is also added after it.

h. Article Elimination

Articles are identified by the POS tag "DT". For all sentences, the ASLGrammar class applies removeNodes() method with "DT" as the parameter. It results in removal of all the nodes with "DT" POS tag as their parent node.

i. Shifting Adjectives

The Type-dependency list for all the sentences is checked. If "amod" relation is present in the type dependency list then the adjective is moved in front of the corresponding word in the Type-dependency list. It is done by using rearrangeChildNodes() method in SemanticParseTree class.

j. Conditional Sentences:

If the sentence is a declarative sentence then it is checked for having "if", and "then" words in it. Once index of "if" and "then" are acquired, then the "then" part is removed and is put at the beginning of the sentence. "Suppose" is added as the leftmost child in the tree and a comma is inserted at the end of "then" part. "if" and "then" are removed from the tree.

4.4 Non-Manual Markers

The basic elements in a signing language are Handshape (or Handform), Orientation (or Palm Orientation), Location (or Place of Articulation), Movement, and Non-manual markers (or Facial Expression), summarised in the acronym HOLME. Non-manual markers consist of the various facial expressions, head tilting, shoulder raising, mouthing, and similar signals that we add to our hand signs to create meaning. Non-manual markers are an essential part of communication using ASL. The signs consisting of emotional expressions are usually signed with the help of non-manual markers (Lifeprint 2010). Therefore, the sentence "I drove here and it was pleasant" has a greater length in English as compared to "I drove here," but in ASL both have the same length since the emotional expression is signed simultaneously with signing of "I drove here." The class NonManualMarker consists of lists of Non-Manual markers. The markers are added to the nodes of Semantic Parse Tree as per the requirement of a sentence, which is decided by the grammatical rules implemented by the ASLGrammar class.

4.5 Gesture Creation

For the completion of the tool, all signs related to the computer science field must be present in the ASL dictionary. This application uses the Sign Smith ASL dictionary to map gestures according to the words for the signs related to computer science that are not necessarily present in the Sign Smith ASL dictionary. It is required to have a tool that can create gestures for computer science terms and then merge those gestures with the ASL

Dictionary so that at the time of signing the words can be mapped from the same dictionary.

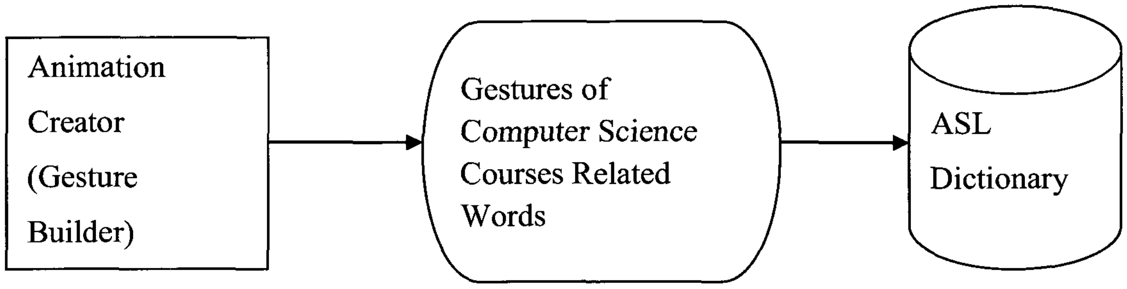

Animation Creator (Gesture Builder)
Gestures of Computer Science Courses Related Words
ASL Dictionary

Figure 4.1: Addition of New Gestures to ASL Dictionary

4.5.1 Sources for Signing Gesture

For creation of signing gestures for computer science terminologies, it is required to have authentic signing sources so that the sign created for new terms are not ambiguous to the deaf and hard of hearing students. Students in the Deaf Education and Deaf Studies Department performed a group discussion to agree upon newly created computer science terminology signs and the signs were recorded. This project also makes use of video tutorials provided by the Rochester Institute of Technology (National Technical Institute for the Deaf) to generate computer science specific signs (Rochester Institute of Technology 2011). Online signing repository such as Shodor Education Foundation (T. S. Foundation 2005) and Signing Savvy (Signing Savvy 2003) were also used to build a few signing gestures.

4.5.2 Gesture Creation Technique

This software uses the SignSmith Studio to generate the animation for the translated text. The gestures created must be compatible to the SignSmith Studio therefore, “VCommunicator Gesture Builder”, a tool by VCom3D, is used to create gestures for computer science related words (VCom3D 2006).

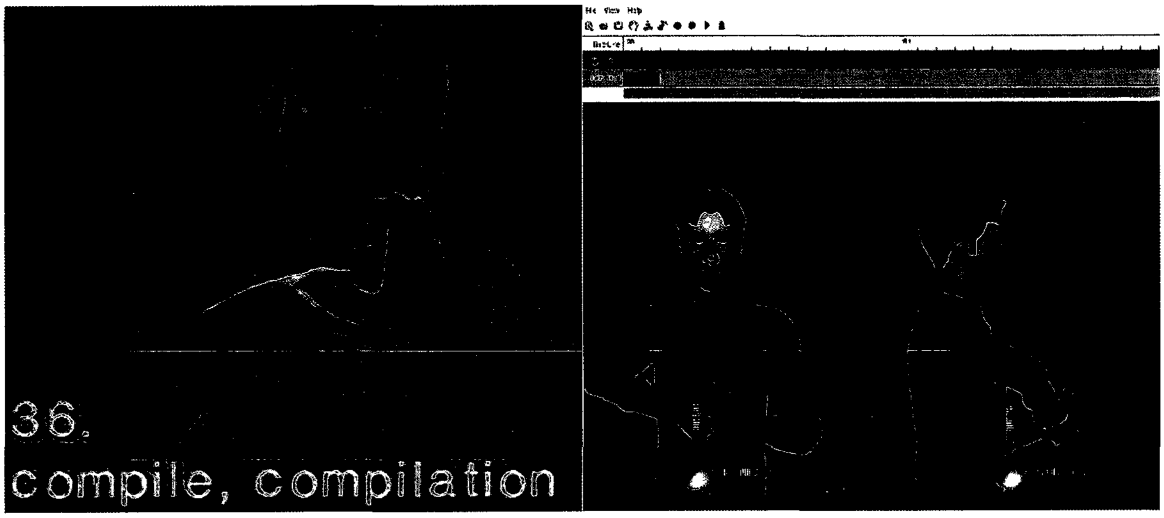


Figure 4.2: Conversion of Signs Using Gesture Builder

Chapter 5

Presentation

This chapter basically focuses on creation of signing gesture for computer science terminologies and generation of the signing avatar. This chapter also provides details of graphical user implementation and automation of the process of signing animation generation.

5.1 Related Tools and Libraries

5.1.1 Java Media Framework(JMF)

The Java Media Framework (JMF) (Oracle 2010) is a Java based library that provides simple, unified architecture to synchronize and control audio, video, and other time-based data within Java applications and applets. Further, this package can capture playback, stream, and transcode multiple media formats. This project uses this library to display the animation videos on the GUI of the software.

5.1.2 AutoIt v3

The AutoIt v3 (AotuIt 2010) is a scripting language that is designed for automating the Windows GUI and general scripting. It uses a combination of simulated keystrokes, mouse movement, and window/control manipulation in order to automate tasks in a way not possible or reliable with other languages. It is a powerful language that supports complex expressions, user functions, loops, etc.

5.1.3 VCom3D SignSmith Studio

SignSmith Studio (VCom3D 2006) is an authoring tool for creating multimedia that incorporates sign language gestures. This tool uses the Signing Dictionary (containing around 2000 gestures) to insert the gesture corresponding to a given English word. The transition from one gesture to another (i.e., transition from one word to another) is very smooth and makes this tool very effective for creating ASL signing animation. In addition, the user can export the animation as video files that can be played back without the need of the software.

5.2 Automated Signing Movie Creation

This application provides a user interface which displays the PowerPoint lecture slides and the signing movie for each sentence in the slide. The process of English to ASL conversion goes hand in hand with the signing movie creation by Sign Smith. With the use of AutoIt, a script is created which is compiled to generate a windows executable file, and the JavaRunCommand Class invokes the executable at the end of the conversion of a sentence. The JavaRunCommand provides the following method:

Table 5.1: Method Description of JavaRunCommand Class

runSignSmith()	This method executes the SignSmith program from the specified location.

5.3 Output Generation

This section describes the process of the creation of the signing avatar performing the gesture corresponding to the translated ASL and tools used to carry out this process.

5.3.1 AutoIt v3 Script

This script is used to automate tasks on a windows system. AutoIt Editor is used to create a desired autoIt script. Once a script is created it is compiled to generate an executable file. The JavaRunCommand class is used to run the compiled executable file. This executable performs the following tasks:

- Executes the SignSmith.exe from C:\Program Files (x86)\Vcom3D\Sign Smith Studio 3.0.
- Opens the import file option, and uses the input file generated by the application as an input file.
- Opens the export options and saves the animation as movie with ".avi" extension.

5.3.2 SignSmith Studio

SignSmith Studio works as the graphic engine for this application and is invoked by the autoIt script. SignSmith uses a text file as input and generates signing gestures according to the text in the file. It can also create a movie for the generated animation.

5.3.3 Animation and PowerPoint Slides Display

This application uses the Java Media Framework and the commercial tool, Aspose Slides, together to integrate the PowerPoint presentation and the animated movie into a Java based GUI. The content of slides are displayed in JTextPane frame. The class and the methods involved in the process of creating this GUI are listed below.

The MediaPlayer Class

This class uses the JMF Application Programming Interface (API) and provides the interface to interact with the video from our Java Based Tool.

Table 5.2: Method Description of MediaPlayer Class

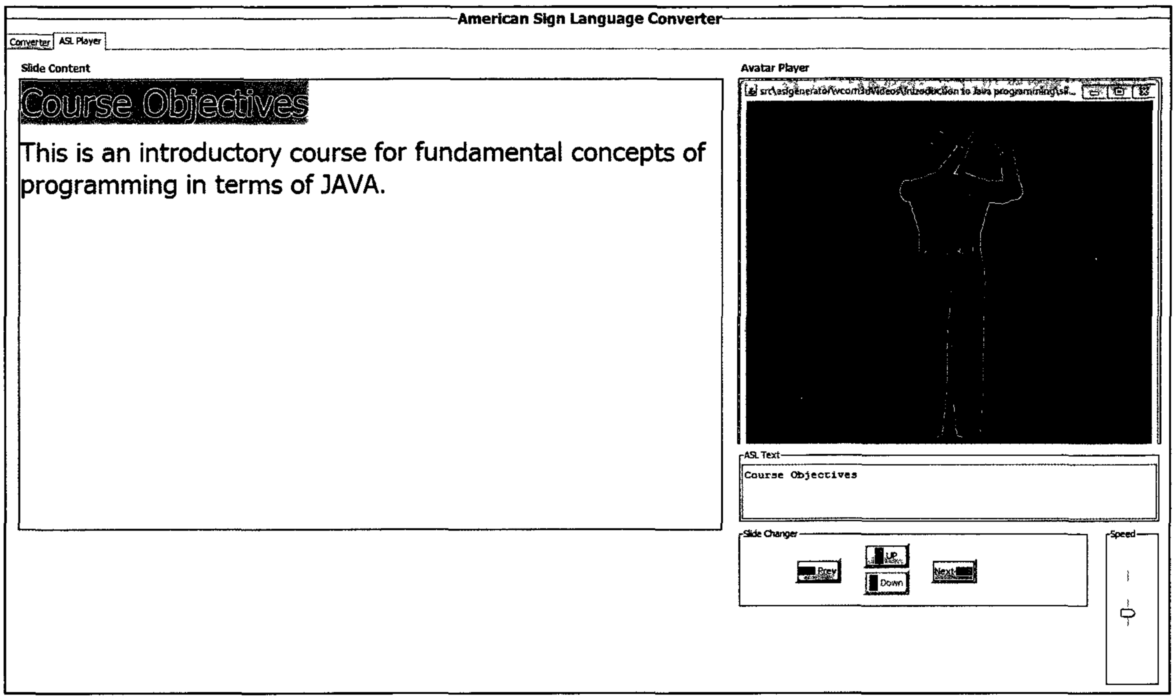
playMedia(String _mediaFile)	This method plays the videos from the given Uniform Resources Locator (URL) into the embedded internal frame by establishing a connection to the data source.
reload(Player player, String title)	This method reloads the new video into the internal frame, i.e., establishes a new connection to the data source.

The AsposePowerPointReader class in the application uses this package to interact with Microsoft PowerPoint slides. It reads the content from the slides in the presentation and displays it in a JTextPane on a Java based GUI. This class also tries to preserve the properties of the text in the presentation such as font size, font style etc. This class provides following methods:

Table 5.3: Method Description of Class AsposePowerPointReader

nextSlide()	This method loads the contents of the next Microsoft© Power Point slide of in the jTextPane.
prevSlide()	This method loads the contents of the previous Microsoft© Power Point slide of in the jTextPane.
moveDown()	This method highlights the next sentence in the current slide in jTextPane and starts playing the corresponding video.
moveUp()	This method highlights the previous sentence in the current slide in jTextPane and starts playing the corresponding video.

Once the translation is done the presentation and the avatar is displayed in a JFrame as shown in Figure 5.1.


American Sign Language Converter
Converter
ASL Player
Slide Content
Avatar Player
src\aslgenerator\wcom3dVideos\Introduction to Java programming\sl...
ASL Text
Slide Changer
Prev
UP
Down
Next
Speed

Course Objectives

This is an introductory course for fundamental concepts of programming in terms of JAVA.

```
Course Objectives
```

Figure 5.1: GUI to Display Slides and Signing Avatar

Chapter 6

Results

The existing tools, SignSmith from VCom3D (VCom3D 2006) and TESSA from ViSiCAST (ViSiCAST 2000), which are used for translation of English to ASL, perform a word by word conversion of the sentences which could be grammatically incorrect formation of sentence in ASL. Conversion of sentences via this application results in application of correct grammar rules on the sentence. The output produced by the application matches the sentence formation of the ASL (Stewart, Stewart and Little 2007), which has a correct syntax.

For example, the ASL translation of the English sentence, "Are you hiding because he is here?" by the application results into the sentence "because he here, you hiding?" In the original sentence, we can see that the topic is "he is here" and the comment is "are you hiding." In the translation process, the topic is placed in front of the comment and the Be-verbs are eliminated. Since this is a yes/no question, "are" is also eliminated from the sentence. Hence, the sentence correctly matches the syntax of ASL.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The Deaf and Hard of Hearing students lose interest in taking computer science courses because of the lack of an online repository that provides the signing gestures related to computer science. Many of these students are born deaf and their first language is ASL instead of English. All the teaching materials for computer science are prepared in English rather than ASL and there is no tool that translates English to ASL with correct usage of grammatical rules. Existing signing tools convert the English sentences word by word which does not convey the meaning to deaf students correctly. This application resolves these issues by introducing computer science related gestures and translation of English to ASL with correct usage of grammar. This application makes the converted ASL sentences more comprehensible to the deaf students. It takes input from the Microsoft PowerPoint file which is the most popular lecture presentation tool. The project provides a Graphical User Interface (GUI) which displays PowerPoint lectures and the animation on the same screen. Hence, this application is of great significance for Deaf and Hard of Hearing students.

7.2 Future Work

This project uses a third party tool, SignSmith Studio, for generating signing gestures and animation which does not allow to create facial expressions that are considered very important in ASL. Non-manual markers are implemented in this project. Hence, the creation of an animation toolkit would make the addition of facial expression to the signing avatar easier. Addition of non-manual markers will help in better translation of English to ASL sentences and express the information in a significant manner.

The AutoIt script automates the process of the translation and movie creation but makes the system much slower. Creation of a new animation toolkit will also remove the need of AutoIt script since the animation can be created directly by using flexible functionality of the toolkit. As a result, this application will not need to save movies of animation and will be able to display the real-time animation for a sentence.

References

AotuIt. 2010. "Automating the Windows GUI." Accessed March 2010. http://www.autoitscript.com/autoit3/index.shtml

Aspose. 2011. "Java PowerPoint Library to Read, Create and Mannipulate Presentations." Accessed March 2010. http://www.aspose.com

Baker-Shenk, Charlotte, and Dennis Cokley. 2002. *"American Sign Language Green Books, A Teacher's Resource Text on Grammar and Culture."* Washington: Gallaudet University Press.

eSign Project. 2004. "Essential Sign Language Information on Government Networks." Accessed March 2010. http://www.sign-lang.uni-hamburg.de/eSIGN

Fourm, ASL-STEM. 2009. "Viewing toic: Computer Science." Accessed Feburary 2010. http://aslstem.cs.washington.edu/topic/view/37

H|Anim. 2000 "Humanoid Animation Working Group." Accessed March 2010. http://h-anim.org

IBM's Extreme Blue Projects. 2007. "MQ Telemetry Transprot, Say It Sign It." Accessed Feburary 2010. http://mqtt.org/projects/sisi

Kelly, Charles, and Lawrence Kelly. 2010. "English Vocublary Wordlist. Adverbs of Time." Accessed March 2011. http://www.manythings.org/vocabulary/lists/a/words.php?f=adverbs_of_time

Lang, Harry G. 2002. "Research Priorities in New Millennium." *Journal of Deaf Studies and Deaf Education*. 267-280.

Lifeprint. 2010. "American Sign Language Resource Site, Nonmanual Markers in ASL." Accessed March 2011. http://www.lifeprint.com/asl101/pages-layout/nonmanualmarkers.htm

Mameffe, Marie-Catherine de, and Christopher D Manning. 2010. "Stanford Typed Dependencies." Accessed Feburary 2010. http://nlp.stanford.edu/software/dependencies_manual.pdf

Marcus, Mitchell P., Marry Ann Marcinkiewicz, and Beatrice Santorini.1993. "Building a Large Annotated Corpus of English." 313-330.

Marschark, M., and P.C. Hauser. 2008. *"Deaf Cognition: Foundations and Outcomes (Perspectives on Deafness)."* New York: Oxford University Press.

Oracle. 2010. "Java Media Framework (JMF)." Accessed April 2010. http://www.oracle.com/technetwork/java/javase/tech/index-jsp-140239.html

Pettibone, Jeanette. 2002. "Penn Treebank Tags." Computational Linguistics Program. Accessed Feburary 2011. http://bulba.sdsu.edu/jeanette/thesis/PennTags.html

Robbins, Curtis. 1996. "Computer Technology Education and the Deaf Student: Observations of Serious Nuances of Communication." Accessed Feburary 2011. http://people.rit.edu/easi/itd/itdv03n4/article3.htm

Rochester Institute of Technology. 2011. *"Technical Signs for Science and Mathematics."* New York: National Technical Institute of Deaf.

Sheryl, Cooper B. 1997. "The Academic Status of Sign Language Program in Instructions of Heigher Education in United States." *Coordinator of Deaf Studies/ Sign Language.* Accessed March 2011. http://pages.towson.edu/scooper/dissertation.html

Signing Savvy. 2003. ASL Sign Language Video Dictionary. Accessed March 2011. http://signingsavvy.com/

Stewart, David Alan, Elizabeth Stewart, and Jessalyn Little. 2009. *"American Sign Language, the Easy Way."* New York: Barron's Educational Series.

The Shodor Education Foundation. 2005 Deaf CS Home. Accessed Feburary 2010 http://www.shodor.org/succeed-hi

The Stanford Natural Language Processing Group. 2006. "The Stanford Parser: A Statistical Parser." Accessed March 2010. http://nlp.stanford.edu/software/lex-parser.shtml

Valli, Clayton, and Ceil Lucas. 2002. *"Linguistics of American Sign Language: An Introduction."* Washington: Gallaudet University.

VCom3D. 2006. "VCom3D Homepage." Accessed Feburary 2010. http://www.vcom3d.com

ViSiCAST. 2010. "ViSiCAST Project." Accessed March 2010. http://www.visicast.co.uk

Wikipedia. 2011. "American Sign Language." Accessed Feburary 2011. http://en.wikipedia.org/American_Sign_Language

Appendix A

Penn Treebank POS Tags

This section provides a brief description of all the clause level and phrase level POS tags used by the Stanford Parser and the Penn Treebank.

List of Phrase Level POS Tags

Tag	**Explanation**
ADJP	Adjective phrase
ADVP	Adverb phrase
CONJP	Conjunction phrase
FRAG	Fragment
INTJ	Interjection. Corresponds approximately to the part-of-speech tag UH.
LST	List marker. Includes surrounding punctuation.
NAC	Not a Constituent; used to show the scope of certain prenominal modifiers within an NP.
NP	Noun phrase
NX	Used within certain complex NPs to mark the head of the NP. Corresponds very roughly to N-bar level but used quite differently.
PP	Preposition phrase.
PRN	Parenthetical.

PRT	Particle. Category for words that should be tagged RP.
QP	Quantifier Phrase (i.e. complex measure/amount phrase); used within NP.
RRC	Reduced Relative Clause.
UCP	Unlike Coordinated Phrase.
VP	Verb Phrase.
WHADJP	Wh-adjective Phrase. Adjectival phrase containing a wh-adverb, as in how hot.
WHADVP	Wh-adverb Phrase. Introduces a clause with an NP gap. May be null (containing the 0 complementizer) or lexical, containing a wh-adverb such as how or why.
WHNP	Wh-noun Phrase. Introduces a clause with an NP gap. May be null (containing the 0 complementizer) or lexical, containing some wh-word, e.g. who, which book, whose daughter, none of which, or how many leopards.
WHPP	Wh-prepositional Phrase. Prepositional phrase containing a wh-noun phrase (such as of which or by whose authority) that either introduces a PP gap or is contained by a WHNP.

List of Clause Level POS Tags

S	Simple declarative clause, i.e. one that is not introduced by a (possible

	empty) subordinating conjunction or a *wh*-word and that does not exhibit subject-verb inversion.
SBAR	Clause introduced by a (possibly empty) subordinating conjunction.
SBARQ	Direct question introduced by a *wh*-word or a *wh*-phrase. Indirect questions and relative clauses should be bracketed as SBAR, not SBARQ.
SINV	Inverted declarative sentence, i.e. one in which the subject follows the tensed verb or modal.
SQ	Inverted yes/no question, or main clause of a *wh*-question, following the *wh*-phrase in SBARQ.

Appendix B

List of Translated Words

Programming	case	public	machine code
Software	catch	protected	Interpreter
Class	char	Private	bytecode
Compilation	const	package	compiler
Execution	continue	long	CPU
OOP	default	interface	volatile
method (java)	double	int	Try
void	enum	import	transient
webpage	extends	goto	this
functional	finally	float	throws
identifier	final	finally	Switch
abstract	float	final	synchronized
assert	import	implements	Super
Boolean	goto	extends	static
break	continue	enum	return
byte	default	double	short
architecture neutral	jdk	syntax rules	semantics
UML	testing	object	inheritance
String	print	println	escape sequence

variable	primitive data type	character set	unicode
ascii	expression	operator precedance	expression tree
postfix	prefix	casting	scanner
instantiation	reference	assignment	alias
concat	toUpperCase	replace	Random
NumberFormat	DecimalFormat	ordinal	wrapper class
flow of control	AND	NOT	OR
indentation	block statements	lexicographic ordering	while
palindrome	iterator	URL	for
main	toString	constructor	scope
GUI	events	container	Listener
Aggregation	overloading	Initializer List	Command Line argument
body	parameter	Driver program	Dependancy
Modifier	Accessor	Mutator	Header
Instance Data	UML Diagram	encapsulation	black box
Components			

Appendix C

AutoIt Script

```
#Script to automatically create animated movie from VCom3D#

WinActivate("[CLASS:com.vcom3d.sss.u]")

;WinSetState("[CLASS:com.vcom3d.sss.u]", "", @SW_SHOW)

ControlSend("[CLASS:com.vcom3d.sss.u]","","","^i");

WinSetState("Save?", "", @SW_HIDE)

ControlSend("Save?","","","!n");

WinWaitActive("[CLASS:#32770]")

ControlSetText("[CLASS:#32770]", "", "[CLASS:Edit; INSTANCE:1]", "
C:\Users\Prashant\Documents\Spring 11\Thesis\Project\input.dat")

ControlClick("[CLASS:#32770]", "", "[CLASSNN:Button2]")

sleep(8000)

;WinActivate("[CLASS:com.vcom3d.sss.u]")

ControlSend("[CLASS:com.vcom3d.sss.u]","","","!f");

ControlSend("[CLASS:com.vcom3d.sss.u]","","","e");
ControlSend("[CLASS:com.vcom3d.sss.u]","","","m");

WinWaitActive("[CLASS:#32770]")

ControlSetText("[CLASS:#32770]", "", "[CLASS:Edit; INSTANCE:1]", "
C:\Users\Prashant\Documents\Spring 11\Thesis\Project\output.avi")

ControlFocus("[CLASS:#32770]", "", "[CLASS:Edit; INSTANCE:1]")

ControlSend("[CLASS:#32770]","","","{TAB}");

ControlClick("[CLASS:#32770]", "", "[CLASS:Button; INSTANCE:1]")

WinActivate("[CLASS:SunAwtDialog]")

WinWaitNotActive("[CLASS:SunAwtDialog]")

sleep(10000)

;WinClose("[CLASS:com.vcom3d.sss.u]")

;WinSetState("Save?", "", @SW_HIDE)

;ControlSend("Save?","","","!n");
```