Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

New Jersey's Science & Technology University

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

ABSTRACT

PARAMETRIC SYNTHESIS OF SIGN LANGUAGE

by
Jerome Allen

The isolation of the deaf community from mainstream society is in part due to the lack of knowledge most hearing people have of sign language. To most, there seems to be little need to learn a language that is spoken by such a small minority unless perhaps a relative is unable to hear. Even with a desire to learn, the task may seem insurmountable due to the unique formational and grammatical rules of the language.

This linguistic rift has led to the call for an automatic translation system with the ability to take voice or written text as input and produce a comprehensive sequence of signed gestures through computing.

This thesis focused on the development of the foundation of a system that would receive English language input and generate a sequence of related signed gestures each synthesized from their basic kinematic parameters. A technique of sign specification for a computer-based translation system was developed through the use of Python objects and functions. Sign definitions, written as Python algorithms, were used to drive the simulation engine of a human-modeling software known as Jack. This research suggests that 3-dimensional computer graphics can be utilized in the production of sign representations that are intelligible and natural in appearance.

PARAMETRIC SYNTHESIS OF SIGN LANGUAGE

by

Jerome Allen

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Biomedical Engineering

Department of Biomedical Engineering

May 2004

APPROVAL PAGE

PARAMETRIC SYNTHESIS OF SIGN LANGUAGE

Jerome Allen

Dr. Richard A. Foulds, Thesis Advisor Associate Professor of Biomedical Engineering, NJIT	Date
Dr. Sergei Adamovich, Committee Member Associate Professor of Biomedical Engineering, NJIT	Date
Dr. William Hunter, Committee Member Professor and Chair of Biomedical Engineering, NJIT	Date

BIOGRAPHICAL SKETCH

Author:	Jerome Allen
Degree:	Master of Science
Date:	May 2004

Undergraduate and Graduate Education:

- Master of Science in Biomedical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2004

- Bachelor of Science in Biomedical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2002

Major: Biomedical Engineering

Presentations and Publications:

Jerome M. Allen and Richard A. Foulds,
"An Approach To Animating Sign Language: A Spoken English To Sign English Translator System,"
Proceedings of the IEEE 30th Annual Northeast Bioengineering Conference, Springfield, MA, April 2004.

Jerome M. Allen and Pierre K. Asselin, Richard A. Foulds,
"American Sign Language Finger Spelling Recognition System,"
Proceedings of the IEEE 29th Annual Northeast Bioengineering Conference, Newark, NJ, March 2003.

To my parents, Victor and Jean Allen,
and my sister, Martine Allen,
whose love and support seem boundless

ACKNOWLEDGMENT

I would like to express my deepest gratitude to Dr. Richard Foulds for his support and guidance while serving as my research supervisor. His infectious enthusiasm and unwavering belief in my abilities to accomplish whatever I set out to do, has been truly inspiring. Special thanks to Dr. Sergei Adamovich and Dr. William Hunter for taking interest in my research and for serving as committee members.

To Darnell Simon, Don Helt III, Gillian Sherry, Bruno Mantilla, Matt Noesner and Corey Birmingham, my fellow graduate students in the Neuromuscular Engineering Laboratory, thank you for your support and friendship over the years and for helping to keep sometimes stressful work enjoyable.

I would like to thank those closest to me; my family and friends. I have been extremely fortunate to be blessed with family and friends whose love and support seem boundless. My sincerest appreciation to my parents, Jean and Victor Allen, and my sister Martine Allen, any success on my part is a reflection of your unconditional love, undying support, and sacrifice. Special thanks to Pierre Asselin and Rehana Kassim for their constant encouragement and necessitous comic relief.

Finally, I would like to gratefully acknowledge the support of the Millbank Foundation and the Pfeiffer foundation throughout this research, which was conducted at the Neuromuscular Engineering Research Laboratory at NJIT.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

CHAPTER 1

INTRODUCTION

1.1 Objective

American Sign Language (ASL), like any other language is one that has its own linguistic structure and grammatical rules, as well as its own pool of native users. It is the natural language of North Americans who are congenitally deaf or who lost their hearing as young children [1].

Learning a spoken language is extremely difficult for someone who is unable to hear, and as a result, deaf individuals experience difficulty speaking and reading English if their deafness is present in their early years of life. Similarly, learning sign language later in life is an arduous endeavor for hearing people and the majority seem to have no immediate reason to learn the language. Consequently, there is an enormous demand for ASL translators. These interpreters however, are costly to hire, limited in number, and usually available only in formal settings. This clash of interests and ability has resulted in the isolation of the Deaf community as a linguistic minority within mainstream society.

It is believed that the creation of a computer translation system could assist tremendously in the social integration of the Deaf community, while still preserving the use of their native language. Such technology may not achieve the capability of a human interpreter, but would be practical for application in casual settings such as grocery stores, hotel lobbies, gas stations, and cafés, where interpreters are not usually available.

One side of this full translation system would entail an oral/written language to sign language translation apparatus. Such an apparatus that would take acoustic or textual information as input and produce a visual sequence of signed gestures. The acoustic information, in the form of a person's voice may be acquired through the use of speech recognition technology. The spoken words would be converted to text, which is then recognized, processed and used to synthesize the appropriate signed gestures (Figure 1.1).

The goal of this thesis is to develop the foundation of a system that would receive English language input and generate a sequence of signed gestures each synthesized from their kinematic parameters. The term "synthesis" means the generation of original sign language messages without prior recording of the sign. The signed gestures are conveyed though the use of a three dimensional avatar that is realistic and human-like in both movement and appearance.

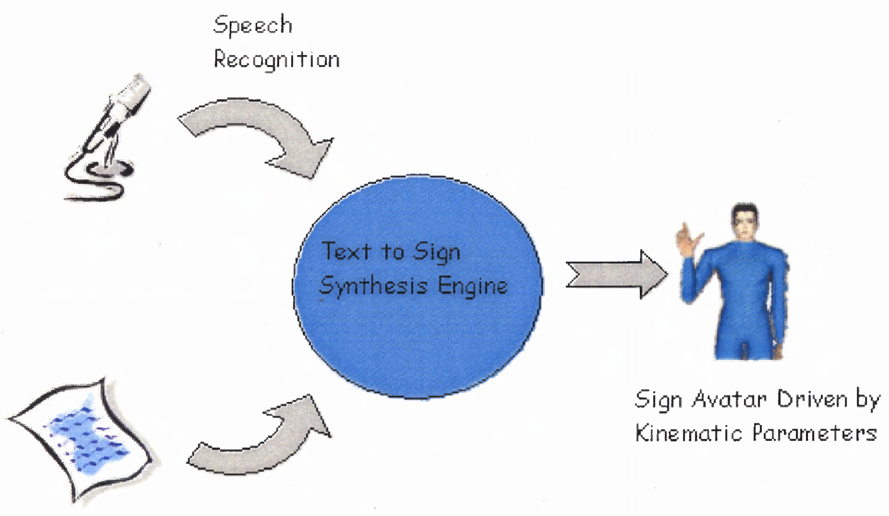


Figure 1.1 Basic Architecture of English to Sign Language System.

1.2 The Ear and Deafness

1.2.1 The Hearing System

Hearing is a complex process in which sound waves are converted to mechanical movement and then to neural impulses, which are processed by the brain. The first step in hearing is the entrance of sound waves into the pinna, which catches sound waves and directs them down the ear canal. The sound waves fill the external auditory canal with the continuous vibrations of pressure waves. These waves push air molecules against the eardrum, causing it to vibrate at the same frequency as the sound wave. These vibrations are passed across the middle ear by the malleus, the incus, and the stapes. These bones amplify the vibrations before passing them into the cochlea via the oval window. As the vibrations from the bones in the middle ear enter the cochlea they cause movement in the fluid. This in turn causes the hair cells to bend. As the hair cells move they create a small electrical current, which triggers an action potential to move along the auditory nerve to the brain where they are converted into signals that can be understood as sound (Figure 1.2).

For an individual to hear properly, all of these elements must work well. Deafness happens when at least one part of this system is not working effectively [2,3].

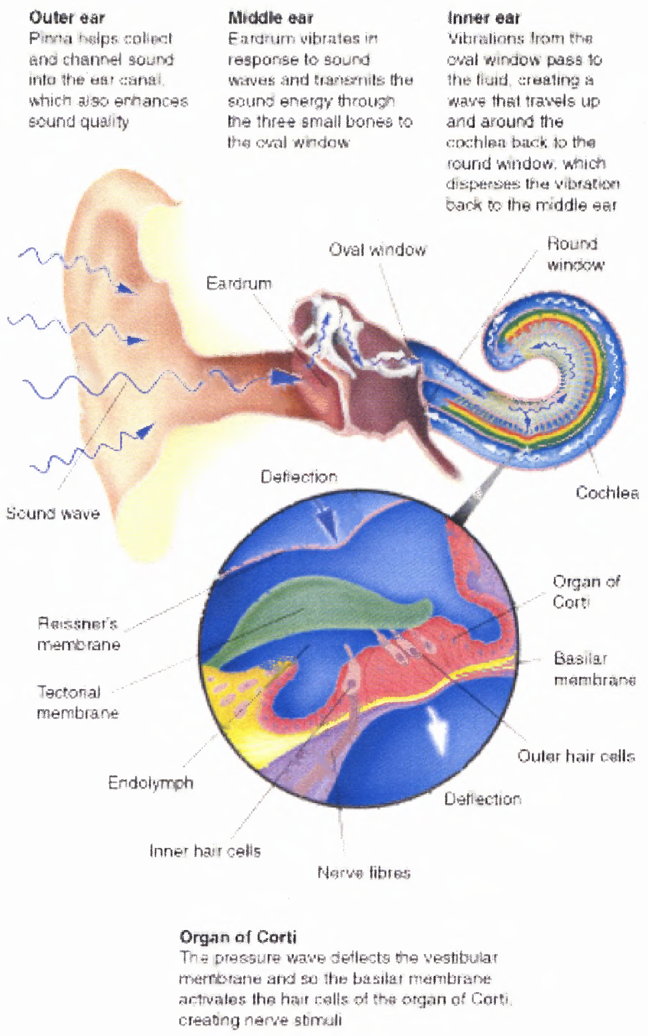

Outer ear
Pinna helps collect and channel sound into the ear canal, which also enhances sound quality
Middle ear
Eardrum vibrates in response to sound waves and transmits the sound energy through the three small bones to the oval window
Inner ear
Vibrations from the oval window pass to the fluid, creating a wave that travels up and around the cochlea back to the round window, which disperses the vibration back to the middle ear
Oval window
Round window
Eardrum
Deflection
Cochlea
Sound wave
Organ of Corti
Reissner's membrane
Basilar membrane
Tectorial membrane
Outer hair cells
Endolymph
Deflection
Inner hair cells
Nerve fibres
Organ of Corti
The pressure wave deflects the vestibular membrane and so the basilar membrane activates the hair cells of the organ of Corti, creating nerve stimuli

Figure 1.2 The mechanism of hearing [4].

1.2.2 Types and Levels of Deafness

Deafness is loss of hearing of any degree, and may be slight or severe, temporary or permanent. It may affect a person's ability to hear sounds of a certain frequency, or sounds at particular intensities. Whatever the level of hearing loss, the condition is usually classified as being one of two main types: conductive deafness or sensorineural deafness [5].

Conductive deafness is the most common type of deafness. It occurs when something interferes with the conduction of sound to the fluids of the inner ear. There are a plethora of things that could cause this to happen, including the physical blockage of the ear canal with earwax, rupture of the eardrum, middle ear infections with accompanying fluid accumulation, or restriction of the ossicular movement because of bony adhesions between the stapes and oval window [6].

Sensorineural deafness on the other hand is caused by problems within the inner ear or auditory nerve. It results from damage to the neural structures at any point from the cochlear hair cells to and including the auditory cortical cells. This type of deafness may be partial or complete but is typically permanent. There are also cases where an individual may experience a combination of both conductive and sensorineural deafness, often referred to as mixed deafness [7].

Regardless of the type of deafness, every individual falls within one of four levels of deafness. Individuals are classified as being mildly, moderately, severely or profoundly deaf (Figure 1.3).

- Someone that suffers from mild deafness may be able to hear sounds ranging between 24 and 40 decibels (dB) on average in their better ear. They usually find is difficult to follow speech in situations where there is a lot of background noise.
- A person with a moderate level of deafness may hear sounds between 40 and 70 decibels. They usually find it difficult to follow speech without a hearing aid or other sound amplification technology.

- Severe deafness affects individuals who may only hear sounds within the 70 to 95 decibel range. Children with a severe level of deafness may rely heavily on lipreading or use sign language as a mode of communication.
- The fourth level, profound deafness, is one in which a person can only hear sounds of around the 95 decibel level or greater on average in their better ear. Children suffering from profound deafness may rely heavily on lipreading and/or a method of sign language.

Levels of deafness:

Mild deafness
20 - 40 dB

Moderate deafness
41 - 70 dB

Severe deafness
71 - 95 dB

Profound deafness
95+ dB

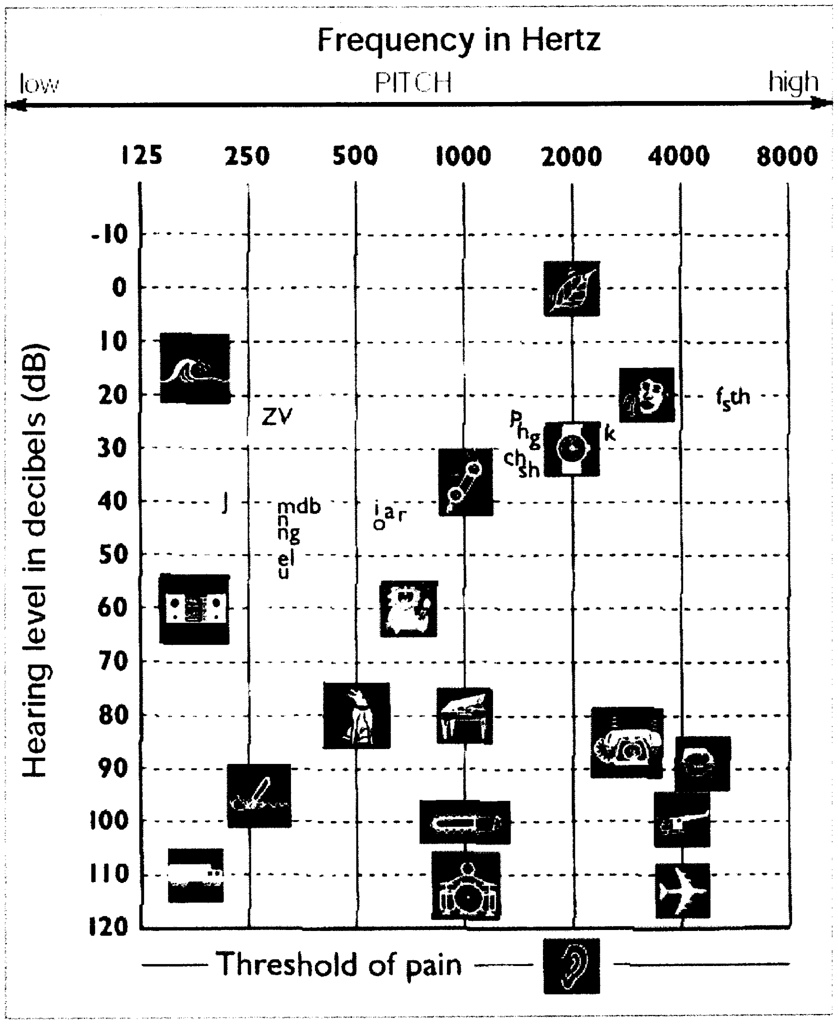


Figure 1.3 Representation of loudness and pitch of everyday sounds [3].

It is not always possible to identify the reason a person is deaf. Some children are born deaf (congenital deafness), while others become deaf later in life (deafened). Many children in early childhood may become deaf due to infections such as meningitis, mumps and measles. Congenital deafness occurs prior to birth due to complications during pregnancy, or illnesses such as rubella, cytomegalovirus, toxoplasmosis, and herpes. There also is a class of medicines, known as ototoxic drugs, which can damage the hearing of a baby before or after birth [3].

The main cause of congenital deafness in children however, is heredity. Deafness can be passed down in families even though there appears to be no immediate family history of deafness (about 90 percent of deaf children are born to hearing parents) [7]. When this occurs, hearing loss is usually significant and permanent. These children suffer from moderate to profound hearing loss, and tend to develop auxiliary modes of communication [7].

1.3 Communication and Language

1.3.1 Methods of Communication

Developing good communication technique is vital for any child or person whether they can hear or not. With communication skills children can express themselves, as well as influence and learn from others. This truth is no different for deaf children, however sometimes the methods used to communicate may be different from those of hearing children.

For those with residual hearing, the ability to develop listening skills and spoken language is possible with the assistance of hearing aids, and other forms of amplification. Cochlear implants are now being used in profoundly deaf individuals. Such neural prostheses are extremely invasive however; and though this approach may improve their hearing, the quality leaves much to be desired.

Lipreading/speechreading is also one of the techniques used by the Deaf to facilitate communication. Some deaf people nurture the ability to read lip patterns. This approach requires an in-depth understanding of spoken language and the ability to read lip patterns. Since only a third of words and speech sounds can be lipread under the best conditions, it is not possible to distinguish between all the different parts of speech from lipreading alone [3,7]. Effective lipreading, relies on residual hearing to provide cues related to vocal tract movements.

A large population of the deaf community, especially those with severe to profound deafness, use a form of sign language to communicate. Sign languages are languages in a visual context that are capable of the same intellectual, expressive, and social functions as a spoken language. Instead of being based on signals produced by the voice and perceived by the ear, these systems are based on signals produced by the hands and perceived by the eye [8]. Sign Languages are gestural-visual systems that have taken their own course of development separate from spoken languages. In the United States nearly 500,000 deaf people use a form of sign language known as American Sign Language (ASL or Ameslan) [1], while in Britain an estimated 70,000 people use British Sign Language [3]. These languages and others like them use handshapes, facial expressions, gestures and body language to communicate.

1.3.2 American Sign Language

American Sign Language (ASL) is the natural language of approximately half a million deaf people in the United States and in the English speaking parts of Canada [1,9]. It is the native language of many deaf people who have deaf parents, and is the language used by many deaf adults among themselves. Increasingly, hearing people are learning ASL as a second language, making it the third most commonly used language in the United States [9].

For deaf people who use ASL, the language is a bond that unites them as part of the deaf community. They are also connected through similarities in their attitudes and beliefs about themselves and the world around them. They tend to view themselves as sharing a common cultural experience, and take pride in their rich heritage. This strong sense of identity in the deaf community is nurtured by the use of ASL.

There are certainly definite reasons to be proud of this form of language. ASL is a legitimate language that has undergone hundreds of years of development by deaf people and their advocates. In fact, American Sign Language has roots in French Sign Language (FSL), which was brought to the United States in 1817 by Thomas Gallaudet and Laurent Clerc, a deaf teacher from France [1]. These educators brought FSL to the United States, while others spread FSL throughout Europe, influencing such sign languages as Swedish, Latvian, Irish, Spanish, Dutch, Italian, Swiss, Austrian, Russian, and eventually, Australian [10].

One of the most common fallacies in regard to American Sign Language is the belief that it is a visual-gestural language based on English. However, it is not derived from any spoken language. Instead, it is a distinct language with its own unique set of grammatical and syntactical rules. It is as capable as any other spoken language of communicating complex and abstract ideas. English, however, can be used to describe ASL just as it can be used to describe any other language. This description could give some insight to the differences between American Sign Language and English. Below are two sentences, each written in both ASL and English. These sentences will give some idea of the difference in grammar and syntax between the two languages.

English	There is food in the store.
ASL	IT STORE HAVE FOOD.

English	He gave her a trophy.
ASL	TROPHY HE-PRESENT-HER [11].

In addition, in ASL signs can be arranged in a variety of different orders to form the same sentences. For instance, the sentence "I am happy" may be signed as "I happy I", "I happy", or "Happy I". Furthermore, there is no distinction between the different grammatical forms for nouns, adjectives and adverbs. For example, "happy", "happiness", and happily" are all still signed exactly alike, so the appropriate meaning must be determined from the context [9].

In ASL, one sign may represent a number of different ideas. And so, it is necessary to incorporate supra segmentals, such as facial expressions and the appropriate acceleration or velocity of a sign to convey its intended meaning. These non-manual characteristics are just as important as the actual signs made by the hands.

Since ASL is so different in grammar and syntax from English, new sign systems have been developed in the past twelve years, with the intent of better facilitating deaf children's acquisition of the English Language. These sign systems make an effort to represent English Language grammar as best they can. They are collectively referred to as Manually Coded English (MCE).

1.3.3 Manually Coded English Systems

Manually Coded English is a general term used to refer to all sign systems devised to represent English grammar. These are most often used in deaf schools to teach its students English grammar. It is also common in schools since hearing teachers may have learned sign language as adults and impose English grammatical structure on their signing. The two more frequently used systems are Signed English and Signing Exact English [9]. Theses systems include representations of plurals, tenses, and articles all signed in English Language word order.

Since in Manually Coded English Systems each sign produced is used to represent an English word rather than a complete concept, these systems are thought of as codes rather than a language. The signs of ASL are used, but they are employed in an order indicating English Language grammar. A few signs have been created, however, for frequently used English words that had no corresponding sign in American Sign Language.

1.3.4 Fingerspelling

Fingerspelling is the use of a manual alphabet to represent English words. It is used as an adjunct to all of the systems described in the sign language continuum (see Table 1), including previously mentioned signing systems. In these systems, each of the twenty-six letters of the alphabet is represented by a particular shape of the hand and fingers. Fingerspelling is used in ASL to indicate words that do not have a prescribed sign associated with, such as, names of places, people, and things. It is also understood that fingerspelling is sometimes used to substitute for signs that are either forgotten, or not learned as yet.

When learning sign language fingerspelling is one of the most important bases. It helps teach many of the handshapes used in everyday signs. Fingerspelling is actually the main component of a sign system known as the Rochester Method, where each and every word is fingerspelled.

Table 1.1 Different Sign Language Methods and Their Relationship to English [12]

Sign Language Continuum			
	American Sign Language (ASL)	Manually Coded English (MCE)	Fingerspelling
Characteristics	Combines standard sign, fingerspelling, and elements of pantomime; has a syntax of its own; is ideographic and idiomatic; follows one sign-one concept rule	Combines some standard signs with invented and adapted ASL signs; represents visually the syntax of English, including inflections, suffixes, and prefixes; follows one sign – one word rule	Is a letter-by-letter representation of English
Sign Systems	American Sign Language (ASL), also known as Ameslan	Signed English Seeing Essential English (SEE I) Signing Exact English (SEE II) Linguistics of Visual English (LOVE)	Rochester Method Visible English
Relation to English	Some representation of English Elements	Complete representation of English elements	Complete representation of English

CHAPTER 2

BACKGROUND ON SIGN PRODUCTION AND ANIMATION

2.1 Technologies Used to Present Sign Language Digitally

The social integration of deaf communities around the world has been the focus of much research. The past several decades have seen several attempts to create assistive technologies that aid in the presentation and translation of sign language. The underlying idea is that improved techniques for teaching and translating sign language will help narrow the linguistic gap between the deaf and hearing worlds.

There are several techniques that have been used to generate sign language, either for educational purposes or as part of translation systems. These techniques include: still image technology, digital movie clips, key frame computer animation, and motion capture technology. Both key frame animation and motion capture utilize three-dimensional computer graphics.

2.1.1 Still Image Technology

Representing sign language by displaying still images is an approach that has been taken in both educational and sign translation systems. These systems usually produce a sequence of still images from an image database that correspond to the appropriate sign pattern. The images, whether photographic or illustrated, are digitized so that they may be displayed on a computer [13, 14, 15].

The most basic forms of this technology simply display still images of fingerspelled letters in a sequence corresponding to text that is entered [14, 15]. This can then be read by someone who knows, or is learning the manual alphabet. Similarly, there are educational software packages that represent ASL words using animated GIFs

(Graphic Interchange Format), which are merely compressed sequences of still images [16].

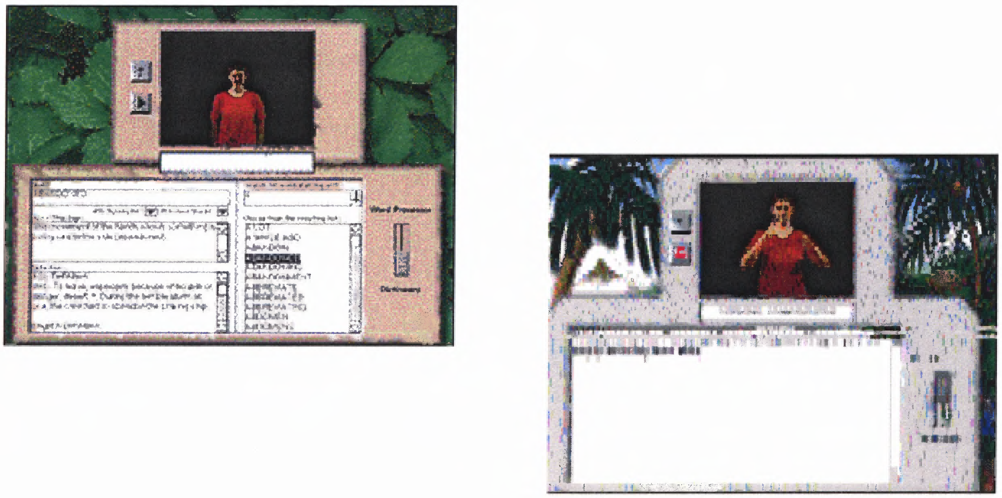
A sign language generation system, which uses still image technology, was proposed by Hideo Kawai and Shinichi Tamura of the Osaka University in Japan. The system was developed to recognize speech and generate the corresponding animation-like sign language sequence in real-time. It is implemented on a personal computer, which has both video RAM and a voice recognition board. Image files representing sign language patterns and fingerspelling are stored on a disk. When text is received from the keyboard or the voice recognition circuit board, an analogous sequence of sign language patterns and/or fingerspelling patterns is generated and displayed on the computer monitor. Each sign pattern is composed of one to four sub-pattern still images, and is read from the disk and transferred to the video RAM. Consequently, spoken language is displayed as sign language animation [13].

2.1.2 Movie Clip Technique

A very similar but improved technique employs digital movie clips as an alternative to using still images. Here, video of a personal signer is played according to words entered or selected on the system.

One such system is "The Personal Communicator" developed by Michigan State University (Figure 2.1). The software package was originally written in a language known as Hypercard, and was eventually redesigned and ported to Macromedia Director, in order to make it a cross-platform product. The software used digital video and compression technology to record and store over 2500 video clips of signs on a CD-ROM.

"The Personal Communicator", is a tool both for learning and communicating American Sign Language. It was created with the intention of helping deaf children and their hearing peers communicate better. It utilizes word processing techniques to convert typed English text to American Sign Language signs and Speech for learning or communication purposes. The program however does not tackle grammar or syntax conversion issues [17].



(a) (b)

Figure 2.1 The Personal Communicator: (a) English-ASL Dictionary, (b) Word Processor [17].

2.1.3 Key Frame Computer Animation

In an attempt to better visually document sign language, and effectively represent its three dimensional aspects, some researchers opt to use three dimensional computer graphics to represent the language; as opposed traditional two dimensional media such as movie clips and pictures.

Some systems with this focus are mainly concerned with teaching the language. Three-dimensional models give users the ability to view signs from different visual perspectives, and hence better facilitate the learning process.

An example of this is a system for teaching the manual alphabet on the Internet proposed by Geitz et al. The system uses VRML (Virtual Reality Modeling Language) to create three-dimensional static handshapes representing the manual alphabet. Each VRML file represents a single handshape, and is placed on a Web server accessible to clients over the World Wide Web. These client computers are equipped with Web browsers, as well VRML browsers, in order to be able to view the handshape files [18]. Users are able to go online and observe the subtle differences in handshapes by rotating the hand models in 3-dimension. The system was envisioned as a means to reach and help people in remote locations better learn sign language. However, it also possesses potential as a classroom aid since it allows students to study handshapes from any perspective.

A very similar but more comprehensive system is described by Ling and Long in their paper “An On-line Sign Language Communication System” [19]. This system uses a full-body VRML model, and tackles the problem of describing dynamic signs by employing the use of key frame animation. The overall system consists of an editor applet for creating animation sequences (Figure 2.2), and a viewer applet for viewing the Sign Language animation files (Figure 2.3). The animation files and the VRML human model are stored on a web server that is accessible all users over the Internet. Users defining animation sequences do so by specifying selected static gestures as key frames, and then rely on the system to generate the in-betweens. Users can use the system to learn sign language, as well as to communicate by simply selecting the animation files that they desire their comrade to view.

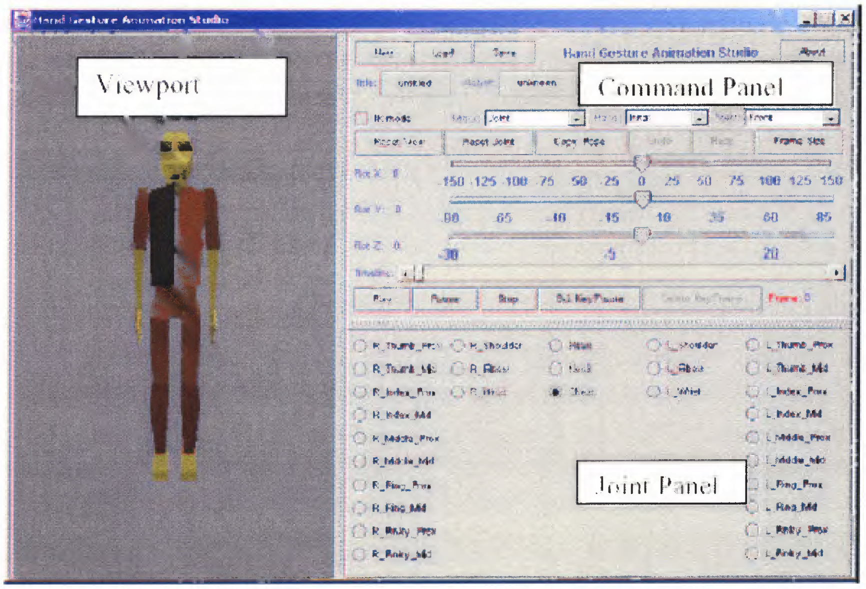
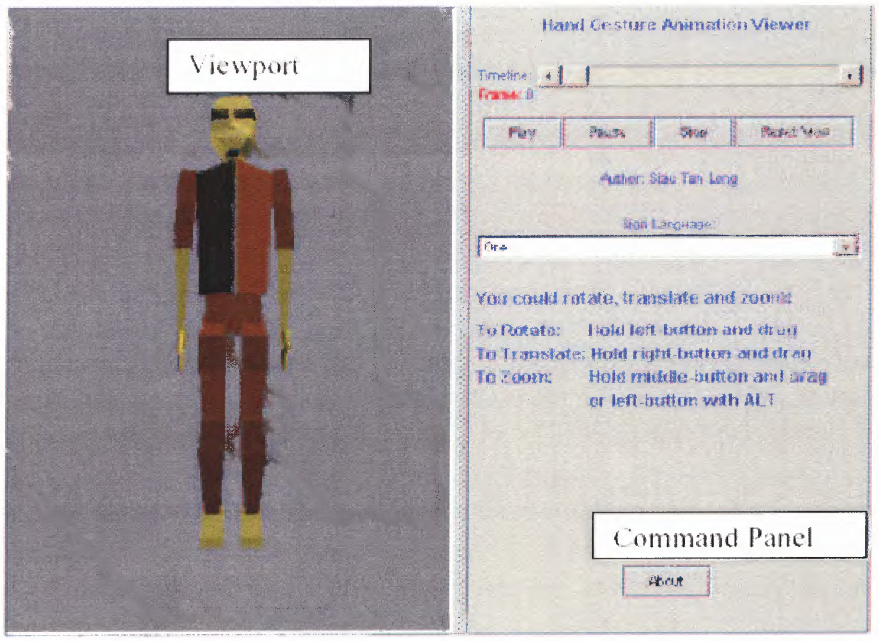


Figure 2.2 Editor applet of Hand Gesture Animation Studio [19].



You could rotate, translate and zoom it

To Rotate: Hold left-button and drag
To Translate: Hold right-button and drag
To Zoom: Hold middle-button and drag or left-button with ALT

Figure 2.3 Viewer applet of Hand Gesture Animation Studio [19].

2.1.4 Motion Capture Technology

Motion capture is yet another technique for animating virtual human models. Its proponents usually apply it when they desire realistic human-like movements from a virtual actor.

There are a large number of Sign Language translation and animation systems that utilize this technique, as opposed to key frame animation. One such system is being developed in Tokyo, Japan. The system helps to support conversation between hearing people and people with hearing impairments. The sign language representation portion of the system drives an avatar by regenerating motion data obtained by an optical motion capture system [20]. While capable of presenting very realistic signs, motion capture based animations are not easily expanded. Adding additional signs require extensive production capabilities

2.2 Advantages of Computer Graphics Sign Language Generation

As mentioned in previous sections of this chapter, the use of three-dimensional computer graphics in sign generation stands to better facilitate the acquisition of the language, due to its more accurate representation of the true nature of the language over more traditional two-dimensional media. However, this is not the only accolade that sets computer graphics aside from the rest. Apart from its ability to allow users to view signs from different angles, computer graphics give a sign translator system the flexibility to blend movements of an avatar based on kinematic rules. For instance, as opposed to using key frame or motion capture animation, a sign translator system could use inverse kinematic calculations based on certain given parameters to determine the intricacies of a movement. Furthermore, this mechanism is easily applied to interpolating between individual dynamic signs to produce a fluid transition from one sign to another. This quality of continuous and connecting movements is extremely important when considering realism in a translator. Translator systems (such as those using any of the four previously mentioned techniques) that simple play animation files or display still images, lack this realism since there is no interpolation involved to create the natural movements necessary to transition from one sign to another.

Through using computer graphics and algorithms to generate signs from their basic kinematic parameters, new signs can be easily synthesized by combining the appropriate predefined parameters. This will also lead to a consistent signing pattern, which users' will become accustomed to and remember.

2.2.1 Sign Animation versus Sign Synthesis

It is now possible to synthesize signs as opposed to merely animating them. It is important to distinguish between sign animation and sign synthesis. Animation is the production of motion or movement through the creation of artificial moving images, such as in cartoon and movie clips. The use of motion capture and key frame animation in computer graphics is also animation. For each of these, if new signs are needed in a system, new videotaping, new motion capture, or new frame-by-frame images need to be created. Sign Synthesis on the other hand, is the generation of sign language by combining its separate elements or parameters to produce the intended sign representation. These parameters are discussed in the following chapter.

CHAPTER 3

IMPORTANT ASPECTS OF CONSIDERATION IN SIGN SYNTHESIS

3.1 The Input and Output Language

In developing a computerized system to translate English into Sign Language, it is important to realize that the English language has two basic forms, a spoken form, and written form. Hence, it is desirable that a translator system accommodate both forms of the language as input. It should be able to acquire input data whether it is textual or acoustic in nature.

The first step towards this end, however, is to ensure that the more primitive of the two can be translated. As a result, English text is the primary form of input to the system that was designed. Entire sentences, and even paragraphs, can be entered into the translator via the keyboard, or through cutting and pasting.

It is possible to acquire acoustic information, in the form of spoken English, with the aid of speech recognition software. Such software can easily be integrated with the translator application, and used to convert spoken English into written English, which is then processed and translated. At present however, speech recognition technology does not yet accommodate speaker independent continuous recognition, and so this feature is best left for future implementation when the technology becomes more advanced.

In order to avoid the difficulties of grammar translation between English language and American Sign Language (ASL), a compromise was struck between the two. The prototype system temporarily employs Signed English as its output language instead of American Sign Language. This ensures focus on the matter of immediate importance, the parametric synthesis of signs, and not on the development of language processing techniques.

Signed English is one of the methods of communicating with the deaf. It is taught in many deaf schools around the country as a manual interpretation of English using fingerspelling and signs from American Sign Language. Signed English executes a sign for every word in an English sentence, whereas ASL would merely seek to convey the idea or concept instead [21]. By using Signed English as the output language for the translator system, focus is geared toward the synthesis of intelligible signs.

It is important to acknowledge that deaf signers, even without learning Signed English in school, are able to understand variations in sign language syntax. This is evident since most deaf signers see and understand ‘bad signing’ by hearing people, which is really ASL signs in an English sentence structure.

3.2 Software

3.2.1 Jack 3.0 – The Synthesis Engine

In order to realistically mimic human motion and appearance when generating signs, the Neuromuscular Engineering Laboratory of the New Jersey Institute of Technology, employs the use of a computer animation tool known as Jack (see Figure 3.1).

Though it may be used to generate sign language, Jack was developed at the University of Pennsylvania as human modeling and ergonomics analysis software. It is Jack’s extensibility and capacity as a powerful interactive, real-time visual simulation solution, that enables it to lend its services as a sign synthesis engine.

Jack gives its users the ability to manipulate virtual humans that can be either chosen from a menu of predefined human figures, or created based on anthropometric data. It provides the industry's most biomechanically accurate human models, which are based on body dimension measurements taken from the 1988 Anthropometric Survey of U.S. Army Personnel (ANSUR 88). Jack human figures [22]:

- have 69 segments, 68 joints, a 17-segment spine, 16-segment hands, coupled shoulder/clavicle joints and 135 degrees of freedom
- obey joint limits derived from NASA studies (Anthropometric Source Book, Vol. 2: A Handbook of Anthropometric Data, Technical Report NASA RP-1024)
- can be represented as stick figures, wireframe, shaded, high-resolution or transparent model

A detailed list of all of Jack's segments, joints and sites can be viewed in Appendix C of this thesis.

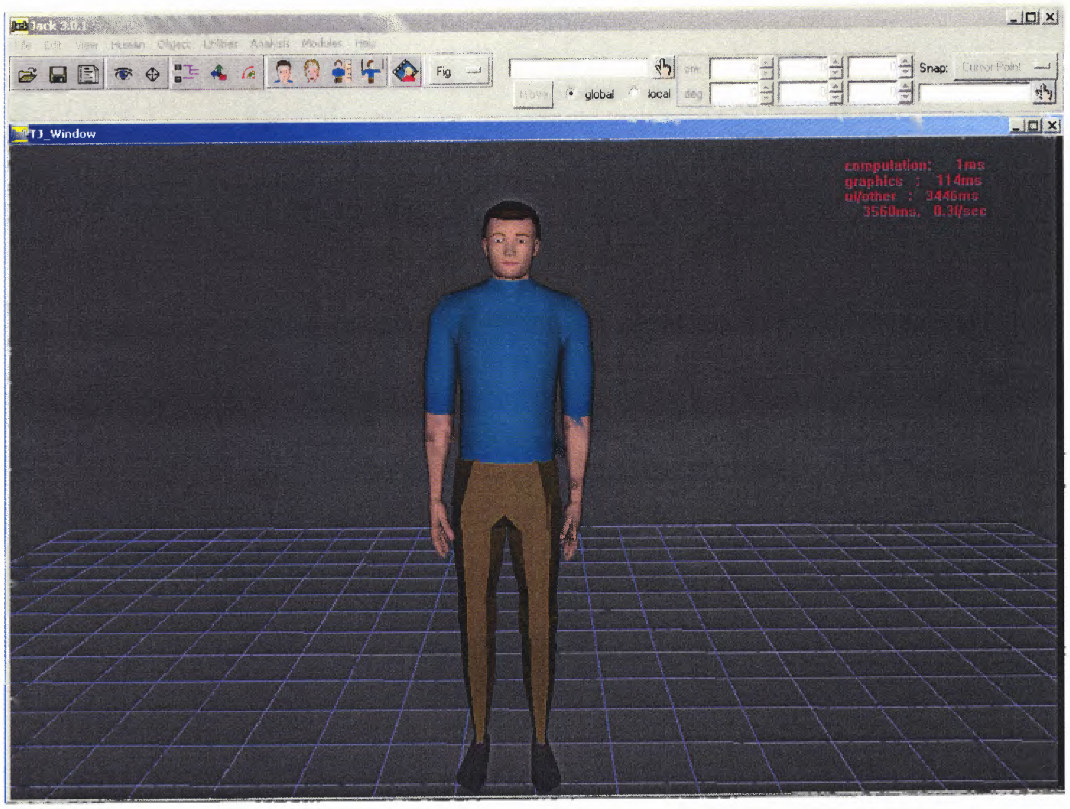


Figure 3.1 Jack interface with human figure.

Jack allows its users to manipulate individual body segments connected by joints that obey angle limits derived from NASA studies. As an end-effector of a body segment on a Jack virtual human is moved, the software uses real-time inverse kinematics to determine the joint angles that move the linked segments into anatomically reasonable positions [22]. For instance, when a figure's hand is moved, the upper and lower arm segments and related joints move in a way that is expected of a human body. This provides realism in Jack's motions.

Furthermore, Jack provides a built-in motion system for defining tasks that must be performed under time constraints. Jack simulations may consist of several distinct motions, many occurring simultaneously and defined for a specified interval of time. The user can create motions interactively or using a scripting language in Jack to control the movement of the head, eyes, torso, pelvis, center of mass, arms, hands, feet, and more. In addition, it is possible to move objects, as well as the camera perspective.

Conveniently, there are a number of ways in Jack to save, store and reuse postures of the whole human figures as well as postures of parts of human figures such as handshapes. This turns out to be very helpful when synthesizing sign language, since a library of handshapes can be created and stored prior to use in generating a sign.

In addition to modeling humans, Jack can also create such primitives as cubes, sphere, cylinders, cones and toroids. Constraints can then be defined between theses objects and human figures, if desired. For example, constraints can be created between Jack's arms and cubes so that when the arm moves these cubes move as well.

Most importantly the Jack software is extensible. It is accessed through scripting languages such as Python, Tcl/Tk and Lisp to enable users to extend its functionality. These scripting APIs (Application Programming Interface) allow the general programming of Jack internal functions, and each have consoles easily accessible from the Jack menu. In fact, this access is supported by a specific high level scripting interface for Jack based on Python, called JackScript. JackScript can be accessed from the Python menu, where is can be used to provide support for procedural animations as well as many other utilities.

3.2.2 Python

Python is an open source language whose popularity is increasing. It combines ease of use with the capability to run on multiple platforms. Guido van Rossum created the language nearly 11 years ago and it has evolved into a powerful programming language [23].

Python is an interpreted language that employs an object-oriented approach. It is a high-level programming language, which separates the user from the underlying operating system as much as possible, although it still provides the ability to access the operating system at a lower level if desired. Because of this ability, Python is often classified somewhere between such languages as Visual Basic or Perl and the system-level C language [24].

Of the three programming languages capable of extending the functionality of Jack, Python is the most suitable for this sign synthesis application. The key reason is that JackScript, the scripting language for the Jack toolkit, is itself written in Python, and is easily used within Python modules.

In addition, numerous other features support the use of Python. Its capabilities, portability, extensibility, extensibility, and superior text processing, as well as its ease to learn and code make it the language of choice for this application.

There is very little that cannot be done with Python. The core of the language is very small, but it provides enough of the basic building blocks to allow the design of most applications. Furthermore, because the language can be extended using C, C++, and even Java in certain circumstances, it does not limit program development. The Python interpreter actually comes with a large library of modules that extend the capabilities of the language to allow network communication, text processing, and regular expression matching [24].

Python is able the run on a range of operating systems. Its design is not attached to a specific operating system because it is written in portable ANSI C [23]. This means that a Python program written on one operating system can be run, tested and/or uploaded on different systems. This would allow the created sign synthesis application to be portable and run on any machine supported by Jack. Python and Jack are well integrated with both UNIX and Windows platforms.

Because Python is written in C and because there is access to the source code, it is easy to write extensions to the language. Python can be embedded into C or C++ applications so it is possible to provide a scripting interface to any desired application using the Python language. Due to the support for cross-language development, Python can be used to design and conceptualize an application and later port it to C. There is no need to rewrite the application in C before using it. Python and C can work in tandem [24]. Extensibility of the application at hand was an important consideration. There will be many improvements made in the future to the sign synthesis program and it was important to use a language that would facilitate any necessary improvements, no matter how unexpected.

In Python, everything (numbers, strings, lists, hashes, file-objects, and compiled code) is an object, and can be passed easily to functions. Python is particularly known to be an excellent string handler; it is very similar to Perl in this regard. It comes with expression libraries that allow use of the same expressions as Emacs, Perl, and many other utilities. With Python's support for other text-processing engines (regular expressions and the natural splitting/combing of information) and flexible variable and object handling, Python becomes a very useful tool for the text-processing programmer [24]. This particular attribute of Python is pertinent to use in the sign synthesis program, since input to the program is textual and requires processing prior to its conversion to visual information.

Python possesses suitability for large tasks. The language has features that support the complexity as programs grow in size. This is extremely important for a program that intends to eventually support a sign dictionary with over 50,000 signs. Tcl, for example, is specifically intended for use in relatively small programs, though some have pushed the envelope [25].

One of the most attractive aspects of the language is that it employs a compilation stage that translates the raw-text Python script into a series of bytecodes, which are then execute by the Python Virtual Machine. The use of the compilation and bytecode stages helps to improve performance and makes Python much faster than pure interpreters such as BASIC, but slower than the truly complied languages such as C and Pascal. However, unlike many other languages the bytecode versions of modules can be saved and executed without having to recompile them each time they are required, thereby improving performance by eliminating the compilation stage [23].

Best of all Python is easy to learn. This is mainly because its source code resembles pseudo code. Once the basic principles of the language are understood, learning the rest is relatively easy. The language is also fast to code, since it is a high level language and the programmer can skip many lower level tasks that other languages require. In addition, the interactive interpreter that comes with the Python distribution brings rapid development strategies to any project. Each line of code can be run prior to putting them all together. This enables the programmer to edit code with the execution environment.

3.3 The Kinematic Parameters of Sign Language

When synthesizing signs of any sign language, thought has to be given to: (1) gesture specification and how the signs will be recreated, (2) the input and output language, and any linguistic conversion that might have to take place, and (3) the type of sign generation engine needed, and the mechanisms needed for its functionality.

Of these three, the one of immediate importance is the ability to create realistic sign representation. This requires knowledge of what distinguishes each sign from all other signs in the language, and is a topic that has been the focus of research by many linguists over the past four decades.

Currently, most researchers agree that five parameters completely describe a sign, and differentiate it from all others signs. They are:

(1) Handshape	- the configuration of the hand defined by the way the fingers are arranged;
(2) Hand Orientation	- the direction the palm of the hand faces;
(3) Location	- the place in space or on the body where the sign is made;
(4) Movement	- the way the hand or hands move;
(5) Non-Manual Gestures	- these include facial expressions, and movements of the shoulders, head, and body [26].

There are several notation systems, which attempt to provide a way to write signs. These notation systems use methods of gesture specification that rely on the parameters of signs. One of the more popular systems is known as Stokoe Notation.

3.3.1 Stokoe Notation

Stokoe notation is an American Sign Language notation system that was devised by linguist Willam Stokoe in 1960. Stokoe's notation system was the first of its kind, and as a result grew a following of many researchers. It is actually considered a family of similar linguistic notations, since various groups of researchers have made improvements to the system and no standard version was developed.

Stokoe believed that each sign in American Sign Language had three defining parameters: the place where it was made, the handshape accompanying it, and the action of the hand or hands [27]. These he named the *tab* (tabulation, location), *dez* (designator, handshape), and *sig* (signation, movement), respectively. In his notation he described Orientation as a part of handshape and in some cases, as a part of movement. Over the years, researchers have realized the necessity of treating Orientation as its own independent parameter, and as a result this view has become standard.

Stokoe purposely omitted Non-Manual Gestures in sign descriptions. He recognized their importance to sign formation, especially as supra segmentals, but also realized the extreme difficulties in analyzing them. This he left to be studied by subsequent researchers.

In his book, A Dictionary of American Sign Language, Stokoe specifies a limited number of primes for each of the three parameters; handshape, location, and movement. Symbols were assigned to represent each of these primes, which include: 19 different handshapes, 12 different locations, and 24 alternate movements. Stokoe uses these symbols to define over 2,000 different signs. A table of these symbols can be seen below.

Table 3.1 Stokoe's Transcription Symbols [27]

		Tab symbols	
1.	Ø	zero, the neutral place where the hands move, in contrast with all places below	
2.	○	face or whole head	
3.	⌒	forehead or brow, upper face	
4.	△	mid-face, the eye and nose region	
5.	⌣	chin, lower face	
6.	ʒ	cheek, temple, ear, side-face	
7.	Π	neck	
8.	[]	trunk, body from shoulders to hips	
9.	\	upper arm	
10.	√	elbow, forearm	
11.	a	wrist, arm in supinated position (on its back)	
12.	D	wrist, arm in pronated position (face down)	
		Dez symbols, some also used as tab	
13.	A	compact hand, fist; may be like 'a', 's', or 't' of manual alphabet	
14.	B	flat hand	
15.	5	spread hand; fingers and thumb spread like '5' of manual numeration	
16.	C	curved hand; may be like 'c' or more open	
17.	E	contracted hand; like 'e' or more claw-like	
18.	F	"three-ring" hand; from spread hand, thumb and index finger touch or cross	
19.	G	index hand; like 'g' or sometimes like 'd'; index finger points from fist	
20.	H	index and second finger, side by side, extended	
21.	I	"pinkie" hand; little finger extended from compact hand	
22.	K	like G except that thumb touches middle phalanx of second finger; like 'k' and 'p' of manual alphabet	
23.	L	angle hand; thumb, index finger in right angle, other fingers usually bent into palm	
24.	3	"cock" hand; thumb and first two fingers spread, like '3' of manual numeration	
25.	O	tapered hand; fingers curved and squeezed together over thumb; may be like 'o' of manual alphabet	
26.	R	"warding off" hand; second finger crossed over index finger, like 'r' of manual alphabet	
27.	V	"victory" hand; index and second fingers extended and spread apart	
28.	W	three-finger hand; thumb and little finger touch, others extended spread	
29.	X	hook hand; index finger bent in hook from fist, thumb tip may touch fingertip	
30.	Y	"horns" hand; thumb and little finger spread out extended from fist; or index finger and little finger extended, parallel	
31.	Ȣ	(allocheric variant of Y); second finger bent in from spread hand, thumb may touch fingertip	
		Sig symbols	
32.	^	upward movement	vertical action
33.	v	downward movement	vertical action
34.	N	up-and-down movement	vertical action
35.	>	rightward movement	sideways action
36.	<	leftward movement	sideways action
37.	z	side to side movement	sideways action
38.	⊤	movement toward signer	horizontal action
39.	⊥	movement away from signer	horizontal action
40.	I	to-and-fro movement	horizontal action
41.	a	supinating rotation (palm up)	rotary action
42.	D	pronating rotation (palm down)	rotary action
43.	ω	twisting movement	rotary action
44.	ŋ	nodding or bending action	
45.	□	opening action (final dez configuration shown in brackets)	
46.	#	closing action (final dez configuration shown in brackets)	
47.	ɤ	wiggling action of fingers	
48.	@	circular action	
49.	)(	convergent action, approach	interaction
50.	×	contactual action, touch	interaction
51.	⌶	linking action, grasp	interaction
52.	+	crossing action	interaction
53.	⊙	entering action	interaction
54.	÷	divergent action, separate	interaction
55.	"	interchanging action	interaction

Stokoe specified 19 different values for the handshape or *dez* parameter (Figure 3.2), although there are more handshapes used in ASL. Some of the handshapes he specified are not necessarily identical in configuration to the letters in the manual alphabet with the same name, but closely resemble them in formation [8]. For instance, the handshape named 'B' in Stokoe's notation is really the flat hand configuration, and lacks the bent thumb formation of the actual manual alphabet handshape. Also, the fist configuration though referred to as 'A', is interchangeable with the handshapes for the manual letters 's' and 't', since they each generally resemble a closed fist.

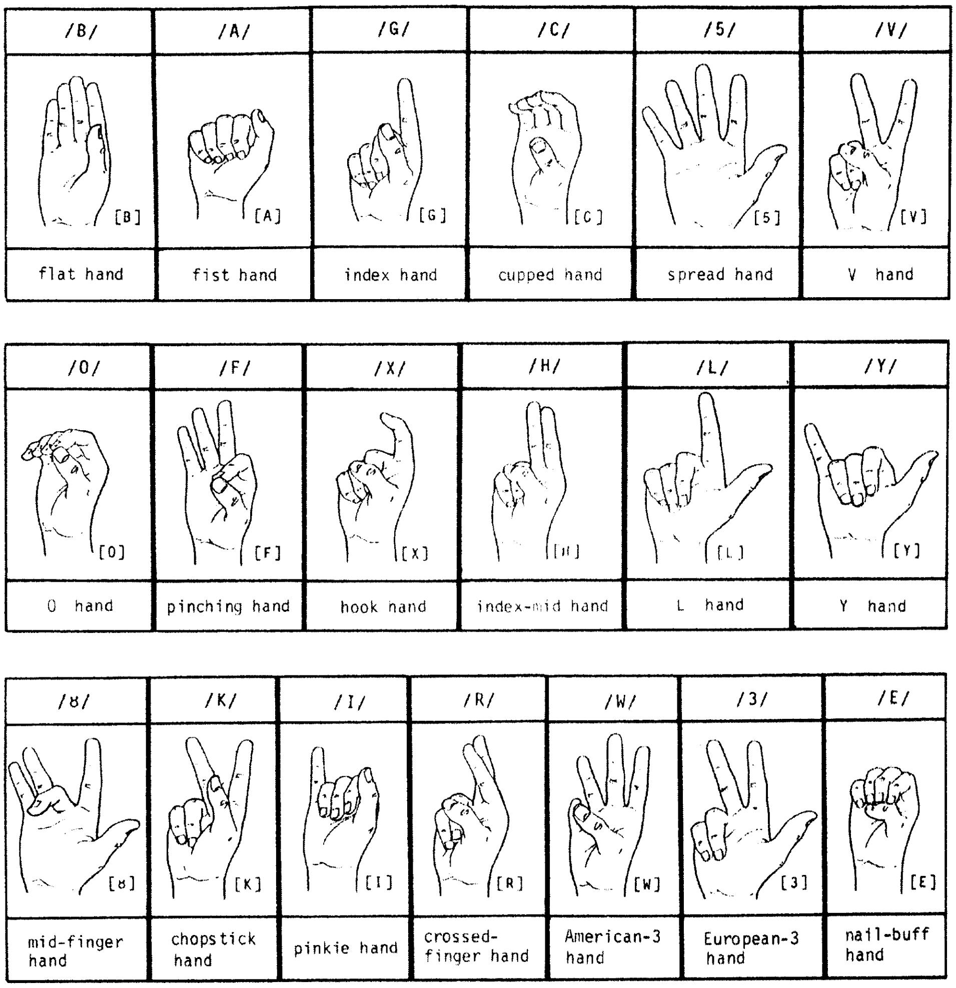


Figure 3.2 Stokoe Handshapes arrange in order of frequency of use [8].

The space in which signs are made is a delimited region referred to as the signing space. It is described as extending from the top of the head to just below the waist, and spanning the reach of arms bent at the elbows from side to side. A prime of the place of articulation of a sign is always defined by a specific location on or around the body within the signing space. Stokoe described 12 different locations that he thought would minimally define the place of articulation of signs (see Figure 3.3). He also determined that in addition to these locations, several of the *des* handshapes would serve as place of articulation for signs that involved two hands touching [1].

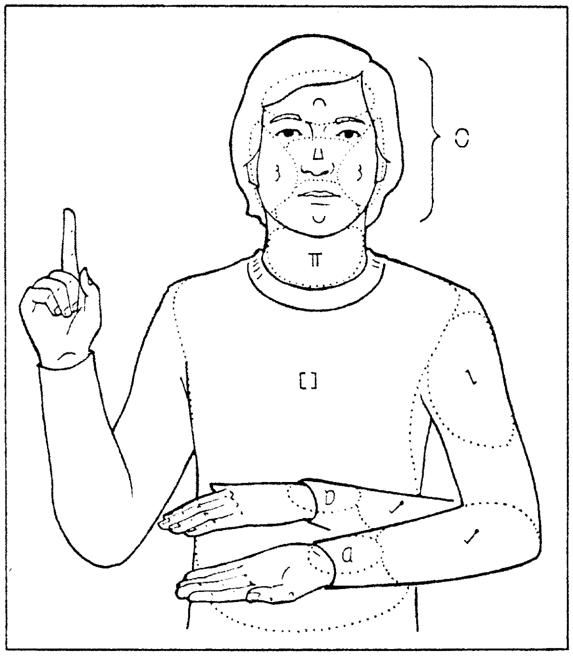


Figure 3.3 Places of articulation according to Stokoe notation [8].

The most complex of the three Stokoe parameters is of movement. It can be exceedingly difficult to analyze and transcribe. This complexity, arises due to the seemingly infinite number of way the arms and hands can be articulated. Luckily, just as the formational system of American Sign Language limits the number of handshapes that it uses, so does it limit the different types of movements. These movements may occur in isolation, in series with one another, or concurrently, depending on the sign [8]. Stokoe proposed 24 different types of movement (some shown in Figures 3.4 and 3.5), which he thought could be used in describing most ASL signs.

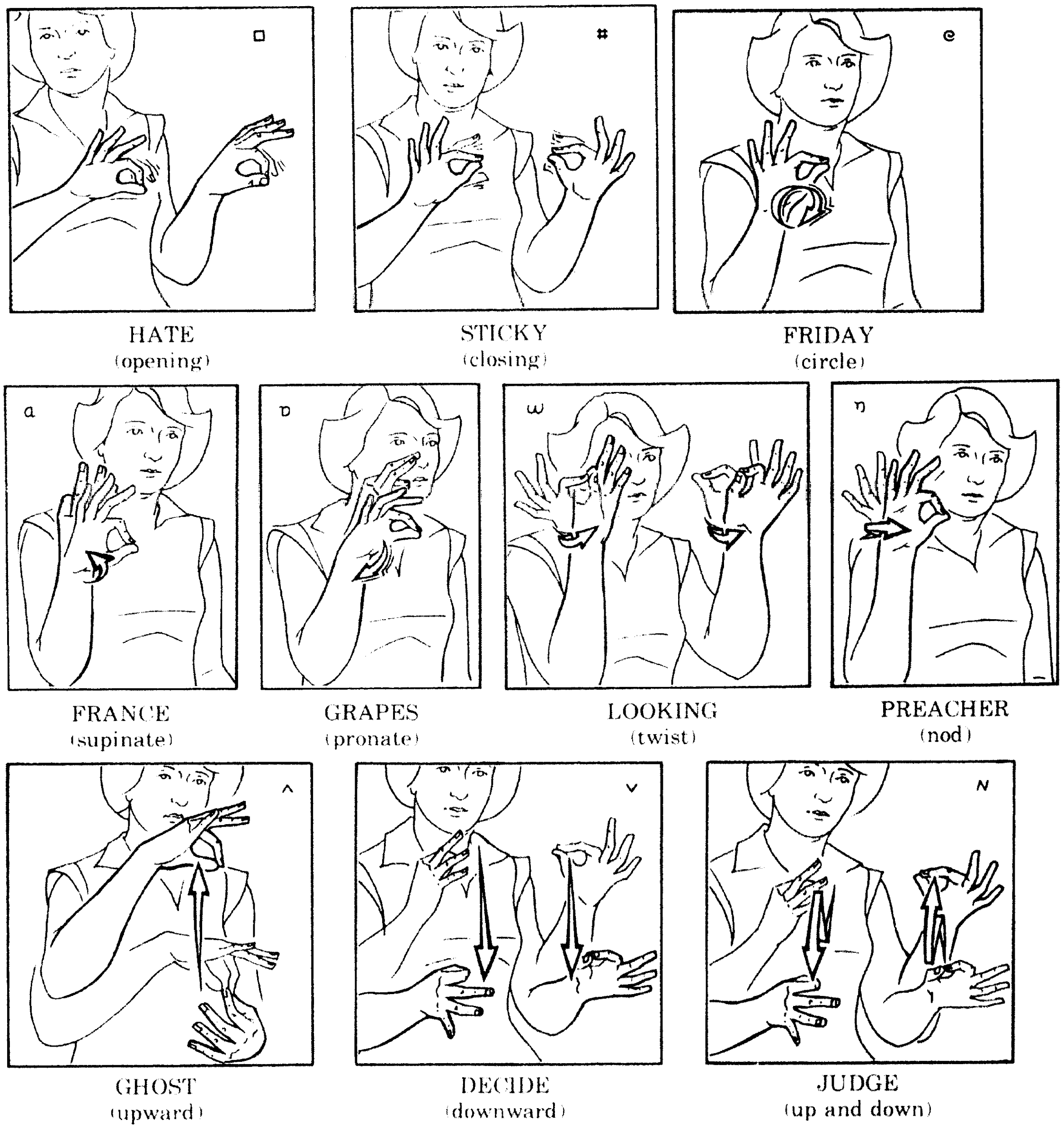


Figure 3.4 Some Stokoe movements as related to their *sig* symbols [8].

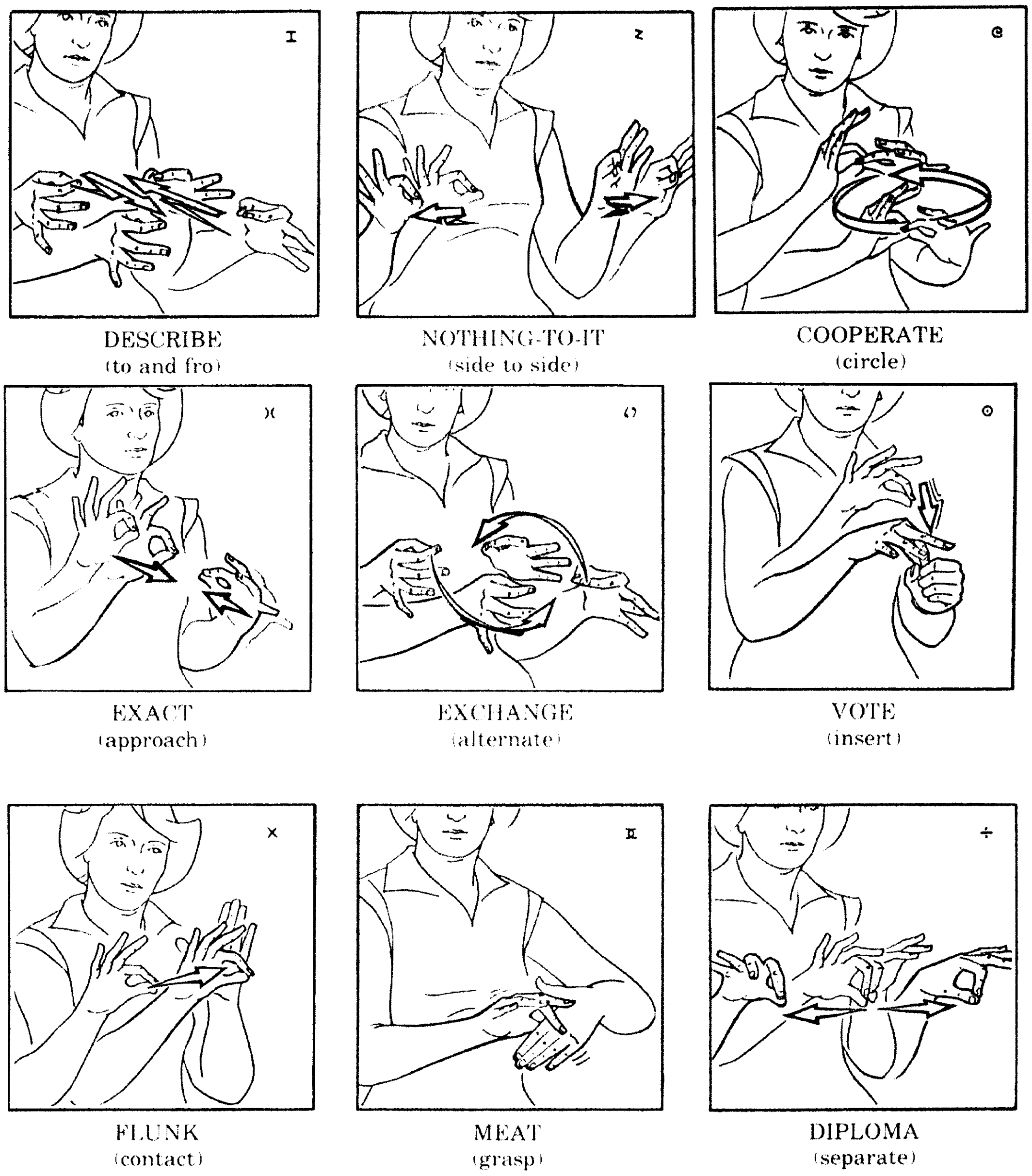


Figure 3.5 More Stokoe movements as related to their *sig* symbols [8].

Once the three parameters of a sign have been identified, the sign can then be transcribed and described using conventions which Stokoe provided. Stokoe proposed that each sign be described using a TD^s notation, where ‘T’ denotes the *tab* (location), ‘D’ the *dez* (handshape), and ‘s’, the *sig* or movement in the sign. This convention indicates that at some location ‘T’ the handshape ‘D’ is formed and is used to perform the action ‘s’.

There are however, additional symbols and conventions used along with this notation. For instance, the sign for the word 'me' is transcribed as [ ]$G_{\top}^{\times}$ (Figure 3.6). The *sig* symbol, $_{\top}$, written as a subscript to the *dez* symbol, G, shows the way that the hand is held. According to this notation the sign for ME is described as having the dominant hand in the 'index hand' configuration (see Table 3.1), pointing and moving towards the trunk of the body, and then touching it. Here, Stokoe uses the *sig* as a subscript to allude to the orientation of the hand. This can be better seen with the sign for MONEY, which is transcribed as $B_{\alpha}B_{\alpha}^{\times \cdot}$ (Figure 3.7). The subscript 'α' of both the *tab* and the *dez* shows that they are supinated, turned palm up. In this sign a *dez* symbol is used as a *tab* to indicate that the sign takes place at the location of the non-dominant, which is in the 'flat hand' configuration. A sharp or tense movement is made as the hands touch, as indicated by the dot following the '×'.

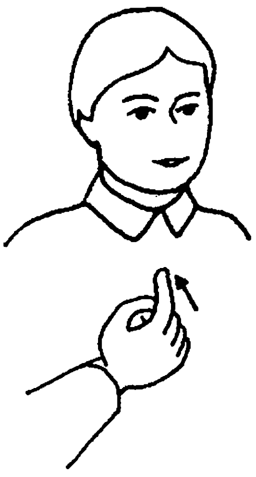


Figure 3.6 The sign for ME transcribed as [ ]$G_{\top}^{\times}$ [28].

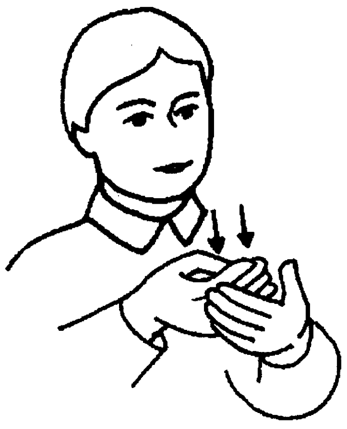


Figure 3.7 The sign for MONEY transcribed as $B_{\alpha}B_{\alpha}^{\times \cdot}$ [28].

3.3.2 Sign Specification in the Translator System

As discussed in previous chapters, the intention of this research is to propose a method of synthesizing signs from their basic kinematic parameters. In order to accomplish this, a technique had to be developed that codes each sign in sufficient detail to allow graphical rendering by the synthesis engine. Such a technique was derived from Stokoe's seminal investigation of sign formation.

This kinematic coding system defines the simultaneous occurrence of four parameters: handshape, location, movement, and orientation. This parametric breakdown of a sign is similar in nature to that of Stokoe, with the addition of orientation, which is treated as an independent parameter. The orientation of the hands is equally significant to the other three parameters when describing and distinguishing between signs. This is evident in ASL where many signs are similar to one another in all aspects except for the orientation of the hand or hands. The signs for CHILD and THING, for instance, differ only in orientation (see Figure 3.8). In the sign for CHILD the palm of the dominant hand faces up (palm up), where as in the sign for THING the palm of the hand faces the downward direction (palm down).

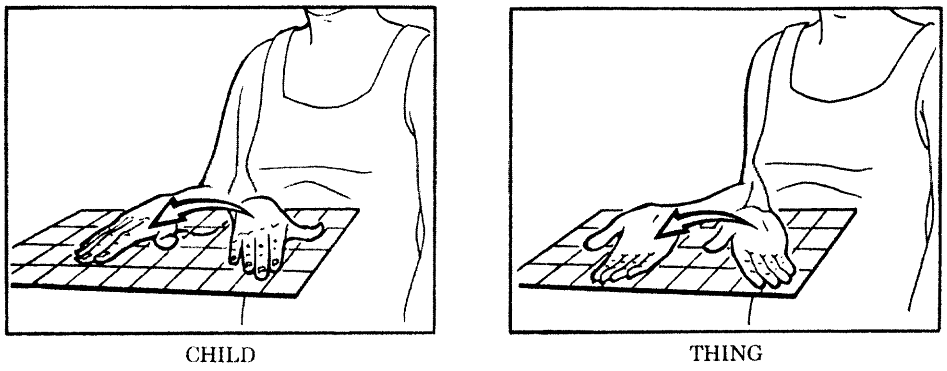


Figure 3.8 Importance of orientation as a parameter in sign language [8].

At present, as in Stokoe's work, non-manual signals have been omitted from use in the translator system. Though these are important in the formation of a sign, they are beyond the scope of this effort and will be left for others to study. In the interim, a system that synthesizes signs without producing facial expressions will still convey meaningful information.

The key to efficient sign synthesis based on parametric values, is a limitation on the number of possible entries or primes, while effectively generating the majority, if not all, of the signs in the language. The primes chosen for this system include 39 handshapes. These are stored in a handshape library, and can be used in the specification of signs, letters of the manual alphabet, and the numbers one through ten. These handshapes are the 24 used to represent the letters of the alphabet, the 10 used to represent numbers one through ten, and the 5 remaining most frequently used handshapes in ASL and Signed English (see Figures 3.9 through 3.11). Only 24 handshapes are used to represent the manual alphabet, and thus used in fingerspelling, since the letter 'J' uses the 'I' handshape in its formation, and the letter 'Z' uses the '1' handshape.

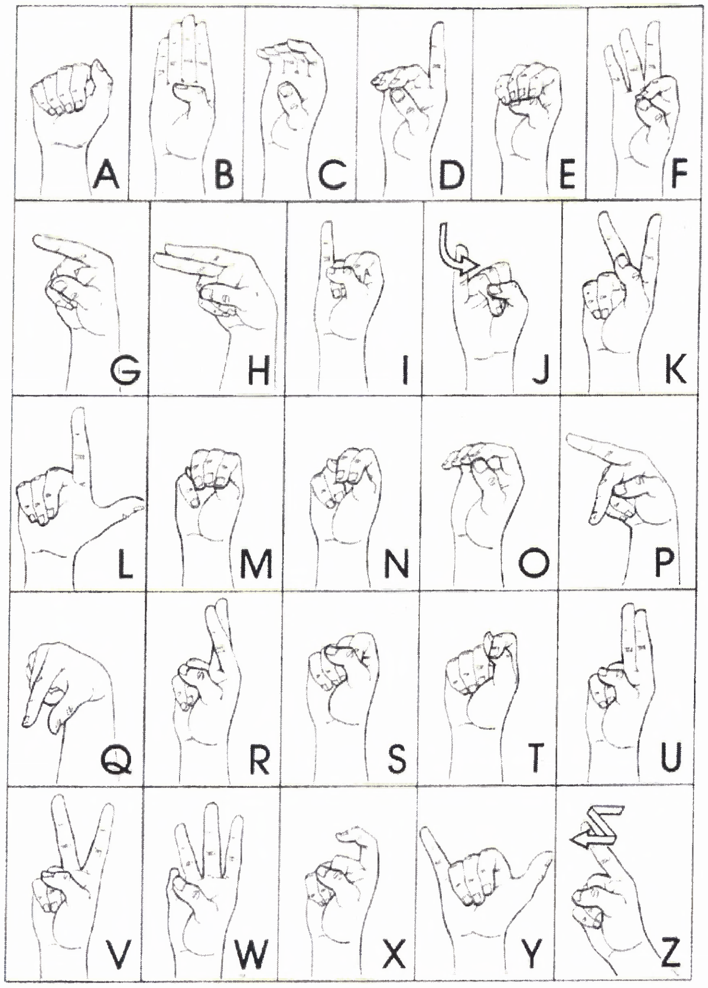


Figure 3.9 Handshapes used in the manual alphabet [29].

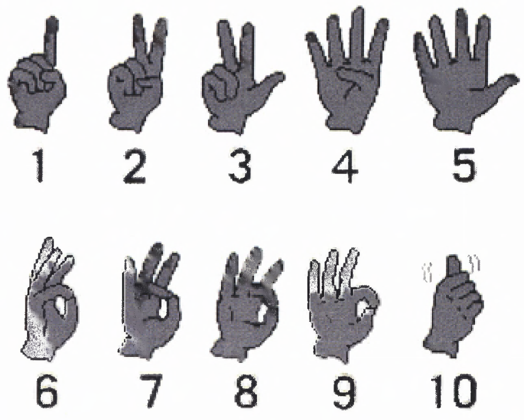


Figure 3.10 Hanshapes used in numbers one through ten [30].

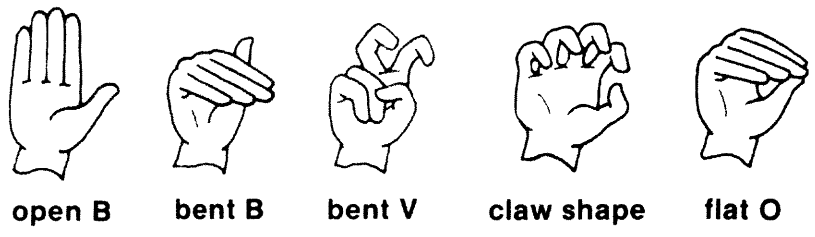


Figure 3.11 Additional frequently used handshapes [31].

The increase in the number of handshapes from the 19 proposed by Stokoe is justified by recent changes in ASL, as well as the desire for well-formed-ness in certain ASL signs. Initialization of signs has been introduced to American Sign Language since the completion Stokoe's work. This technique has resulted in the creation of several signs from one, by increasing the importance of handshapes used in some signs. For instance, where a single sign made with the 'A' handshape would initially represent all three words TRY, STRIVE, and ATTEMPT, a distinction is now made between the three by choosing the handshape 'T', 'S' or 'A', respectively [1]. This has added all 24 fingerspelling handshapes to those used in ASL.

The 19 handshapes proposed by Stokoe are considered the 19 major hand configuration sub-primes in ASL. There are actually additional sub-primes within the different hand configuration classes [8]. For instance, the 'B' handshape in Stokoe's notation is only one of the many different 'B' sub-primes in the 'B' hand configuration class. There is also the 'open B' configuration, and the 'closed B' configuration to name a few (see Figure 3.11). By increasing the number of sub-primes used in the translator system, more accurate well-formed signs can be generated than would be possible with with Stokoe's 19.

The places of articulation (location) associated with signs are defined relative to the torso of the avatar. In order to create a naturally appearing sign, the Jack animation engine must be instructed to move the appropriate end effector (eg. tip of index finger of the dominant hand) to a location, while also determining anatomically appropriate joint angles of the finger, wrist, elbow and shoulder joints. This requires the application of inverse kinematics, which allows the joint angles of the limb to be determined from the desired end effector location.

Unfortunately, Jack's internal inverse kinematics algorithms are not accessible in JackScript and hence are unavailable to a Python program. This difficulty was overcome through the introduction of virtual cubes as a convenient means of directing Jack to correctly position an end-effector at a desired location, with appropriate arm joint trajectories. This method also provides a convenient mechanism to control the orientation parameter associated with the hand.

This technique utilizes Jack's innate ability to align one cube's position and orientation with that of another cube. An invisible cube is placed at a target location with a desired orientation (location cube), and a second cube is constrained to the desired end-effector of the arm (end-effector cube). Thus, when the end-effector cube is moved, the constraint requires that the end-effector and all attached segments (i.e. hand, forearm, upper arm) change to accommodate the movement. When the JackScript command is given to the end-effector cube to align with the location cube, Jack performs real-time inverse kinematic calculations to determine the joint angles of the arm necessary to comply with the constraints of the human-model, and generate human-like movement to the location. The local co-ordinate systems of the cubes are aligned, allowing the orientation of the location cube to play an important role in the orientation of

The local co-ordinate systems of the cubes are aligned, allowing the orientation of the hand at the desired location (see Figure 3.12). For instance, in the figure below, two visible location cubes are placed on the nose, and an end-effector cube is constrained to the fingertip. One location cube is oriented at a forty-five degree angle to the bridge of the nose, while the other is oriented vertically, parallel to the nose bridge. In Figure 3.12 A and B the end-effector cube is instructed to move to the first location. In A, the finger, and hence the hand, approaches the location with a correct orientation. In C and D the finger is moved to the second location cube, which results in both a different approach trajectory and final orientation.

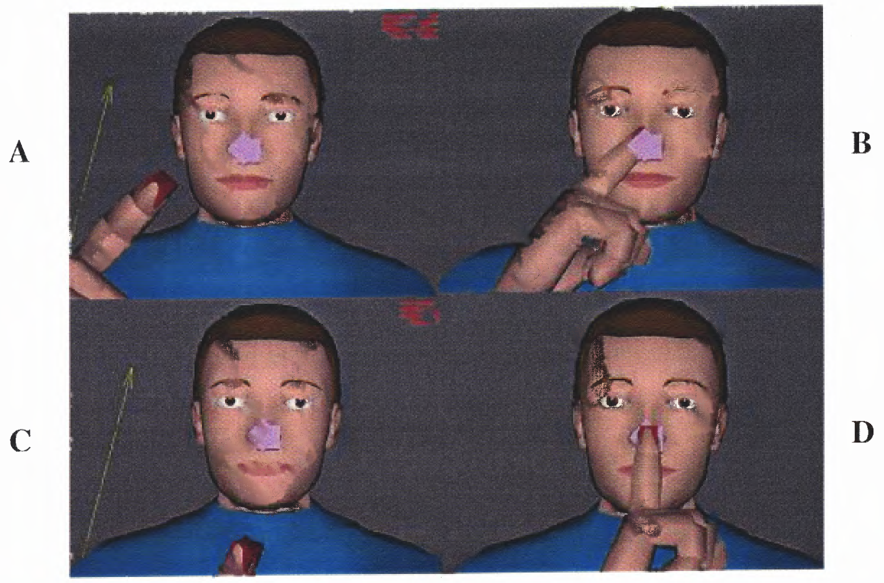


Figure 3.12 Virtual cubes used to specify location.

It should be noted that this method is very powerful. It not only provides the location parameter, but also partially defines the movement and orientation. Figure 3.12 shows that the same location can be specified by different cubes, which have different orientations. These orientations determine the final alignment of the end-effector cube, which controls the final orientation (3 degrees of freedom) of the hand. Jack's inverse kinematics computation also determines the trajectory followed by the end-effector cube and makes the movement trajectory of the hand appear realistic.

The execution of a movement is accomplished with the JackScript command, 'Reach', which instructs the human-model to move a hand or hands to a particular location. This command is normally very limited in its functionality, requiring the end effector to be either the palm of the hand or the model's forearm. Additionally, Jack is incapable of reaching to any of the predefined sites on its body. The introduction of virtual cubes expands the capabilities of the 'Reach' function. 'Reach' allows the specification of an object that is constrained to the hand to be the defacto end-effector. 'Reach' also allows the target location to be an object as opposed to an anatomical site. The default setting for 'Reach' aligns the local coordinate system of the end-effector object and the target object.

3.3.2.1 Placing Virtual Cubes. As an example, the sign for YOU uses the '1' handshape with the index fingertip as the end-effector. The location of the end-effector is specified as a position directly in front of the signer's chest. The orientation of this sign requires the finger to be pointed horizontally, away from the signer. Since the index finger of the dominant hand is often used as an end-effector, an end-effector cube is constrained to its tip. The orientation of that cube is defined with its:

- y-axis parallel to the palm of the hand and positive in the direction of the thumb
- x-axis normal to the dorsal side of the finger and positive on the dorsal side
- z-axis co-linear to the central axis of the distal phalange and positive in the distal direction

The target cube for the sign YOU is placed approximately .3 m directly in front of the avatar's chest. Its orientation is accomplished by setting its local reference frame parallel to that of the site located at the center of Jack's chest. This determines that the target cube axes are aligned with its:

- y-axis oriented with its positive direction up relative to Jack
- x-axis oriented with its positive direction to Jack's right
- z-axis oriented with its positive direction in Jack's direction of view

Executing the JackScript "Reach" function aligns the end-effector cube's local reference frame with that of the target cube. This brings the finger tip to the desired location, with its distal end aimed in the direction of Jack's view, and with the dorsal side of the finger (and hand) oriented toward Jack's right.

The ASL sign for ME uses the same handshape, with the fingertip touching the middle of the signer's chest, pointing toward the signer. Thus, it differs from YOU in both the place of articulation (location) and partial orientation of the hand.

Accomplishing the display of this sign uses the same end-effector cube as YOU, and a different target cube placed at the center of Jack's chest. Its local orientation is first made parallel to the orientation of Jack's waist, and then rotated 180 degrees about its y-axis. Thus, when the "Reach" function is executed, the axes of the end-effector cube are aligned with the new target cube and the finger points toward Jack's chest, while its dorsal side is now oriented to Jack's left.

Rotating the target cube about only its y-axis changes only the direction in which the finger and hand point, while the thumb continues to point upward relative to Jack.

The structure and possible parameters of the 'Reach' command, and other pertinent JackScript functions can be viewed in Appendix C.

3.3.2.2 Virtual Cube Sites. Once a method of specifying cube locations was defined, the next step was to decide upon the sites for the location cubes, and end-effector cubes. For this, Stokoe's model of places of articulation (Figure 3.3) was modified to provide specific sites of articulation (location sites shown in Figure 3.13). In this prototype implementation, multiple cubes, with different orientations, are superimposed at location sites. Depending upon the desired hand orientation, the synthesis engine passes the correct cube to the 'Reach' function. Cubes were attached to three of the most frequent end-effector sites used in American Sign Language. Figure 3.14 shows cubes placed at the tips of the thumb, index finger and middle finger, as well as the center of the palm. Not shown is the fifth cube located on the dorsal side of the palm. All cubes placed in the signing environment are made invisible so as not to distract the viewer.

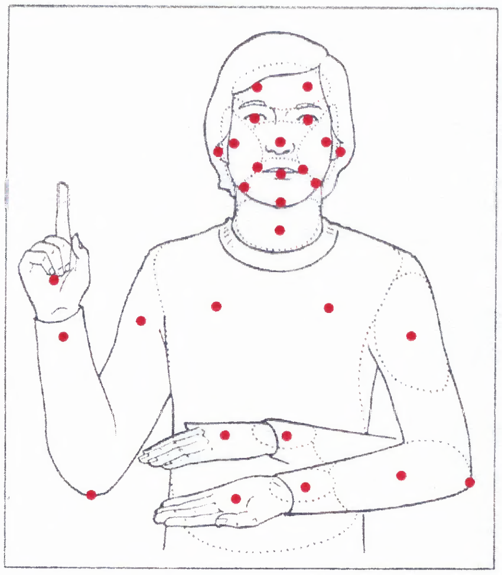


Figure 3.13 Sites of location cubes in the translator system.

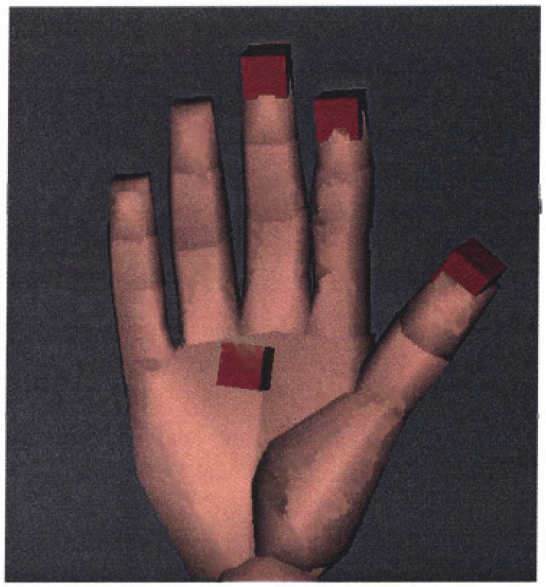


Figure 3.14 End-effector cubes on the right hand of the human-model.

3.3.2.3 Wrist Orientation. Although the virtual cubes specify the orientation of the end-effector they do not necessarily define the correct wrist orientation needed to produce a correct sign representation. As a result, seven primes were created to represent the possible wrist orientation in a sign. Each prime is defined by three angle values, representing the three degrees of freedom of the wrist in the human-model. For example, the palm-up orientation of the hand would be defined by the following three lines of code (where 'rw' specifies the right wrist, and 'wor3'indicates wrist orientation number 3):

```
rw = jack.right_wrist
wor3 =(0,0,90*u.deg)
rw.Move(wor3)
```

3.3.2.4 Synthesis of Signs. In order to synthesize a sign in the Sign Translator System, it is first necessary to create a detailed kinematic description of each sign. This description must provide information regarding the specific parameter primes needed to produce the sign representation, and the instances at which they occur. As a result, a complete description requires, the beginning, ending, and any intermediate configurations of the sign. Depending on the complexity of the sign, multiple intermediate configurations may be required.

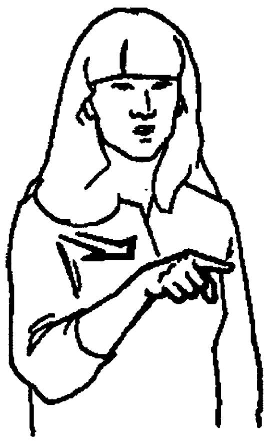


Figure 3.15 ASL sign representing YOU [32].

Returning to the sign YOU for instance, requires a signer to form the one handshape with the dominant hand, in front of the body, and point the finger straight out [32] (Figure 3.15). While this description and an illustration, may be enough to instruct a person to recreate the sign, more information is needed to synthesize the sign using a computer driven avatar. As a result, a descriptive format was customized to effectively parse a sign definition into parameters the Sign Translator System could synthesize and combine (Table 3.1). By describing a sign in this way, coding can be developed in Python to mimic the structure of this description, and hence generate a sign representation that is intelligible.

As can be seen in the sign specification table, the sign YOU, does not have a particular beginning configuration (handshape, orientation, location and movement). Its initial configuration is inherited from the previous sign, and its ending configuration, has the dominant arm out in front of the body with the '1' handshape, while oriented with the palm of the hand facing the signer's left (assuming the dominant hand is the right hand).

Table 3.1 Sign Specification prior to entry in the Sign Translator System

YOU

Dominant hand	Right hand	Dominant hand	Left hand
Beginning Handshape	Inherited	Beginning Handshape	
Beginning Orientation	Inherited	Beginning Orientation	
Beginning Location	Inherited	Beginning Location	
Beginning Movement	Inherited	Beginning Movement	
Duration		Duration	
Intermediate Handshape		Intermediate Handshape	
Intermediate Orientation		Intermediate Orientation	
Intermediate Location		Intermediate Location	
Intermediate Movement		Intermediate Movement	
Duration		Duration	
Ending Handshape	1	Ending Handshape	
Ending Orientation	Palm left	Ending Orientation	
Ending Location	In front of the body	Ending Location	
Ending Movement	None	Ending Movement	
Duration		Duration	

This sign specification can be imitated using a combination of JackScript functions. The functions will instruct the human-model to form the '1' handshape on the right hand, and to orient the palm of the hand vertically to face the avatar's left side. Simultaneously, the avatar is instructed to align the position and orientation of the end-effector cube on the index finger, to that of the target location cube in front of the avatars body.

The JackScript algorithm for signing YOU can be seen below:

```
DoTogether(jack.Reach('right',point,endeff = rindex,duration
= 2),HandShapeMotion(jack,letters['1'],'right',duration
=1.5),rw.Move(wor1,duration = 1.5))
```

The highest-level function, `DoTogether`, assures that all functions within it are executed simultaneously. The avatar is instructed to align the end-effector cube, labeled `rindex`, to the position and orientation of the location cube `point`. At the same time, the joint angle description of the handshape `1` is loaded from the handshape library file, and the wrist, `rw` (right wrist), is instructed to assume the orientation allocated to the variable `wor1`.

CHAPTER 4

FEATURES OF THE SIGN TRANSLATOR SYSTEM

4.1 Input and Output Environment

The input and output environments of the Sign Translator System are based on the graphical user interface of the Jack software. Once launched, Jack provides two windows; a graphics window, where objects are rendered, and a control bar that contains the standard menus and icons of the program (Figure 4.1).

An additional window, the JackScript console, can be accessed by selecting the Python icon after adding it to the control bar through View >Toolbars>Consoles (Figure 4.2). This console is where the supervisory Python program, for sign translation is accessed. This program is initiated by typing 'execfile' (a Python command for executing a file), followed by the location and the name of the file, as follows:

```
execfile('/Python23/prgmodule/SignJack.py')
```

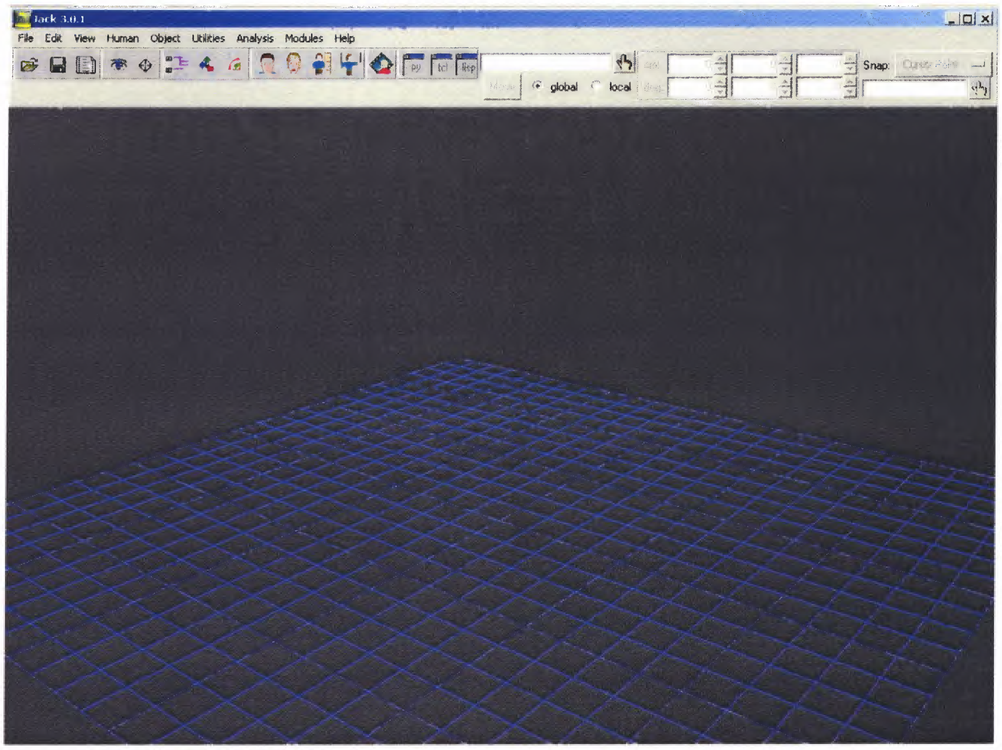
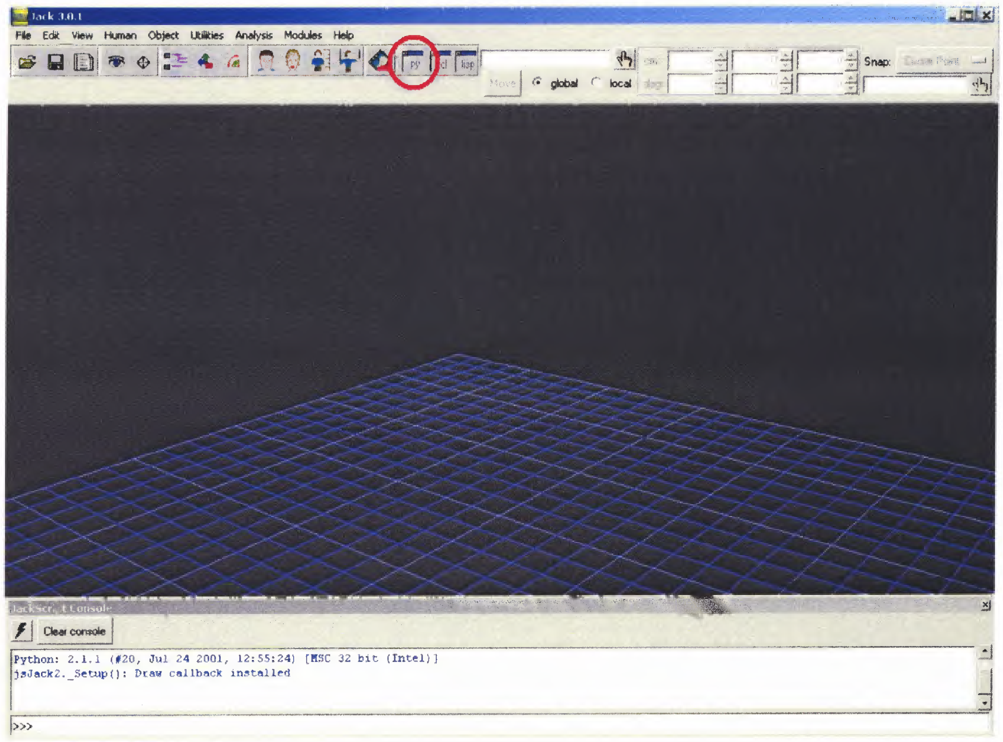


Figure 4.1 Jack user interface showing control bar and graphics window.

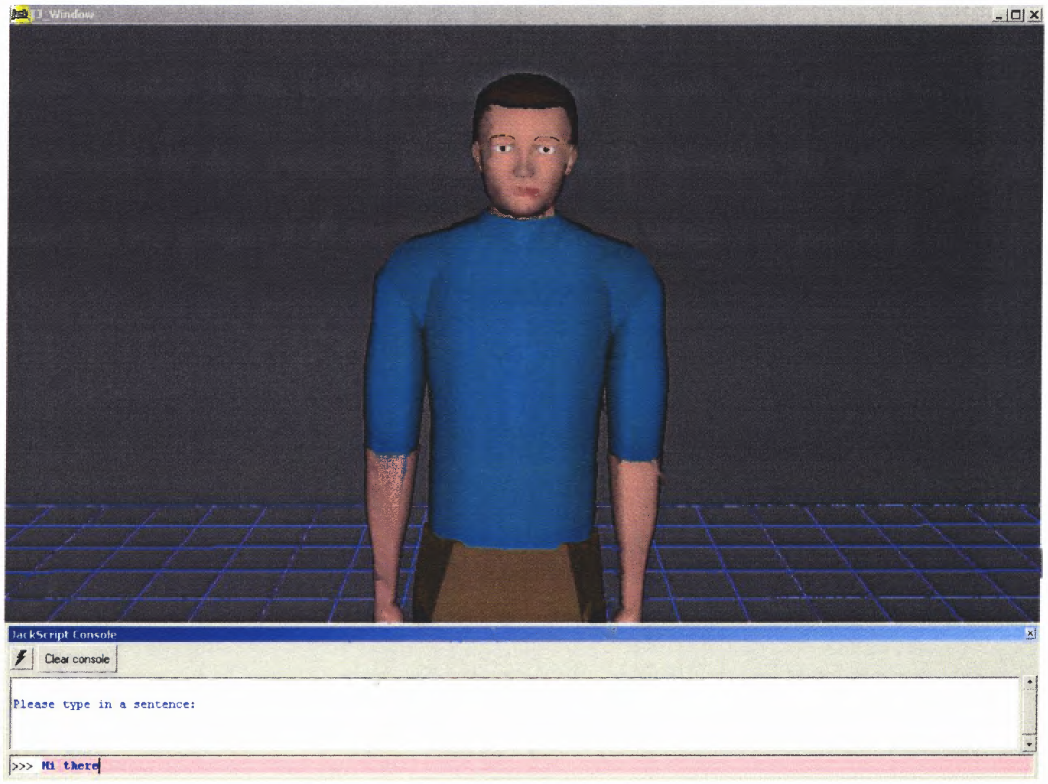


```
Python: 2.1.1 (#20, Jul 24 2001, 12:55:24) [MSC 32 bit (Intel)]
jsJack2._Setup(): Draw callback installed
```

```
>>>
```

Figure 4.2 Jack user interface showing Python icon and JackScript console.

Once the Sign Translator Program is launched, the human-model or avatar is automatically loaded in the graphics window, and the user is prompted to input an English sentence. At this point the JackScript console assumes the role of the input environment for the system, with user input typed in this window (Figure 4.3).



```
Please type in a sentence:
>>> Hi there
```

Figure 4.3 JackScript console is the input environment for the translator system.

The output environment is the graphics window that contains the avatar. Important consideration was given to the signing space of the avatar. The camera view was arranged such that the avatar was close enough for users to distinguish between handshapes, but wide enough to respect the sign envelope of all possible arm configurations used in signing. This space is limited by the waist line, the top of the head, and the reach of bent arms (see Figures 4.4 and 4.5) [8].

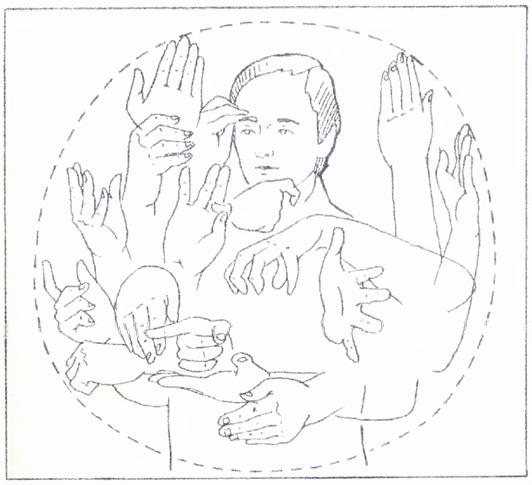


Figure 4.4 The region in which signs are made [8].

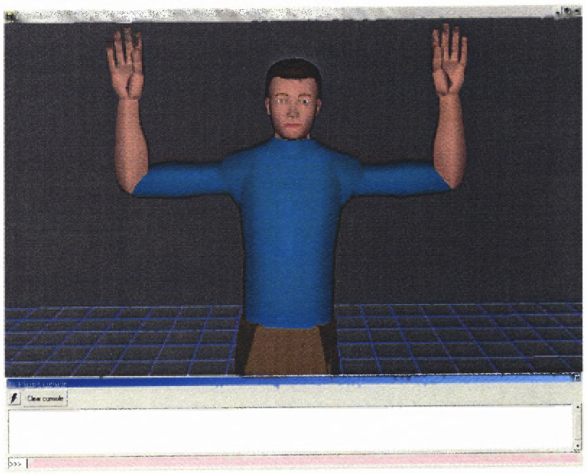


Figure 4.5 Signing space in the translator system.

4.2 Interpolation in the Sign Synthesis Process

A sign is synthesized in the Sign Translator System by obeying combined joint angle data imposed on the human-model by the appropriate parameter primes. Joint angle information from inverse kinematic calculations (to align cubes), the handshape library file, the appropriate wrist orientation variable, and the pertinent motion file (when applicable), are combined and obeyed in order to generate the desired sign representation. Jack's simulation engine sets the requested joints to the required configurations while still complying with the inherent joint constraints of the model. This results in realistic human like motions.

The Jack software produces interpolation between the requested configurations in order to produce continuous signing (Figure 4.6). Interpolation occurs between all dictated configurations, whether they are a part of the same sign or not. This allows the ending configuration of one sign to smoothly bend into the beginning configuration of the following sign, preventing the detached representations that animation techniques usually provide.

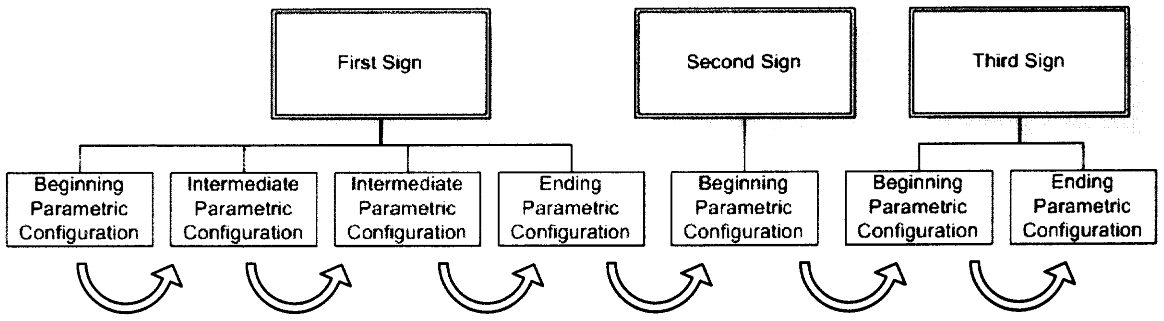

First Sign
Second Sign
Third Sign
Beginning Parametric Configuration
Intermediate Parametric Configuration
Intermediate Parametric Configuration
Ending Parametric Configuration
Beginning Parametric Configuration
Beginning Parametric Configuration
Ending Parametric Configuration

Figure 4.6 Interpolation in the Sign Translator System.

In defining a sign representation, the Sign Translator System allows the specification of the duration of the sign sequence. This is accomplished through the simulation engine using a real-time clock. As a result, if a sign is created with 'duration = 2', the sign will appropriately run for two seconds. However, the number of steps that will be computed during the allocated time will vary, depending on the complexity of the motion and the processing power of the computer [22].

The simulation engine clock can be forced to perform computations and steps as if they occurred every 1/30th of a second by adding the following line of code to the translator program:

```
ClockControl(dT=1/30., rt =0)
```

In this Python command the value assigned to 'rt' identifies the use of a real-time clock (rt = 1 indicates real-time clock, rt = 0 clock is not real-time). The variable 'dT' specifies the time step assumed between consecutive frames [22]. The use of these variables realizes a fast processor in order to avoid discontinuities in the frame rate.

4.3 Software Design

The program for sign translation was implemented in Python and calls a number of functions in JackScript. It consists of four parts (Figure 4.7); a main module in which all supporting files are loaded and user input is requested, and three sub-modules that are branched to in order process this input and produce the corresponding sign representations. The three sub-modules are: (1) the Word List module, which contains a list of the words that the system is capable of signing, (2) the Sign Word module, which contains the sign specifications for words in the list, and (3) the Sign Letter module that contains the sign specifications for each letter of the manual alphabet.

In the main module the user is requested to input the text to be translated. This text then under goes some basic string processing, where all letters are made lower case, and punctuation symbols or special characters are removed. The input is made lower case to ensure that words present in the Word List module can be matched with the corresponding words in the input string. This permits the user to be unconcerned with the case of letters or words that are cut and pasted, or typed in the system. The removal of punctuation symbols and special characters is a precautionary measure to prevent any discrepancies when matching words or letters to their corresponding sign definitions in the system. Even more importantly, fingerspelling and Sign Language do not use capitalization or punctuation to convey any messages, which makes them irrelevant for recognition and processing.

Next, the input string is parsed into its individual words, and the words are indexed to preserve the order in which the words are processed. The first word in the index is then sent to the Word List module where the list is checked for an occurrence of the word. If the word is present, it is then sent to the Sign Word module where its sign definition (specification), which is based on kinematic data, is used to generate the equivalent sign. If the word is not present in the word list, it is sent to the Sign Letter module where the letters of the word are indexed, and the equivalent handshape is generated for each letter of the word in the order of its occurrence.

The fully commented algorithm can be found in the Appendix A.

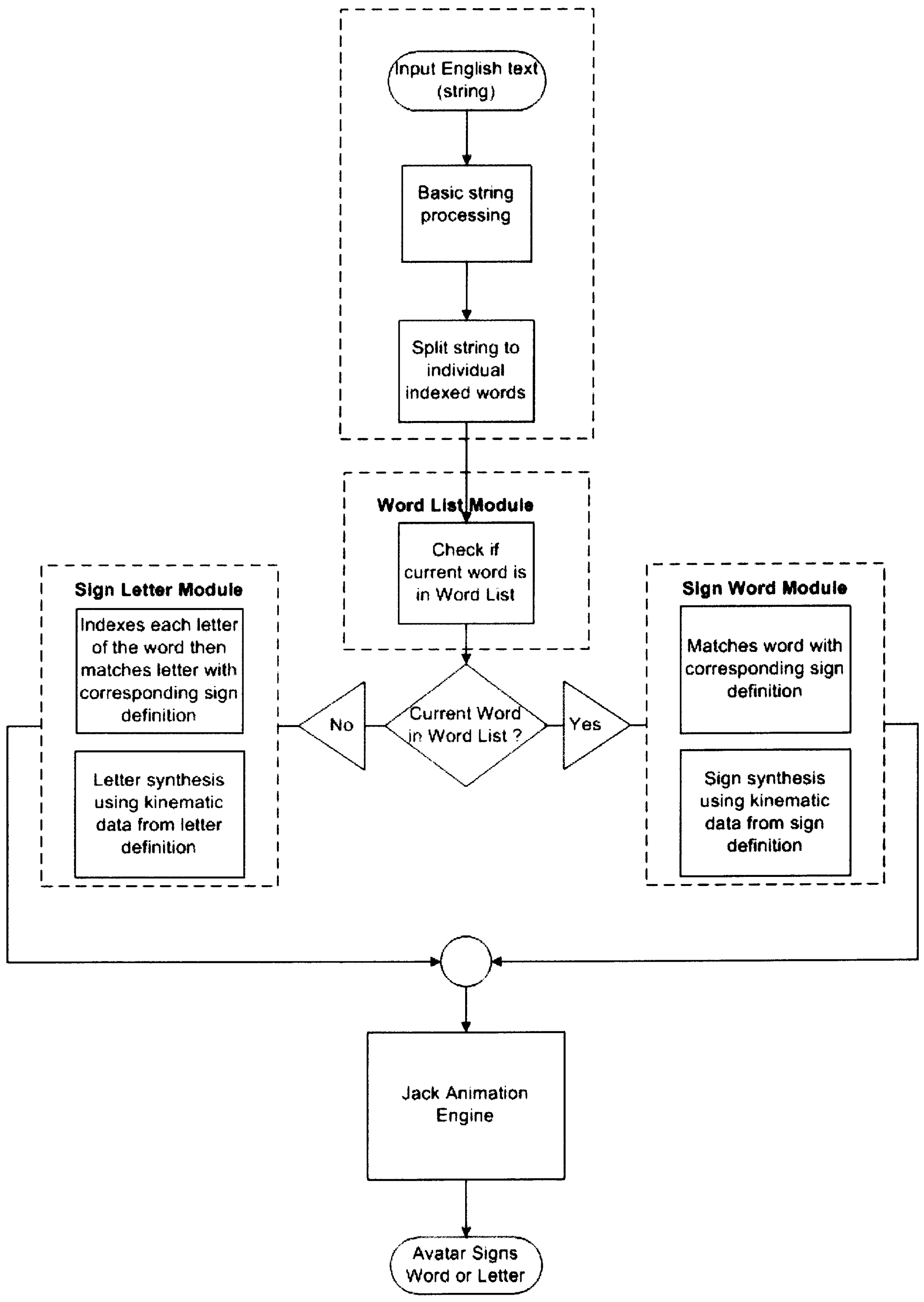

Input English text (string)
Basic string processing
Split string to individual indexed words
Word List Module
Check if current word is in Word List
Current Word in Word List ?
No
Yes
Sign Letter Module
Indexes each letter of the word then matches letter with corresponding sign definition
Letter synthesis using kinematic data from letter definition
Sign Word Module
Matches word with corresponding sign definition
Sign synthesis using kinematic data from sign definition
Jack Animation Engine
Avatar Signs Word or Letter

Figure 4.7 Schematic showing the architecture and data flow in the translator system.

CHAPTER 5

CONCLUSIONS

This thesis focused on the development of the foundation of a system that would receive English language input and generate a sequence of related signed gestures each synthesized from their basic kinematic parameters. A technique of sign specification for a computer-based translation system was developed through the use of Python objects and functions. Sign definitions (written as Python algorithms), as well as a supervisory sign synthesis application, were created to drive the simulation engine of a human-modeling software known as Jack.

The work presented in this thesis provides the foundation for a scalable Sign Language translation system that is capable of synthesizing intelligible signs based on parametric sign definitions. The prototype employs a commercial animation engine, Jack, to produce an avatar that physically resembles a human, and is capable of moving with human-like motion. The system has the potential of an unlimited vocabulary of signs, which depends only on the labor to code each sign in the format expected by the synthesis engine.

An informal viewing of sample signs and fingerspelled words out put by the system was held, and the results were very encouraging. The evaluators were two hearing individuals knowledgeable in both fingerspelling and Sign Language, and claimed that the signs were intelligible and natural in appearance. They noticed however, some minor production deficiencies, such as the hand penetrating the torso of the avatar due to Jack's internal inverse kinematics, which lacks collision avoidance. Should this be a problem in the use of the system, modifications to hand trajectories can be made by adding via points in the form of intermediate locations of virtual cubes along the desired trajectory. Subsequent versions of Jack are likely to remedy this problem however.

In the future the inventory of signs will be greatly increased, as more signs are code in the appropriate format. Currently, movements in the prototype are embedded in the way Jack moves the limbs from one location to the next. Some additional, secondary movements are not possible as of yet and will need to be specifically added in the future. Non-manual signals including facial expression, head movement, shoulder movement and eye gaze are very intricate details that have not been incorporated and are left for theses of the future.

The linguistic conversion from English grammar to ASL is also not included within the scope of this thesis. What has been produced is a working system that synthesizes ASL signs and fingerspelled words using English grammar in the form referred to as Manually Coded English. This is a solid foundation for continued research at the New Jersey Institute of Technology.

APPENDIX A

SIGN LANGUAGE TRANSLATOR PROGRAM

The algorithm for the translator program is divided into four modules, and hence four sections. Within the algorithm all indentations made are essential.

A.1 Main module: SignJack.py

```
sys.path.append('/Python23/prgmodule')
# adds the modules' directory to the search path

os.chdir('/Python23/prgmodule')
# makes '/Python23/prgmodule' the current working directory

import string
# imports string functions such as 'translate' and 'maketrans'
from WordList import List, WordCheckList

v = View()
prevloc = v.LookingFrom()
prevfoc = v.LookingAt()
# stores the current view parameters of the graphics window

LoadFile("/Program Files/jack30/library/data/SignJack1.env")
# loads the human-model with cubes and fingerspelling postures
letters = ReadHandShapeFile("/Program
Files/jack30/library/data/sys/JASignedEnglishHandshapes.data")
# loads all stored handshape data and names it 'letters'


v.LookFrom(xyz(-0.250591, -0.002582, -0.000661) * trans(-
7.147087, 165.768768, 158.128845))
v.LookAt(trans(-6.708587, 123.651581, -6.409498))
# adjusts the camera view to define the signing space


jack= scene.jack
rw = jack.right_wrist
lw = jack.left_wrist
# assigns Jackcore objects to Jackscript variable names


# the following postures were imported and saved in the env file
# lists all postures imported for fingerspelling
#jack.AddPosture('fingerspell_a.post', 'fingerspell_a')
#jack.AddPosture('fingerspell_b.post', 'fingerspell_b')
#jack.AddPosture('fingerspell_c.post', 'fingerspell_c')
```

```
#jack.AddPosture('fingerspell_d.post', 'fingerspell_d')
#jack.AddPosture('fingerspell_e.post', 'fingerspell_e')
#jack.AddPosture('fingerspell_f.post', 'fingerspell_f')
#jack.AddPosture('fingerspell_g.post', 'fingerspell_g')
#jack.AddPosture('fingerspell_h.post', 'fingerspell_h')
#jack.AddPosture('fingerspell_i.post', 'fingerspell_i')
#jack.AddPosture('fingerspell_j.post', 'fingerspell_j')
#jack.AddPosture('fingerspell_k.post', 'fingerspell_k')
#jack.AddPosture('fingerspell_l.post', 'fingerspell_l')
#jack.AddPosture('fingerspell_m.post', 'fingerspell_m')
#jack.AddPosture('fingerspell_n.post', 'fingerspell_n')
#jack.AddPosture('fingerspell_o.post', 'fingerspell_o')
#jack.AddPosture('fingerspell_p.post', 'fingerspell_p')
#jack.AddPosture('fingerspell_q.post', 'fingerspell_q')
#jack.AddPosture('fingerspell_r.post', 'fingerspell_r')
#jack.AddPosture('fingerspell_s.post', 'fingerspell_s')
#jack.AddPosture('fingerspell_t.post', 'fingerspell_t')
#jack.AddPosture('fingerspell_u.post', 'fingerspell_u')
#jack.AddPosture('fingerspell_v.post', 'fingerspell_v')
#jack.AddPosture('fingerspell_w.post', 'fingerspell_w')
#jack.AddPosture('fingerspell_x.post', 'fingerspell_x')
#jack.AddPosture('fingerspell_y.post', 'fingerspell_y')
#jack.AddPosture('fingerspell_z.post', 'fingerspell_z')
#jack.AddPosture('fingerspell_0.post', 'fingerspell_0')
#jack.AddPosture('fingerspell_1.post', 'fingerspell_1')
#jack.AddPosture('fingerspell_2.post', 'fingerspell_2')
#jack.AddPosture('fingerspell_3.post', 'fingerspell_3')
#jack.AddPosture('fingerspell_4.post', 'fingerspell_4')
#jack.AddPosture('fingerspell_5.post', 'fingerspell_5')
#jack.AddPosture('fingerspell_6.post', 'fingerspell_6')
#jack.AddPosture('fingerspell_7.post', 'fingerspell_7')
#jack.AddPosture('fingerspell_8.post', 'fingerspell_8')
#jack.AddPosture('fingerspell_9.post', 'fingerspell_9')
#jack.AddPosture('fingerspell_10.post', 'fingerspell_10')
# this is also the end postion for the ten sign
#jack.AddPosture('j_end.post', 'j_end')
#jack.AddPosture('z_end.post', 'z_end')
#jack.AddPosture('relaxed.post', 'relaxed')
#jack.AddPosture('stand_normal.post', 'stand_normal')
```

```
### Motions ###
jmotion = LoadChannelSet('/Program
Files/jack30/library/data/sys/j_chset.env')
j = jmotion.Bind([scene.jack], ['jack'])
zmotion = LoadChannelSet('/Program
Files/jack30/library/data/sys/z_chset.env')
z = zmotion.Bind([scene.jack], ['jack'])
tenmotion = LoadChannelSet('/Program
Files/jack30/library/data/sys/ten_chset.env')
ten = tenmotion.Bind([scene.jack], ['jack'])


### locations ###
start = scene.lcube
nose1 = scene.lcube0
nose2 = scene.lcube1
nose3 = scene.lcube2
midchest1 = scene.lcube3
midchest2 = scene.lcube4
midchest3 = scene.lcube5
midchest4 = scene.lcube6
lshoulder = scene.lcube7
reye    = scene.lcube8
rmouthcorner = scene.lcube9
leye    = scene.lcube10
lmouthcorner = scene.lcube11
point   = scene.lcube12



### end effectors ###
rindex = scene.fcube.cube.base
rthumb = scene.fcube0.cube.base
rmiddle = scene.fcube1.cube.base
rpalmcenter = scene.fcube2.cube.base

lindex = scene.fcube3.cube.base
lmiddle= scene.fcube4.cube.base
lthumb = scene.fcube5.cube.base


### wrist orientations ###
wor1 =(0,0,0)              # hand oriented vertical (handshake)
wor2 =(0,0,45*u.deg)
wor3 =(0,0,90*u.deg)       # palm down
wor4 =(0,0,-45*u.deg)
wor5 =(0,0,-90*u.deg)      # palm up
worflex1 = (0,-45*u.deg,0)
worflex2 = (0,45*u.deg,0)
# wrist orientation 1 => rw.Move(wor1, duration=1)
```

```
reply = 'yes'
while 'yes' == reply:

    sentence = raw_input("Please type in a sentence:\n")
    print sentence


    sentence = string.lower(sentence)
    # because input words need to match exactly with words in
    # the WordList (case sensitive)

    nonletnum ="$!@#$%^&*,;:+_-~=`><.?/\[]{}()'"
    # all punctuation symbols and special characters are
    # stripped away from input string
    transtable = string.maketrans('', '')
    # builds a list of all characters
    sentence = string.translate(sentence,transtable,nonletnum)
    # deletes all non-letter and non-number characters
    #(incase enter's text with punctuation)

    sentence = sentence+' #'
    # the '#' character tells Jack to relax at the end of a
    # sign request in the 'SignLetter.py' file
    # a space was needed before the '#' to separate it from the
    # last word in the sentence, allowing the word to be
    # recognized in the wordlist

    word = sentence.split(" ",)
    # uses the space , " ", between words as a delimiter
    # allocates each word from the input string to the variable
    # name 'word'. The variable is indexed so that each word is
    # allocated a number

    for i in range(len(word)):
    # loops as many times as there are words in the input
    # string

        wordi = word[i]
        # can't put word[i] in the function def
        # so used wordi instead
        if (List(wordi)):
        # checks to see if current word
        # is in the WordList (returns '1' if it is)
            execfile('SignWord.py')
            # branches to 'Sign Word' sub-module

        else:
            execfile('SignLetter.py')
            # branches to 'Sign Letter' sub-module
```

```
          else:
          # else for the 'for' loop
               reply = raw_input("Do another ?: ")
               # requests another input when the previous one is
               # processed
               print reply
               while (reply !='yes' and reply != 'no'):
                    print 'Not a valid reply'
                    reply = raw_input("Do another ?(yes or no): ")
                    print reply


     else:

          print 'Thank you for using the program\n'
          Destroy(jack)
          v.LookFrom(prevloc)
          v.LookAt(prevfoc)
          # returns camera view to its original viewing configuration
          # prior to initiating the Sign Jack program
```

A.2 Sub-module: WordList.py

```
# list is case sensitive, so  the entire input string was made
# lower case

# made 'WordCheckList' a global variable by putting it in the top
# level of this module

# the word list is extensible, new words can be added within the
# brackets

WordCheckList = ['eye', 'i', 'me', 'my', 'nose', 'you']

def List(wordi):

     InList = wordi in WordCheckList
     return InList    # returns a value (0 or 1) to the function
                      # that called it, indicating whether the
                      # word is present in the list or not
```

A.3 Sub-module: SignWord.py

```
# the sign for words are generated by combining information
# pertaining to which hand or hands are used in a sign, the
# handshape assumed, the location of the sign, the orientation of
# the hand, and any additional motion, if necessary

if(wordi =='nose' ):
     DoTogether(jack.Reach('right',nose2,endeff= rindex,duration
     =2),HandShapeMotion(jack,letters['1'],'right',duration
     =1.5),rw.Move(worl,duration =1.5))
     Flush()    # needed to ensure that the motion generated by
                # the above script is carried out to completion
                # before any other line of code is executed

elif(wordi == 'my'):
     DoTogether(jack.Reach('right',midchest1,endeff=
     rpalmcenter,duration = 2),HandShapeMotion(jack,letters
     ['OpenB'],'right',duration =1.5),rw.Move(worl,duration
     =1.5))
     Flush()

elif(wordi == 'i'):
     DoTogether(jack.Reach('right',midchest4,endeff=
     rpalmcenter,duration = 2),HandShapeMotion(jack,letters['I']
     ,'right',duration = 1.5),rw.Move((0,45*u.deg,0 ),duration
     = 1.5))
     Flush()

elif(wordi=='eye'):
     DoTogether(jack.Reach('right',reye,endeff= rindex,duration
     = 2),HandShapeMotion(jack,letters['1'],'right',duration
     =1.5),rw.Move((0,0,0 ),duration = 1.5))
     Flush()

elif(wordi =='me'):
     DoTogether(jack.Reach('right',midchest2,endeff=
     rindex,duration = 2),HandShapeMotion(jack,letters['1'],
     'right',duration =1.5),rw.Move(worflex1,duration = 1.5))
     Flush()

elif(wordi =='neutral'):
     DoTogether(jack.Reach('right',start,endeff= rindex,duration
     = 2.0),HandShapeMotion(jack,letters['1'],'right',duration
     =1.5),rw.Move((0,0,0 ),duration = 1.5))
     Flush()
```

```
elif(wordi =='you'):
     DoTogether(jack.Reach('right',point,endeff= rindex,duration
     = 2),HandShapeMotion(jack,letters['1'],'right',duration
     =1.5),rw.Move(worl,duration = 1.5))
     Flush()

else:
     pass          # represents that nothing is done here and that
                   # the control returns to the  higher-level module
```

A.4 Sub-module: SignLetter.py

```
jack._CachePostures()
# places all the posture files in the cache so that they may be
# accessed a lot faster than if they were present only on the
# hard drive

n = 0
# used the variable 'n' in-order to be able to use the subsequent
# letters in conditional statements, eg.(1&0 is ten)

preletter='$'
wordi = wordi+' '
# This space is integral, it allows jack to hold letters for
# awhile, indicating a space between one spelt word and another
# (rochester method)

for letter in wordi:

     if(letter =='1' and wordi[n + 1] == '0'):
     # necessary to evaluate 10 before one and zero are
     # evaluated individually

          DoInOrder(jack.Pose("fingerspell_10", duration =  0.4,
          style = EaseInEaseOut), ten.Run(duration=1))
          Flush()
          preletter=letter

     elif(letter =='0' and wordi[n - 1] == '1'):
     # necessary to prevent a zero from being signed after a ten
     # is signed since the for loop evaluates each sentence
     # letter by letter
          preletter=letter
          # once a letter or number is signed, that character
          # then becomes the previous letter

     elif((letter== preletter and letter =='j')):
          DoInOrder(jack.Pose("fingerspell_j", duration =  0.4,
          style = EaseInEaseOut), j.Run(duration=0.5))
          Flush()
          preletter=letter

     elif((letter== preletter and letter =='z')):
          DoInOrder(jack.Pose("fingerspell_z", duration =  0.4,
          style = EaseInEaseOut), z.Run(duration=0.5))
          Flush()
          preletter=letter
```

```
elif(letter == preletter):
     DoInOrder(jack.right_elbow.Move((2.42,), duration =
     0.15,style=Constant),
     jack.right_elbow.Move((2.3561944961547852,), duration
     = 0.15, style= Accelerate))
     # produces the back and forth movement necessary
     # when the same letter appear more then once
     # contiguously
     Flush()

elif(letter =='#'):
     jack.Pose("relaxed", duration = 1, style =
     EaseInEaseOut)
     # relaxes Jack's hands to the sides of his torso when
     # there are no more words of letters left to
     # fingerspell
     Flush()
     preletter=letter

elif(letter ==' '):
# takes care of situations where a letter or number that
# requires secondary movement occurs prior to a space
     if(preletter == 'z'):
          jack.Pose("z_end", duration = 0.5, style =
          EaseInEaseOut)
          # provides a posture for 'z' that can be held
          Flush()
          preletter=letter

     elif(preletter == 'j'):
          jack.Pose("j_end", duration = 0.5, style =
          EaseInEaseOut)
          Flush()
          preletter=letter

     elif(preletter == '0' and wordi[n-2] == '1'):
     # needs be 'n-2' since n is now the space and 'n-1' is
     # the zero
          jack.Pose("fingerspell_10", duration = 0.5, style
          = EaseInEaseOut)
          Flush()
          preletter=letter

     elif(preletter == '#'):
          pass

     else:
          jack.Pose("fingerspell_"+ preletter, duration
          =0.5, style = EaseInEaseOut)
          Flush()
          preletter=letter
```

```
elif(letter =='j'):
     DoInOrder(jack.Pose("fingerspell_j", duration =  0.4,
     style = EaseInEaseOut), j.Run(duration=0.5))
     # adds motion for 'j' to the sign
     Flush()
     preletter=letter

elif(letter =='z'):
     DoInOrder(jack.Pose("fingerspell_z", duration =  0.4,
     style = EaseInEaseOut), z.Run(duration=0.5))
     Flush()
     preletter=letter

elif(letter):
# condenses the code by adding the letter that occurs to
# name of the file that needs to be retrieved
     jack.Pose("fingerspell_"+letter, duration =0.5, style
     = EaseInEaseOut)
     Flush()
     preletter=letter



else:
     Pass

n = n+1
```

APPENDIX B

LIST OF JACK PARTS: SITES, SEGMENTS, AND JOINTS

The following is a comprehensive list of the two hundred and sixty eight parts of the human-model.

B.1 Sites

Jack.left_eyeball.sight	Jack.left_eyeball.base
Jack.right_eyeball.sight	Jack.right_eyeball.base
Jack.right_foot.distal	Jack.right_foot.right
Jack.right_foot.toes	Jack.right_foot.left
Jack.right_foot.proximal	Jack.right_foot.normal
Jack.right_foot.top	Jack.right_foot.heel
Jack.right_foot.bottom	Jack.right_foot.new_heel
Jack.left_foot.distal	Jack.left_foot.toes
Jack.left_foot.normal	Jack.left_foot.proximal
Jack.left_foot.heel	Jack.left_foot.new_heel
Jack.right_toes.toetip	Jack.right_toes.proximal
Jack.right_toes.distal	Jack.left_toes.toetip
Jack.left_toes.distal	Jack.left.toes.proximal
Jack.right_lower_leg.proximal	Jack.right_lower_leg.knee
Jack.right_lower_leg.left	Jack.right_lower_leg.front
Jack.right_lower_leg.right	Jack.right_lower_leg.back
Jack.right_lower_leg.distal	Jack.left_lower_leg.proximal
Jack.left_lower_leg.distal	Jack.right_upper_leg.left
Jack.right_upper_leg.front	Jack.right_upper_leg.proximal
Jack.right_upper_leg.right	Jack.right_upper_leg.back
Jack.right_upper_leg.poplit	Jack.right_upper_leg.poplit2
Jack.right_upper_leg.distal	Jack.right_upper_leg.knee
Jack.left_upper_leg.distal	Jack.left_upper_leg.proximal
Jack.upper_torso.proximal	Jack.upper_torso.distal
Jack.upper_torso.left	Jack.upper_torso.right
Jack.upper_torso.lclav	Jack.upper_torso.rclav
Jack.bottom_head.right_eyeball	Jack.bottom_head.eye_level
Jack.bottom_head.sight	Jack.bottom_head.front
Jack.bottom_head.left_eyeball	Jack.bottom_head.proximal
Jack.bottom_head.right	Jack.bottom_head.menton
Jack.bottom_head.left	Jack.bottom_head.bottom
Jack.bottom_head.top	Jack.bottom_head.top_side
Jack.bottom_head.top_out	Jack.bottom_head.eye_lvl_out

Jack.bottom_head.back	Jack.neck.distal
Jack.neck.base_rt	Jack.neck.back
Jack.neck.proximal	Jack.neck.front
Jack.right_clavicle.lateral	Jack.right_clavicle.acromion
Jack.right_clavicle.top	Jack.right_clavicle.shoulder_back
Jack.right_clavicle.proximal	Jack.left_clavicle.acromion
Jack.left_clavicle.lateral	Jack.left_clavicle.proximal
Jack.right_upper_arm.distal	Jack.right_upper_arm.right
Jack.right_upper_arm.front	Jack.right_upper_arm.back
Jack.right_upper_arm.left	Jack.right_upper_arm.proximal0
Jack.right_upper_arm.proximal	Jack.right_upper_arm.deltoid
Jack.right_upper_arm.acromion	Jack.right_upper_arm.shoulder_level
Jack.right_upper_arm.shlder_lvl_out	Jack.left_upper_arm.deltoid
Jack.left_upper_arm.proximal0	Jack.left_upper_arm.proximal
Jack.left_upper_arm.acromion	Jack.left_upper_arm.distal
Jack.right_lower_arm.distal	Jack.right_lower_arm.left
Jack.right_lower_arm.back	Jack.right_lower_arm.right
Jack.right_lower_arm.front	Jack.right_lower_arm.proximal
Jack.right_lower_arm.elbow	Jack.left_lower_arm.distal
Jack.left_lower_arm.proximal	Jack.t1.distal
Jack.t1.cy2	Jack.t1.proximal
Jack.t1.origin	Jack.t2.distal
Jack.t2.front	Jack.t2.proximal
Jack.t2.origin	Jack.t2.back
Jack.t3.distal	Jack.t3.origin
Jack.t3.proximal	Jack.t4.front
Jack.t4.distal	Jack.t4.proximal
Jack.t4.origin	Jack.t4.back
Jack.t4.interscye_left	Jack.t4.interscye_right
Jack.t5.distal	Jack.t5.origin
Jack.t5.proximal	Jack.t6.distal
Jack.t6.proximal	Jack.t6.origin
Jack.t7.right	Jack.t7.front
Jack.t7.distal	Jack.t7.origin
Jack.t7.proximal	Jack.t7.back
Jack.t7.left	Jack.t8.distal
Jack.t8.proximal	Jack.t8.origin
Jack.t9.distal	Jack.t9.proximal
Jack.t9.origin	Jack.t10.distal
Jack.t10.proximal	Jack.t10.origin
Jack.t10.back	Jack.t11.distal
Jack.t11.proximal	Jack.t11.origin
Jack.t12.distal	Jack.t12.proximal
Jack.t12.origin	Jack.l1.distal
Jack.l1.proximal	Jack.l2.distal
Jack.l2.proximal	Jack.l3.distal
Jack.l3.proximal	Jack.l4.distal

Jack.l4.proximal	Jack.l5.front
Jack.l5.distal	Jack.l5.proximal
Jack.l5.left	Jack.l5.back
Jack.l5.right	Jack.lower_torso.proximal0
Jack.lower_torso.distal	Jack.lower_torso.center_of_mass
Jack.lower_torso.h_point	Jack.lower_torso.front
Jack.lower_torso.lhip_lateral	Jack.lower_torso.rhip_lateral
Jack.lower_torso.ldistal	Jack.lower_torso.rdistal
Jack.lower_torso.left_side	Jack.lower_torso.left_hiphandle
Jack.lower_torso.crotch_level	Jack.lower_torso.right_side
Jack.lower_torso.proximal	Jack.lower_torso.butt
Jack.lower_torso.right_hiphandle	Jack.lower_torso.sit_ext
Jack.lower_torso.seat	Jack.lower_torso.floor
Jack.left_palm.thumb0	Jack.left_palm.palmcenter
Jack.left_palm.base	Jack.left_palm.leftbird
Jack.left_palm.f33	Jack.left_palm.f22
Jack.left_palm.f44	Jack.left_palm.f11
Jack.right_palm.f33	Jack.right_palm.f22
Jack.right_palm.rightbird	Jack.right_palm.palmcenter
Jack.right_palm.back	Jack.right_palm.front
Jack.right_palm.f44	Jack.right_palm.f11
Jack.right_palm.left	Jack.right_palm.right
Jack.right_palm.thumb0	Jack.right_palm.real_base
Jack.right_palm.base	Jack.left_finger32.base0
Jack.left_finger32.tip	Jack.left_finger31.base0
Jack.left_finger31.tip	Jack.left_finger30.base0
Jack.left_finger30.tip	Jack.left_finger22.base0
Jack.left_finger22.tip	Jack.left_finger21.base0
Jack.left_finger21.tip	Jack.left_finger20.base0
Jack.left_finger20.tip	Jack.left_finger12.base0
Jack.left_finger12.tip	Jack.left_finger11.base0
Jack.left_finger11.tip	Jack.left_finger10.base0
Jack.left_finger10.tip	Jack.left_finger02.base0
Jack.left_finger02.tip	Jack.left_finger01.base0
Jack.left_finger01.tip	Jack.left_finger00.base0
Jack.left_finger00.tip	Jack.left_thumb2.base0
Jack.left_thumb2.tip	Jack.left_thumb1.base0
Jack.left_thumb1.tip	Jack.left_thumb0.base0
Jack.left_thumb0.tip	Jack.right_thumb2.base0
Jack.right_thumb2.tip	Jack.right_thumb1.base0
Jack.right_thumb1.tip	Jack.right_thumb0.base0
Jack.right_thumb0.tip	Jack.right_finger32.base0
Jack.right_finger32.tip	Jack.right_finger31.base0
Jack.right_finger31.tip	Jack.right_finger30.base0
Jack.right_finger30.tip	Jack.right_finger22.base0
Jack.right_finger22.tip	Jack.right_finger21.base0
Jack.right_finger21.tip	Jack.right_finger20.base0

Jack.right_finger20.tip	Jack.right_finger12.base0
Jack.right_finger12.tip	Jack.right_finger11.base0
Jack.right_finger11.tip	Jack.right_finger10.base0
Jack.right_finger10.tip	Jack.right_finger02.base0
Jack.right_finger02.tip	Jack.right_finger01.base0
Jack.right_finger01.tip	Jack.right_finger00.base0
Jack.right_finger00.tip	Jack.hair.top

B.2 Segments

Jack.left_eyeball	Jack.right_eyeball
Jack.left_foot	Jack.right_foot
Jack.left_toes	Jack.right_toes
Jack.left_lower_leg	Jack.right_lower_leg
Jack.left_upper_leg	Jack.right_upper_leg
Jack.upper_torso	Jack.bottom_head
Jack.neck	Jack.left_clavicle
Jack.right_clavicle	Jack.left_upper_arm
Jack.right_upper_arm	Jack.left_lower_arm
Jack.right_lower_arm	Jack.t1
Jack.t2	Jack.t3
Jack.t4	Jack.t5
Jack.t6	Jack.t7
Jack.t8	Jack.t9
Jack.t10	Jack.t11
Jack.t12	Jack.l1
Jack.l2	Jack.l3
Jack.l4	Jack.l5
Jack.lower_torso	Jack.left_palm
Jack.right_palm	Jack.left_finger32
Jack.left_finger31	Jack.left_finger30
Jack.left_finger22	Jack.left_finger21
Jack.left_finger20	Jack.left_finger12
Jack.left_finger11	Jack.left_finger10
Jack.left_finger02	Jack.left_finger01
Jack.left_finger00	Jack.left_thumb2
Jack.left_thumb1	Jack.left_thumb0
Jack.right_finger32	Jack.right_finger31
Jack.right_finger30	Jack.right_finger22
Jack.right_finger21	Jack.right_finger20
Jack.right_finger12	Jack.right_finger11
Jack.right_finger10	Jack.right_finger02
Jack.right_finger01	Jack.right_finger00
Jack.right_thumb2	Jack.right_thumb1
Jack.right_thumb0	Jack.hair
Jack.left_eyeball	Jack.right_eyeball
Jack.left_foot	Jack.right_foot
Jack.left_toes	Jack.right_toes
Jack.left_lower_leg	Jack.right_lower_leg
Jack.left_upper_leg	Jack.right_upper_leg
Jack.upper_torso	Jack.bottom_head
Jack.neck	Jack.left_clavicle
Jack.right_clavicle	Jack.left_upper_arm
Jack.right_upper_arm	Jack.left_lower_arm
Jack.right_lower_arm	Jack.t1

Jack.t2	Jack.t3
Jack.t4	Jack.t5
Jack.t6	Jack.t7
Jack.t8	Jack.t9
Jack.t10	Jack.t11
Jack.t12	Jack.l1
Jack.l2	Jack.l3
Jack.l4	Jack.l5
Jack.lower_torso	Jack.left_palm
Jack.right_palm	Jack.left_finger32
Jack.left_finger31	Jack.left_finger30
Jack.left_finger22	Jack.left_finger21
Jack.left_finger20	Jack.left_finger12
Jack.left_finger11	Jack.left_finger10
Jack.left_finger02	Jack.left_finger01
Jack.left_finger00	Jack.left_thumb2
Jack.left_thumb1	Jack.left_thumb0
Jack.right_finger32	Jack.right_finger31
Jack.right_finger30	Jack.right_finger22
Jack.right_finger21	Jack.right_finger20
Jack.right_finger12	Jack.right_finger11
Jack.right_finger10	Jack.right_finger02
Jack.right_finger01	Jack.right_finger00
Jack.right_thumb2	Jack.right_thumb1
Jack.right_thumb0	Jack.hair

B.3 Joints

Jack.rthumb1	Jack.rthumb2
Jack.rthumb0	Jack.rpinfinger31
Jack.rpinfinger32	Jack.rringfinger22
Jack.rringfinger21	Jack.rmidfinger12
Jack.rmidfinger11	Jack.rinfinger02
Jack.rinfinger01	Jack.right_finger00
Jack.right_finger10	Jack.right_finger20
Jack.right_finger30	Jack.right_toes
Jack.left_toes	Jack.right_ankle
Jack.left_ankle	Jack.right_knee
Jack.left_knee	Jack.right_hip
Jack.left_hip	Jack.altanto_occipital
Jack.base_of_neck	Jack.spinet2t1
Jack.spinet3t2	Jack.spinet4t3
Jack.spinet5t4	Jack.spinet6t5
Jack.spinet7t6	Jack.spinet8t7
Jack.spinet9t8	Jack.spinet11t10
Jack.spinet12t11	Jack.spinel1t12
Jack.spinel2l1	Jack.spinel3l2
Jack.spinel4l3	Jack.spinel5l4
Jack.waist	Jack.solar_plexus
Jack.right_clavicle_joint	Jack.right_shoulder
Jack.right_elbow	Jack.right_wrist
Jack.left_elbow	Jack.left_wrist
Jack.left_clavicle_joint	Jack.left_shoulder
Jack.lthumb1	Jack.lthumb2
Jack.lthumb0	Jack.lpinfinger31
Jack.lpinfinger32	Jack.lringfinger22
Jack.lringfinger21	Jack.lmidfinger12
Jack.lmidfinger11	Jack.linfinger02
Jack.linfinger01	Jack.left_finger00
Jack.left_finger10	Jack.left_finger20
Jack.left_finger30	

APPENDIX C

JACKSCRIPT FUNCTIONS

The following JackScipt functions were integral in the generation sign representations in the translator program. This appendix provides a full description of each of these functions.

Reach(self, side, goal, jfrom='shoulder', endeff='palm', reach_duration=0, poweight=0.3, ptype='point_to_point', otype='align_frame', duration=0, start=1, style=EaseInEaseOut)

An arm motion that reaches a goal and holds to it.

`side`	"left" or "right"
`goal`	the goal object or location. If this is an object, and it moves in parallel to this motion, the arm will follow
`jform`	"shoulder" (the default) or "waist". The starting joint for the reach
`endeff`	the end-effector: "palm" (the default), "forearm" or a Site, which must be on an object attached to the human's hand. In the first two cases, the effector is the hand's "palmcenter" Site. In the case of "forearm", the wrist joint doesn't take part in the motion
`reach_duration`	the duration of the "reach" part of the motion, i.e., the part where the end-effector is moving progressively closer to the goal. If 0 (the default), the reach duration will be set to half the total duration
`poweight`	position/orientation weight default = 1.0 (100% position)
`ptype`	positional type. allowed types are: "none", "point_to_point", "point_to_line", "point_to_plane", "limit_spring", "rest_angle" default = "point_to_point"
`otype`	orientation type. default = "none"
`duration`	the length of motion, in seconds
`start`	if true, the created motion will immediately begin to execute. Otherwise, the motion object won't execute until it's explicitly started with the Start() method
`style`	one of the keywords "Constant", "Accelerate", "Decelerate" or "EaseInEaseOut", describing the motion's velocity profile

Pose(self, name, anchor=None, use='both', *args, **kws)

Posture the body (motion).
Return a Motion object that postures the body.Example:
```
human.Pose(name="stand_working", anchor=human.right_foot.distal)
```

`name`	the posture's name (a string). If filename is specified, the name is ignored
`anchor`	a Site on the posed Figure. This site's location will be kept constant as the posture changes. For example, when a human's posture changes from standing to sitting, we would most likely want the foot to be the anchor (Default: the figure's root; for Humans, "right_toes.toetip")
`use`	one of "both" (default), "trans" or "xyz". Specifies which parts of the anchor's location matrix are maintained. For example, if the foot is not flat and the toes are used as anchor, the figure could end up tilted. In this case, it may make sense to have "use='trans'", i.e., use only translational part.
`duration`	the length of motion, in seconds
`start`	if true, the created motion will immediately begin to execute. Otherwise, the motion object won't execute until it's explicitly started with the Start() method
`style`	one of the keywords "Constant", "Accelerate", "Decelerate" or "EaseInEaseOut", describing the motion's velocity profile

HandShapeMotion(human, handshape, side, duration, start, style)

`human`	a human model
`handshape`	a handshape to be interpolated
`side`	"right", "left" or "both", denting which hand or hands will be affected by the motion

DoInOrder(*args, **kws)

Execute actions sequentially.
For example, `DoInOrder(a1,a2,a3)`

`start`	if 0, don't start the actions, just declare.

DoTogether(*args, **kws)

Execute actions in parallel.
For example, `DoTogether(a1,a2,a3)`

`start`	if 0, don't start the action

REFERENCES

[1] Wilbur, R. B., *American Sign Language and Sign Systems,* University Park Press, Baltimore, 1979.

[2] Luciano, D., Sherman, J., Vander, A., *Human Physiology: the mechanisms of body function*, 7th edition, Von Hoffman Press, CA, 1998.

[3] McCormick, B., *Understanding Deafness: an introductory guide to the different types of deafness; hearing tests; communication and language*, 4th edition, The National Deaf Children's Society, June 2003.

[4] Wright, T., *Understanding Deafness and Tinnitus*, Poole: Family Doctors Association, 2003.

[5] Marieb, E. N., *Human Anatomy and Physiology*, Benjamin/Cummings, Inc, CA, 1989.

[6] Sherwood, L., *Human Physiology: from cells to systems*, 4th edition, Brooks/Cole, Inc, CA, 2001.

[7] The National Deaf Children's Society, Deaf children and young people in the hospital – a guide for professionals, 2003 [retrieved April 15, 2004], http://www.ndcs.org.uk/docs/Deaf_children_and_young_people _in_hospital __text.pdf.

[8] Klima, K. S., Bellugi U., *The Signs of Language,* Harvard University Press, Massachusetts, 1979.

[9] Grayson, G., *Talking With Your Hands Listening With Your Eyes,* Square One Publishers, New York, 2003.

[10] Stokoe, W., *Classification and Description of Sign Languages,* In T. Sebeok (ed.), Current Trends in Linguistics 12, Mouton, The Hague, 1972.

[11] Humphries, T., Padden, C., O'Rourke, T., *A Basic Course in American Sign Language,* T.J Pulishers, Silver Spring, Maryland, 1980.

[12] Shroyer, E., *Signs of the Times,* Gallaudet University Press, Washington, D.C, 1982.

[13] Kawai H., Tamura S., "Deaf-And-Mute Sign Language Generation System", *Pattern Recognition*, Vol. 18, Nos. ¾, 1985, pp. 199-205.

[14] M. Waldron, B.C.H. Choi, "Text to Fingerspelling and Speech on the Amiga Micro-Computer", *IEEE/Ninth Annual Conference of the Engineering in Medicine and Biology Society*, 1987, pp. 1769-1770.

[15] ASL Fingerspelling Converter, [retrieved April 19, 2004], http://where.com/scott.net/asl/.

[16] Language Learning, ASL American Sign Language, Laser Publishing Group, 2001. (CD-ROM)

[17] Michigan State University, Comm Tech Lab- ASL Brower and Personal Communicator, 2002 [retrieved April 19, 2004], http://commtechlab.msu.edu/products/asl/index.html.

[18] Geitz, S., Hanson, T., Maher S., "Computer Generated 3-Dimensioanl Models of Manual Alphabet Handshapes for the World Wide Web", *Assets'96 (Vancouver, British Colombia Canada, April 11-12)*, 1996, ACM, pp. 27-31.

[19] Ling, L., Long, S., "An On-line Sign Language Communication System", *IEEE/Second international Conference on Web information Systems Engineering (Wise '01)*, Vol. 1, 2001, pp. 142- 148.

[20] Lu, S., Seiji, Igi., Matsuo, H., Nagashima, Y., "Towards a Dialogue System Based on Recognition and Synthesis of Japanese Sign Language", *Lecture Notes in Computer Science*, Vol. 1371, 1998, p. 259.

[21] Schneider, E., American Sign Language (ASL) vs. Signed English (SE), 2001, [retrieved April 19, 2004], http://www.lessontutor.com/eesASLIntro.html.

[22] Electronics Data Systems, JackScript Tutorial, 1999, [retrieved April 24, 2004], http://support.plms-eds.com/docs/jack.shtml.

[23] Lessa, A. S., *Python Developer's Handbook,* Sams, Indiana, 2001.

[24] Brown, M., *Python: the complete reference,* McGraw-Hill, New York, 2001.

[25] Harms, D., McDonald, K., *Quick Python Book,*Manning, Connecticut, 2000.

[26] Martin, J., A Linguistic Comparison: Two Notation Systems for Sign Language, 2002, [retrieved April 24, 2004], http://www.signwriting.org/forums/linguistics/ling008.html.

[27] Stokoe, W., Casterline, D., Croneberg, C., *A Dictionary of American Sign Language*, Linstok Press, United States, 1976.

[28] Bornstein, H., Jordan, I., *Functional Signs*, Pro-Ed, Austin, Texas, 1984.

[29] Humphries, T., O'Rourke, T., *A Basic Course in American Sign Language*, Silver Spring, MD, 1980.

[30] Deaf Missions' Animated Dictionary of Religious Signs, ASL Manual Alphabet, [retrieved April 24, 2004], www.deafmissions.com/ dic/ASLabc.html.

[31] Bornstein, H., Saulnier, K., *The Signed English Starter*, Gallaudet University Press, Washington, D.C, 1984.

[32] Lane, L., *Gallaudet Survival Guide to Signing*, Gallaudet University Press, Washington DC, 1990.