1-1-1998

Voice-interactive computer graphics and games

Nataraj Jeedigunta
University of Nevada, Las Vegas

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

VOICE INTERACTIVE COMPUTER GRAPHICS AND GAMES

by

Nataraj Jeedigunta

Bachelor of Science
Andhra University, India
1996

A thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science

in

Computer Science

Department of Computer Science
University of Nevada, Las Vegas
May 1998

UNLV

Thesis Approval
The Graduate College
University of Nevada, Las Vegas

April 14, 1998

The Thesis prepared by

Nataraj Jeedigunta

Entitled

Voice Interactive Computer Graphics and Games

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

Examination Committee Chair

Dean of the Graduate College

Examination Committee Member

Examination Committee Member

Graduate College Faculty Representative

ABSTRACT

Voice Interactive Computer Graphics and Games

by

Nataraj Jeedigunta

Dr. Evangelos A Yfantis, Examination Comittee Chair
Professor of Computer Science
University of Nevada, Las Vegas

The art and science of speech recognition have been advanced to the state where it is now possible to communicate reliably with a computer by speaking to it in a disciplined manner using a vocabulary of moderate size. There are many applications of speech recognition technology in telecommunications and other fields. *Voice Interactive Computer Graphics And Games* (VICGG) describes and emphasizes how voice recognition can be applied to design a real-time interactive graphics system. The VICGG system consists of a recognition module and a graphics module.

The recognition module is used for isolated word recognition. Three separate implementations of recognition algorithms have been evaluated. The first algorithm illustrates the effect of noise reduction filters in the recognition process. The second algorithm uses a noise-gate in the recognition process. The third algorithm is an improved version of the first two and is implemented using short-time Fourier analysis and bank of filters. The relative merits and demerits of these algorithms have been discussed. Using a vocabulary of moderate size, accuracy and real-time performance have been achieved in the third algorithm.

The graphics module has a set of 3-dimensional Dinosaur animations, which are

totally controlled by the recognition module. The event driven graphics module is implemented using OpenGL. The recognized word from the recognition module is linked to a corresponding animation in the graphics module.

The VICGG system demonstrates how speech technology will play an increasingly important role in the current multimedia era. This research also illustrates the applications of speech recognition technology in diverse fields.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Evangelos A Yfantis and Mr. Alexander Popovich of computer science department for providing me with much of the vision and energy for this work. I would like to thank my committee members, Dr. Roy H. Ogawa, Dr. Laxmi P. Gewali and Dr. Sahjendra Singh. I also thank my family and friends for their encouragement.

CHAPTER 1

INTRODUCTION

The vision of Speech Recognition Technology (SRT) in the twenty-first century is to provide seamless, high-quality interactive applications between people (or groups of people) and machines, anywhere and at anytime. Speech recognition technology has made major strides in the past decade, however, its applications were limited to telecommunications. This research makes an impressive effort to apply SRT to multimedia, especially in the field of computer graphics.

Applications of Speech Recognition Technology

SRT is rapidly gaining acceptance due to the recent advancements in Computer technology. Clearly, the future of computing will be with voice input. Having given our computers both oral and aural abilities, we have been able to produce innumerable computer applications that further enhance our productivity; not only can we communicate quickly and easily with each other from a distance, our computers can converse with us over the telephone and give us updated information using a group of activities collectively referred to as *voice processing.*

SRT plays a vital role in almost every field. An ultimate goal of speech recognition is to enable natural communication between men and machines. The applications can be classified into following two categories.

1. Telecommunications:

   (a) Automatic information service.

   (b) Voice dialing, free hand phones.

   (c) Command and control of database consultation service.

   (d) Voice banking.

   (e) Booking/purchase services on phone.

2. Other fields:

   (a) Form fill-in by voice.

   (b) Text and data encoding.

   (c) Command and control of manufacturing process.

   (d) Help to the disabled.

   (e) Learning of a language.

   (f) Voice typewriter.

Objective and the Scope of the Thesis

The objective of the thesis is to design an efficient and real-time voice interactive computer graphics and games system(VICGG). The thesis work primarily focuses how Speech Recognition Technology can be applied for designing this system. The thesis work also explains and highlights various problems involved in designing a real-time recognition system. Various experiments performed along with their merits and demerits in the process of designing the VICGG have also been discussed. It is also emphasized how various concepts like filters, noise-gates, windowing functions etc., are used in the process of recognition.

Past Work

Considerable research has been done in the area of voice recognition, but not much research has been done in the field of applying these concepts to multimedia and in designing Interactive Computer Graphics Systems.

Organization of the Thesis

The thesis is organized into four chapters. Chapter 1 gives the introduction. Chapter 2 discusses what has been done in the past, regarding the applications of SRT. Chapter 3 discusses the various experiments performed to accomplish the best recognition algorithm. It also discusses the merits and demerits of various techniques used in the recognition process. Chapter 4 deals with explaining how the recognition system, implemented in Chapter 3, can be applied to control a 3-dimensional graphics module to demonstrate VICGG. The conclusions and the results are summed up in Chapter 5. Suggestions for further research are made in this chapter.

CHAPTER 2

LITERATURE SURVEY

Ever since the early days of modern computer technology there has been a significant interest in emulating the intelligent behavior of human beings by computers. Such behavior involves human perception of the environment which includes the ability to recognize and perceive various 1, 2 and, 3 dimensional patterns. Especially the perception of acoustic and visual patterns have attracted significant interest in the past 2 decades. This chapter gives a brief explanation of the following:

1. Evolution of speech recognition technology.
2. Past and present research in applications of speech recognition technology.

Evolution of Speech Recognition Technology(SRT)

The problem of speech recognition has been actively studied since the 1950's. The performance of speech recognizers has improved dramatically due to recent advances in speech science and computer technology. The next two subsections enumerate the dimensions of difficulties in speech recognition technology and how latest technologies have solved them respectively.

Dimensions of difficulty in Speech Recognition Technology

1. Isolated, connected, and continuous speech
2. Vocabulary size

3. Task and Language constraints

4. Speaker dependence or independence

5. Acoustic ambiguity, confusability

6. Environmental noise

Techniques of Speech Recognition

Any recognition system is a pattern recognition system. Any pattern recognition system can be in one of two possible modes of operation:

1. Analysis mode (learning)

2. Recognition mode (application)

   The techniques for computer based pattern recognition can be grouped into 4 groups as follows:

   (a) Statistical.

   (b) Structural.

   (c) Artificial intelligence based.

   (d) Artificial neural networks.

Detailed explanation of Speech Recognition techniques

1. Statistical:

   Depending upon the application, the Statistical methods are based on a pattern description where each sample is represented by a n-dimensional feature vector. Each feature is a numerical measure of a characteristic of the sample, e.g. , its length or color. During the analysis mode the features extraction is done.

Recognition is the process, where it is determined in which region the unknown sample falls and subsequent to which pattern class it belongs. The recognition module of present thesis falls into statistical model.

2. Structural:

   Structural methods make explicit use of knowledge about the structure of the object. It utilize a pattern description where each pattern is divided into sub-patterns called primitives. Each primitive has no direct relation to the structure of the pattern. A pattern is represented by knowledge about how sub-patterns must be combined to make up the entire pattern, and how sub-patterns relate to each other. *Hidden Markov models or HMM's* are examples of Structural methods. HMM's are based on sound probabilistic framework.

3. Artificial intelligence based:

   In methods based on artificial intelligence, numerical, structural, and other types of information are used to describe the pattern. The analysis mode consists of, determining the appropriate discriminant information, specifying each pattern class as a kind of abstract concept, and specifying the conceptual relations between the entities involved in the system. The recognition phase involves taking observations relating to an object and using an inference engine to determine whether or not a pattern can be a specific instance of an abstract concept.

4. Artificial neural networks:

   The artificial neural net approach has many similarities with statistical pattern recognition concerning both the data representation and the classification principles. The practical implementation is, however, very different. The analysis mode involves the configuration of a network of artificial neurons and the train-

ing of the net to determine how the individual neurons can effect each other. The recognition mode involves sending data through the net and evaluating which class got the highest score.

Past and Present Research in Applications of SRT

Not much research has been done in applying SRT in multimedia applications. Thats where this thesis comes into picture. Considerable research has been done in applying SRT in other fields like telecommunications, automatic information service, command and control of manufacturing process, help to the disabled.

Research in Interactive Voice Technology for Telecommunications

Kamm[1] explains how speech recognition for directory assistance applications is of high importance. Telephone companies in America alone handle 6 billion Directory Assistance calls each year. This paper describes how various recognition algorithms, their accuracies and elimination of redundant information is important in these applications.

There are numerous applications of Voice Technology in the field of telecommunications. Information retrieval plays a vital role in telecommunications. Judith[2] explains how to develop a total system solution to interact with users and assist them to search and retrieve information.

Speech recognition techniques are most useful when used over phone. The research paper Toshihiro[3] explains the results of telephone speech recognition hardware for a voice activated home banking system based on a client-server network configuration.

The research paper Yasuhiro[4] aims at establishing fundamental technologies for spontaneous speech translation in telecommunications.

Research in Other Fields

WebGALAXY: Beyond Point and Click - A Conversational Interface to a Browser is a project done at Spoken Language Systems Group MIT Laboratory for Computer Science. This paper Lau[5] presents WebGALAXY, a flexible multi-modal user interface system that allows wide access to selected information on the World Wide Web (WWW) by integrating spoken and typed natural language queries and hypertext navigation.

A considerable amount of research has been done in Multi-lingual Human-Computer Interactions. Glass[6] and Zue[7] are two valuable papers published in this field. They describe how human language plays an important role in human computer systems.

The other applications of voice recognition technology include applying it to control a wheel chair designed for disabled. The research paper Mazo[8] describes how an electric wheel chair is controlled by voice commands.

A similar paper Beattie[9] focuses on designing an intelligent wheelchair to assist disabled or elderly people.

CHAPTER 3

SPEECH RECOGNITION

This chapter is a core chapter in the thesis. It covers the actual work done and is organized as follows:

1. Step1: Explanation of Voice Interactive Computer Graphics (VICGG) system in brief.

2. Step2: Explanation of VICGG system in detail including *Analysis phase*, *Recognition* phase and finally the *Graphics module.*

3. Step3: Detailed explanation of concepts and fundamentals behind the actual design of VICGG system.

4. Step4: Detailed explanation of three isolated word recognition experiments performed is given. This explanation assumes that we understand all the basic concepts explained in step-3. The new concepts like noise-gate, hamming window and filters which actually distinguish these experiments will be explained along with the experiments. The relative merits and demerits have also been discussed.

Modules of the Voice Interactive Computer Graphics System

The VICGG system implemented can be divided into two modules as follows.

1. Pattern recognition system for isolated word recognition.

2. A graphics module containing a set of 3-dimensional Dinosaur animations. These animations are totally controlled by the recognition module.

Detailed Explanation of the Pattern Recognition System

The pattern recognition system implemented has two phases.

1. Analysis phase(learning)

2. Recognition phase(application).

Detailed Explanation of the Analysis Phase

Introduction to Analysis

The importance of the analysis phase lies in the fact that it is the heart of the pattern recognition system. In the analysis phase test patterns are analyzed in order to learn which information is relevant for a classification. It is essentially the implementation of algorithms which process an acoustic waveform received by a microphone into useful parameters.

Science Behind Analysis and Acoustics of Vowels

Nearly all vowel sounds are produced with vibrating vocal cords. Each time the vocal cords open and close there is a pulse of air from the lungs. In a vowel sound, the air in the vocal tract vibrates at three or four frequencies simultaneously. These frequencies are the resonant frequencies of that particular vocal tract shape. This happens irrespective of the fundamental frequency which is determined by the rate of vibration of the vocal cords. This is the reason why we consider magnitudes corresponding to frequencies between 100 and 3000Hz. The important frequencies which we need for recognition of voice of any person are there in this range. The envelope of the power spectra contains the vocal tract information.

Advantages of Digital Processing in the Frequency Domain

1. Any analogue process has its digital counterpart which is more repeatable, accurate and increasingly cheaper.

2. Frequency domain analysis does not take into consideration about when the actual word is spoken.

3. Greatly flexible and allows a much wider range of useful analysis to be carried out than in analogue processing.

Digital Representation of Speech Signals

Bandwidth and sampling rate play a vital role in all pattern recognition systems. For voice processing at present a bandwidth of 4 to 5kHz is often enforced. Human voice frequency components above 4kHz are seen to be considerably less.

Sampling rate

In the digital processing the original waveform, such as speech, is a continuous analog quantity, but since a computer carries out its operations on discrete quantities, the analog quantity has to be sampled into discrete by an analogue to digital converter.

Relationship Between Sampling rate and Bandwidth

There is an important relationship between sampling rate and signal bandwidth. The original signal can be recovered from its sampled version only if the sampling rate is at least twice the bandwidth frequency. This frequency is called Nyquist frequency. In our case the sampling rate is 11025 samps/sec for a 4 kHz bandwidth.

Discrete Fourier Transform

Discrete Fourier Transform (DFT) is the basic transform used in frequency-domain analyses of discrete-time signals such as speech.

$$X(k) = \Sigma_{n=0}^{N-1} x_n e^{-i2\pi nk/N}, \ k = 0, \ldots, N-1$$

This equation defines an algorithm that takes an array of N complex numbers (or equivalently an array of N real numbers and N imaginary numbers) and returns an array of N complex numbers.

Fast Fourier Transform

FFT is a computational method for evaluating DFT in a faster and more efficient manner than by evaluating it directly. A simple interpretation requires $O(N^2)$ operations. The FFT requires $O(N \log N)$ if N is a factor of two. In general, FFT style techniques can be used for any N - firstly N is decomposed into prime factors and then a recursive "butterfly" operation is performed for each factor.

The "Decimation in Time" algorithm (Cooley and Turkey, 1965): Start with the top level Fourier transform described by:

$$X_1(k) = \Sigma_{n=0}^{N-1} x_n e^{-i2\pi nk/N}, \ k = 0, \ldots, N-1$$

$$X_1(k) = \Sigma_{n=0}^{N/2-1} x_{2n} e^{-i2\pi(2n)k/N} + \Sigma_{n=0}^{N/2-1} x_{2n+1} e^{-i2\pi(2n+1)k/N}$$

$$= \Sigma_{n=0}^{N/2-1} x_{2n} e^{-i2\pi(2n)k/N} + e^{-i2\pi k/N} \Sigma_{n=0}^{N/2-1} x_{2n+1} e^{-i2\pi(2n+1)k/N}$$

$$= \Sigma_{n=0}^{N/2-1} x_{2n} e^{-i2\pi nk/(N/2)} + e^{-i2\pi k/N} \Sigma_{n=0}^{N/2-1} x_{2n+1} e^{-i2\pi nk/(N/2)}$$

$$= x_{11}(k) + e^{-i2\pi k/N} x_{12}(k)$$

Now write $X_1(k)$ as a summation of odd terms and even terms:

That is, the original Fourier transform has been written in terms of two Fourier transforms operating on the odd and even portions of the data.

This recursion can be continued until the trivial case of one point is reached. That is, the original Fourier transform has been written in terms of two Fourier transforms operating on the odd and even portions of the data.

This recursion can be continued until the trivial case of one point is reached.

Output of Analysis Phase

After the analysis phase we arrive at a value or values which represent the characteristics of the following.

1. The persons vocal tract who spoke the word.

2. The actual word spoken(phonetics).

Recognition Phase

The input to the recognition phase is the output of the analysis phase . Recognition phase simply compares the feature vectors in the database with the feature vector of the word or words to be recognized. This phase is identical in all the experiments that were performed. We prepare a database of words in the beginning before the recognition process starts. After the feature vector of the word to be recognized is compared with all the feature vectors of the words in the database we get a best match.

Speech Recognition Experiments Performed

The following are the three isolated pattern recognition experiments performed.

Experiment1 using Filters

1. Step1: Recording the amplitude values at a sampling rate of 11025 samps/sec for 2 secs. This recording includes actual spoken word and also noise. After this step we have 22050 amplitude values.
2. Step2: We perform filtering to reduce the effect of the noise.
3. Step3:The filtered values from step2 are ready for frequency domain analysis.
4. Step4:We enter the pattern matching phase.

Detailed Explanation

1. Step1: A word is spoken into the microphone anytime in a 2 seconds duration. It does not matter when the actual word is spoken because we do frequency domain analysis.

2. Step2: The amplitudes thus obtained contain amplitudes corresponding to noise in the room which are undesirable. Filters are needed to reduce the high frequencies corresponding to noise. The following are the two filters used in this experiment.

   (a) Averaging filter.

   (b) Bartlett filter.

   (a) Averaging filter: We explain Averaging filter using following steps.

      i. Step1

         We choose a filter size equal to "n". The value depends upon the amount of filtering effect needed.

      ii. Step2

         For every amplitude in the signal we consider "n" amplitude values on left and right and add them and finally divide by the number of amplitudes under consideration. By doing this we nullify undesirable amplitudes corresponding to noise.

      iii. Step3

         For the beginning and end of the signal we reflect some values from right or left respectively to compensate. The concept of this filter is graphically illustrated below

      iv. Step4

         Observation: For n between 3 and 9 the recognition process is good.

Graphs Illustrating Average Filter

The concept of using an average filter is demonstrated in amplitude domain as follows.

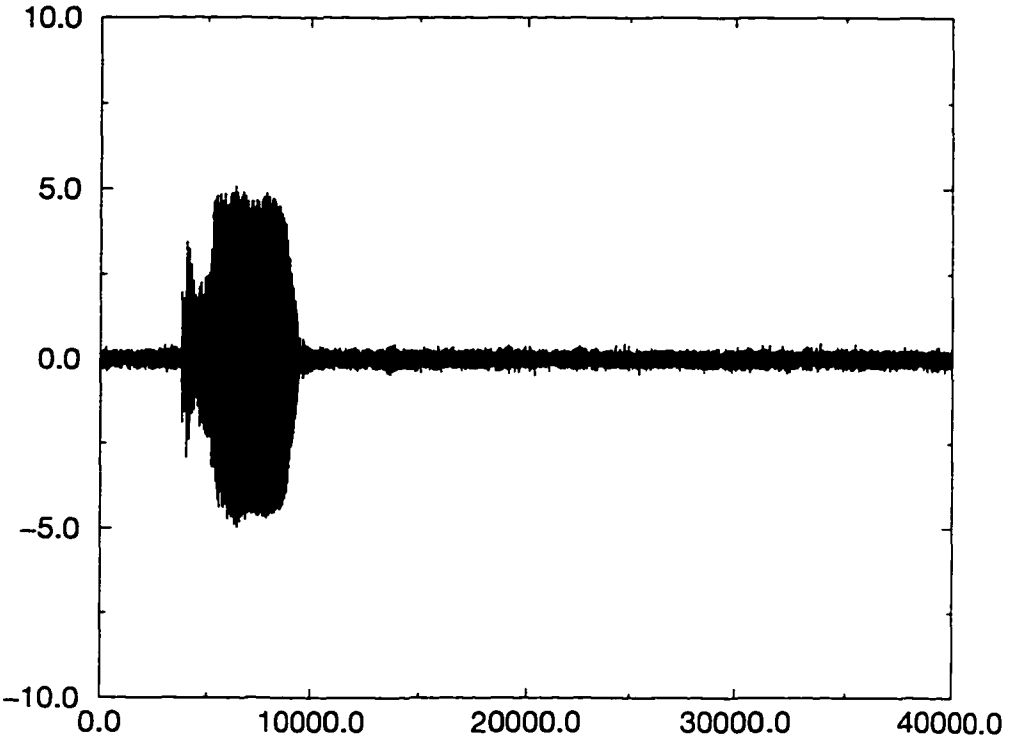


Figure 3.1: Amplitude plot of word "HEL LO" before applying Average Filter .

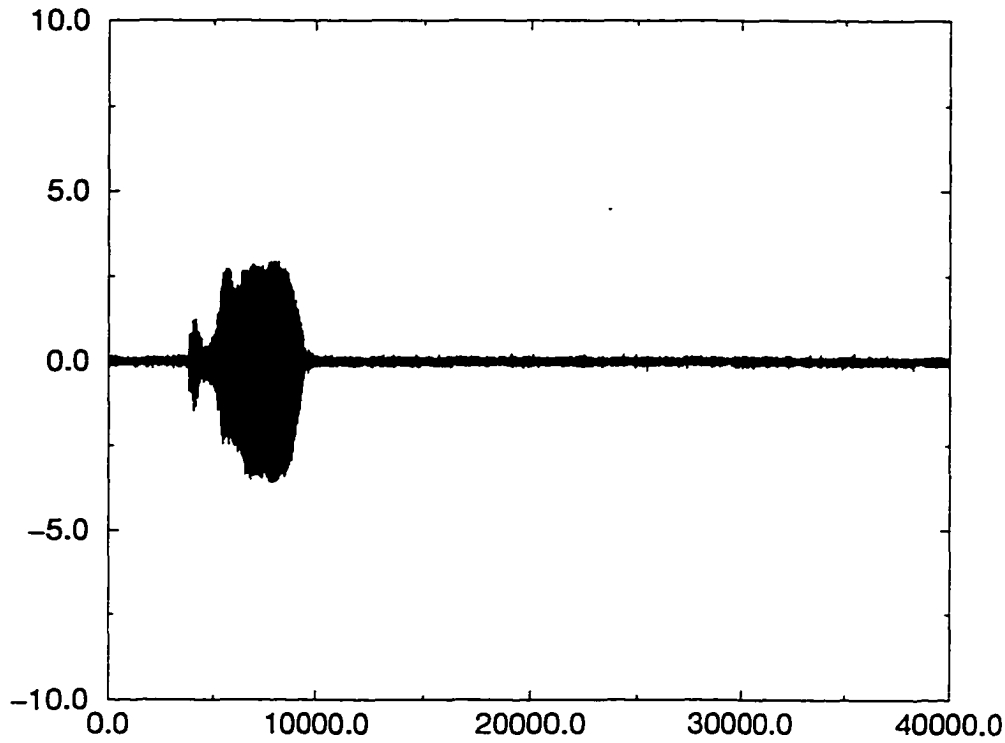


Figure 3.2: Amplitude plot of word "HELLO" after applying Average Filter .

(b) Bartlett Filter: We explain Bartlett Filter using following steps.

i. Step1: We choose a filter size equal to "n". The value depends upon the amount of filtering effect needed.

ii. Step2: We calculate a vector of window values using the Bartlett equation as follows.

Bartlett Windowing Function :

$$w(n) = \begin{cases} \frac{2n}{N-1} & ; 0 \leq n \leq \frac{N-1}{2} \\ 2 - \frac{2n}{N-1} & ; \frac{N-1}{2} \leq n \leq N-1 \\ 0 & ; \text{otherwise} \end{cases}$$

iii. Step3: We exactly repeat the process of step-3 in averaging filter with a small change. Before we add the amplitudes we multiply each amplitude by the corresponding window value obtained in the step-2. We do this to change the shape of the filter. The concept of this filter is graphically illustrated below.

Graphs illustrating Bartlett Filter

The concept of using an Bartlett filter is demonstrated in amplitude domain as follows.The following are the amplitude plots of the word "Hello" before and after applying the filter.

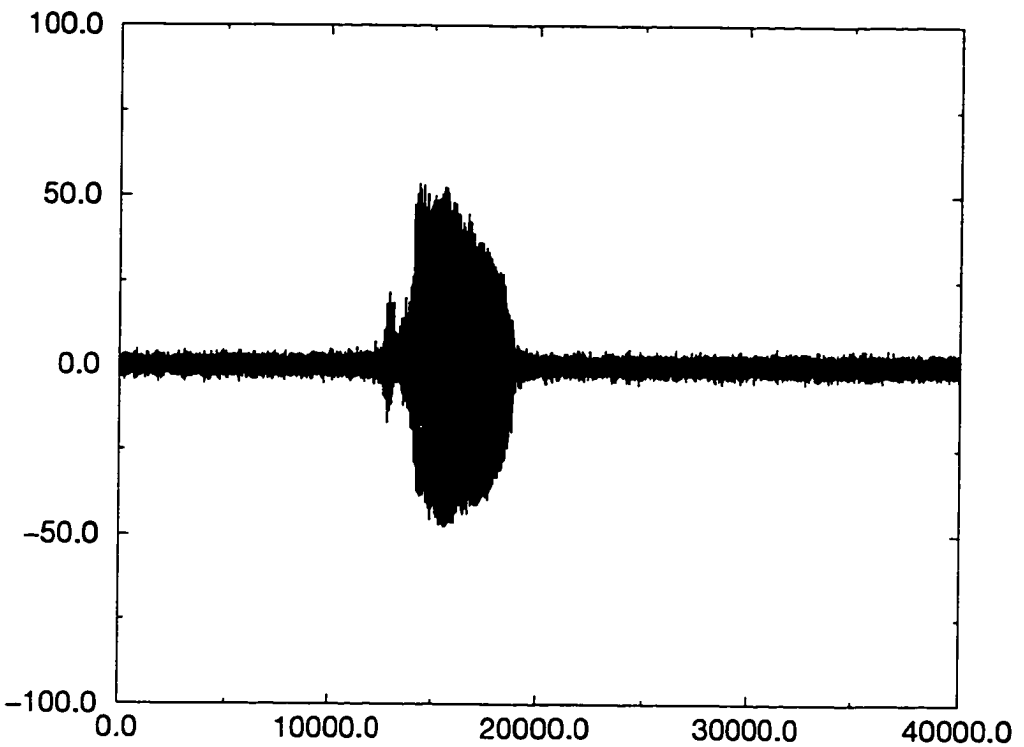


Figure 3.3: Amplitude plot of word "HELLO" before applying Bartlett Filter .

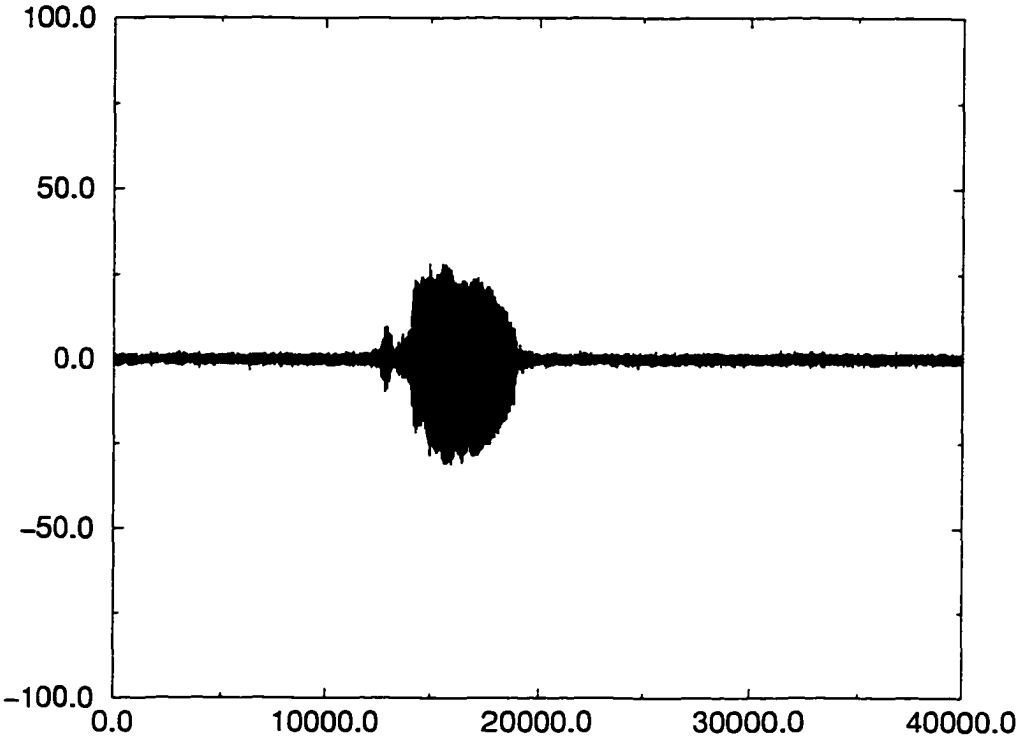


Figure 3.4: Amplitude plot of word "HELLO" after applying Bartlett Filter .

iv. Step4:

Observation:- For n between 3 and 5 inclusive the recognition process is good. This filtering effect is better than averaging filter.

3. Step3: We send these values to a FAST FOURIER TRANSFORM (FFT) algorithm, whose size is 32768, because the least value which is a multiple of 2 and which is greater than 22050 is 32768. As a result of this we get 16384

frequencies in the range 0-5500.

4. Step4: We collect the magnitudes corresponding to the frequencies between 100Hz-3000Hz in an array. This is what we store for comparison purpose.

Merits and demerits

1. The Recognition process is not very accurate.
2. The Recognition process is not real-time.
3. The Recognition process varies with the filter being used.

Experiment2 using Noise-gate

1. Step1: Recording the amplitude values at a sampling rate of 11025 samps/sec for 2 secs. This recording includes actual spoken word and also noise. After this step we have 22050 amplitude values.
2. Step2: We send these amplitude values through a noise gate to remove the noise from the signal and find the beginning and end of the speech signal. This is an efficient method because we do not eliminate some high frequencies which are important for recognition process.
3. Step3: Amplitude normalization.
4. Step4: Frequency domain analysis .
5. Step5: Pattern matching phase.

Detailed Explanation

1. Step1: A word is spoken into the microphone anytime between two seconds duration. It does not matter when the actual word is spoken because we do frequency domain analysis.

2. Step2: The amplitudes thus obtained contain amplitudes corresponding to noise in the room which are undesirable. We find the the beginning and end of the speech signal using a noise-gate. A noise-gate is designed by establishing a *Threshold value* which is basically the noise level of the room. Nothing is recorded when a word is not spoken and noise gate does not open. When we speak the gate opens and all the amplitudes greater than threshold value are recorded ie. . speech signal. We also have a *delay factor* which means that the gate remains open for a short while after the signal goes below the threshold, waiting for a possible raise in the signal immediately. This often happens for words like "it". After this process is finished we store amplitudes (corresponding to speech) and zeros (corresponding to noise) in an array. The principle of noise-gate is graphically illustrated below.

3. Step3: We do amplitude normalization to avoid disparities in the loudness of speech. As a result, it does not matter even if the speaker speaks the word louder or slower. We basically stretch the signal depending upon the maximum amplitude range.

4. Step4: We send these values to an FFT of size 32768 because the least value which is a multiple of 2 and which is greater than 22050 is 32768. As a result of this we get 16384 frequencies in the range 0-5500. The only difference between this and previous experiment is that instead of sending filtered noise values we send zeros.

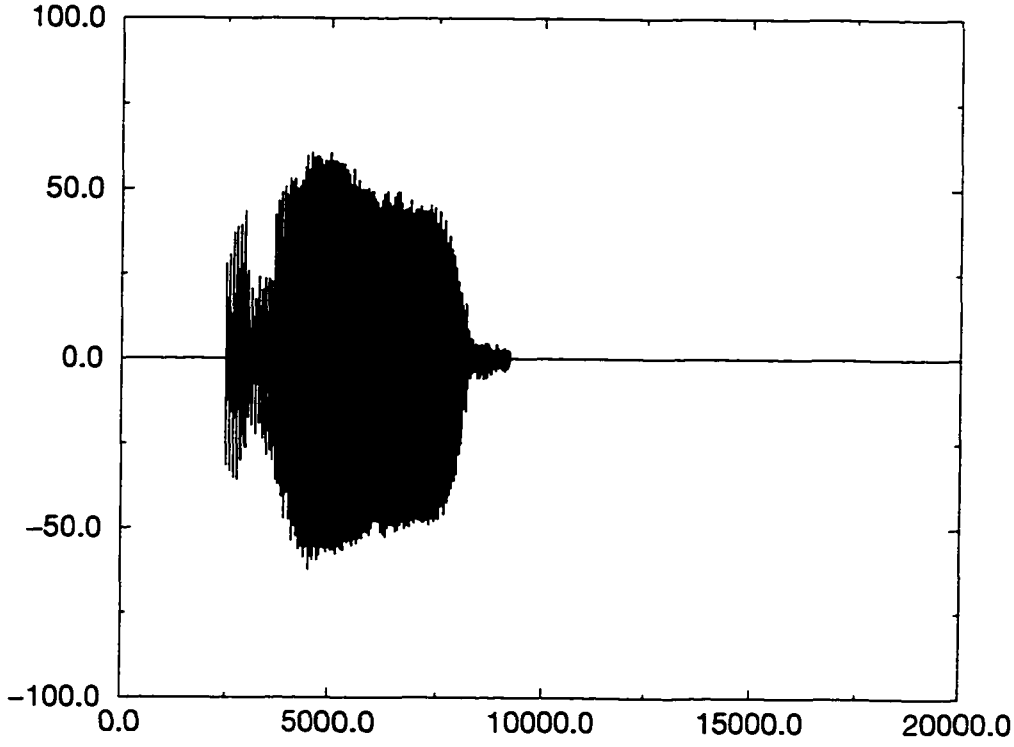


Figure 3.5: Amplitude plot of word "HELLO"after applying Noisegate .

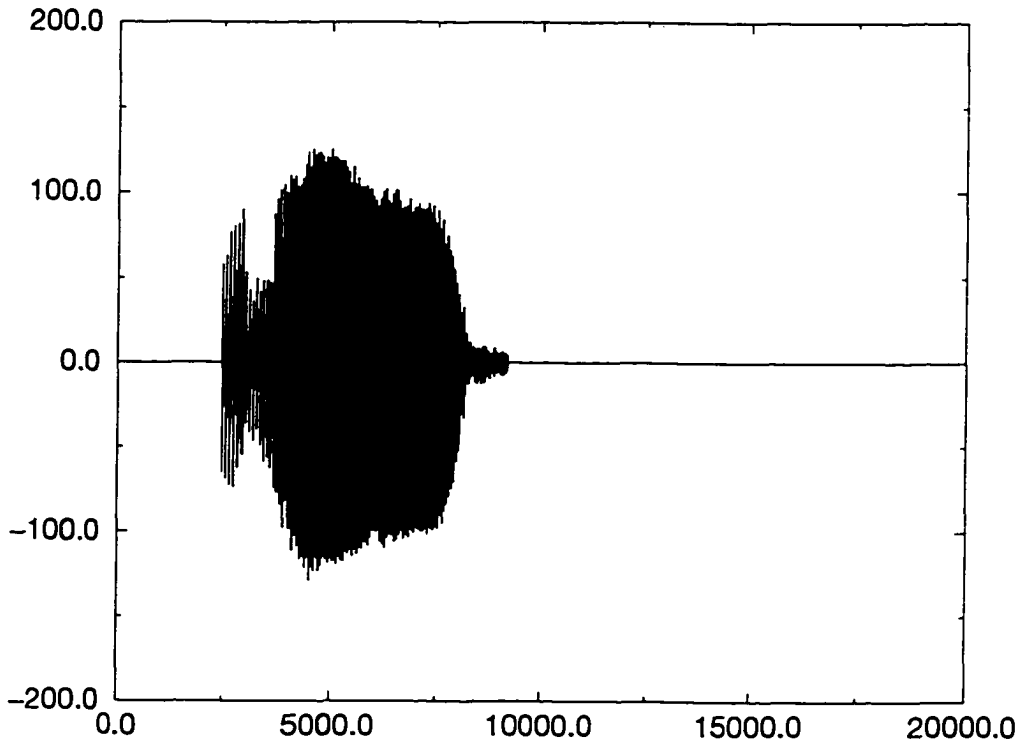


Figure 3.6: Amplitude plot of word "HELLO" with noise gate and normalization .

5. Step4:

   We collect the magnitudes corresponding to the frequencies between 100Hz-3000Hz in an array. This is what we store for comparison purpose.

Merits and Demerits

1. The recognition process is accurate when compared to previous experiment.
2. The recognition process is slow and not real-time because of the huge FFT size.
3. The recognition process varies with noise level in the room. Threshold adjustment must be done before we run the application.

Experiment3 using Short-Time Fourier Analysis

Before we start on a journey through the steps of experiment3 we shall enumerate and explain the concepts behind this experiment. This experiment has been proven to be much more accurate, consistent and real-time.

1. Short-Time Fourier Analysis:

   Speech waveform occupies a bandwidth of about 5kHz and that it is time-varying but relatively slowly, at the articulatory rate. Short-Time Fourier analysis deals with applying Fourier analysis to finite-length, time-varying sequences, such as speech. Short-Time analysis depends on the **windowing** of the speech signal and its properties. This is explained in the next section.

2. Windowing and its importance:

   For speech processing we want to assume the signal is short-time stationary and perform a Fourier transform on these small blocks. The solution is to multiply the signal by a window function . The possible candidates are a

(a) Rectangular window:

$$w(n) = \begin{cases} 1 & ; 0 \leq n \leq N-1 \\ 0 & ; \text{otherwise} \end{cases}$$

Which is an abrupt function in the time domain .

(b) Hamming window:

$$w(n) = \begin{cases} 0.54 - 0.46\cos(2\pi\frac{n}{N-1}) & ; 0 \leq n \leq N-1 \\ 0 & ; \text{otherwise} \end{cases}$$

Which is a smoother raised-cosine function in the domain.

(c) Rectangular window vs Hamming window:

Rectangular windowing results in discontinuities at the ends, which further results in **leakage** between adjacent harmonics. The spectrum for the Hamming window is much smoother because of its sharper cut off and is commonly used for spectral analysis.

Steps in Experiment3

1. Step1: Recording the amplitude values at a sampling rate of 11025 samps/sec for 2 secs. This recording includes actual spoken word and also noise. After this step we have 22050 amplitude values.

2. Step2: The amplitudes thus obtained contain amplitudes corresponding to noise in the room which are undesirable. We find the the beginning and end of the signal using a noisegate. The noisegate is explained in the previous experiment.

3. Step3: We analyze short pieces of the speech signal by first dividing it into **n** **(14 in our case)** number of divisions and then applying hamming window of size **N (1024 in our case)**. We obtain 14 sets of each 1024 values by just moving the hamming window by number of divisions (14 in our case). It may be observed that we have overlapping amplitude values.

Graphs Illustrating Hamming Window

The concept of using an Hamming Window is demonstrated in amplitude domain as follows.

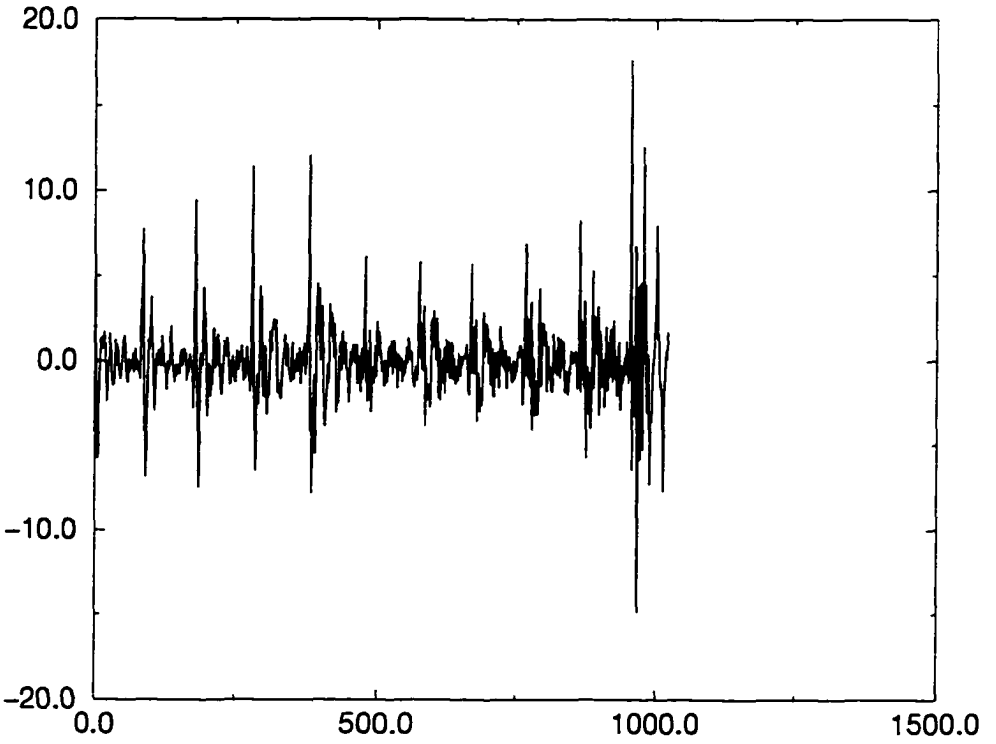


Figure 3.7: Amplitude plot of word "HELLO" before applying Hamming Window.

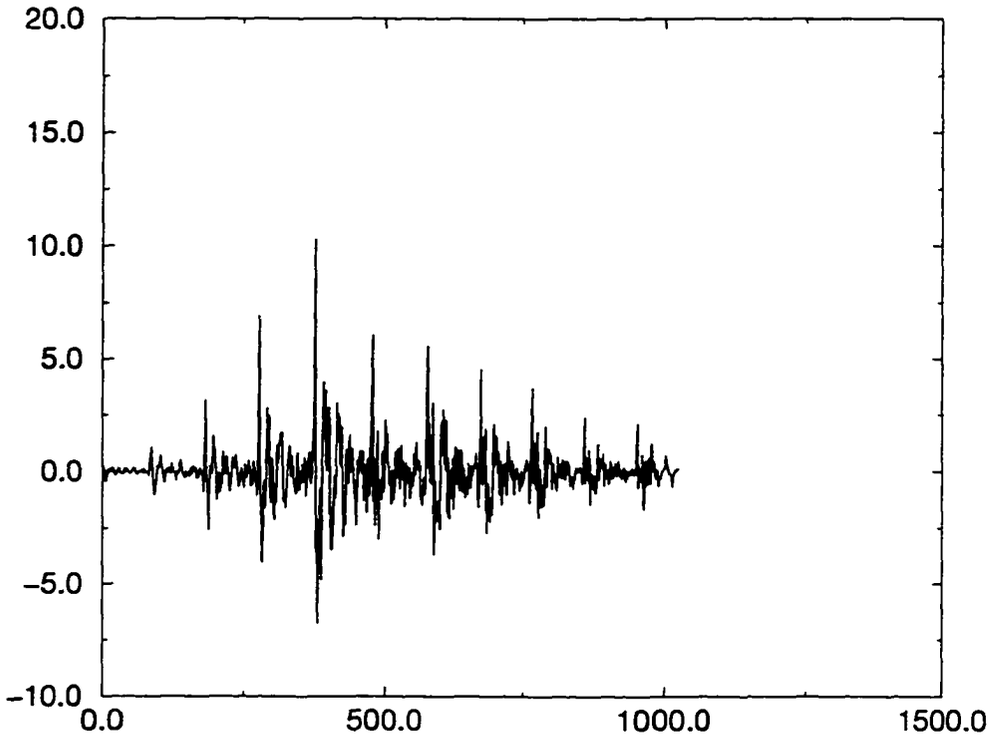


Figure 3.8: Amplitude plot of word "HELLO"after applying Hamming Window.

4. Step4: We send each of these sets of 1024 values to an FFT of size 1024 at a sampling rate 11025 samps/sec.

Graph Illustrating FFT

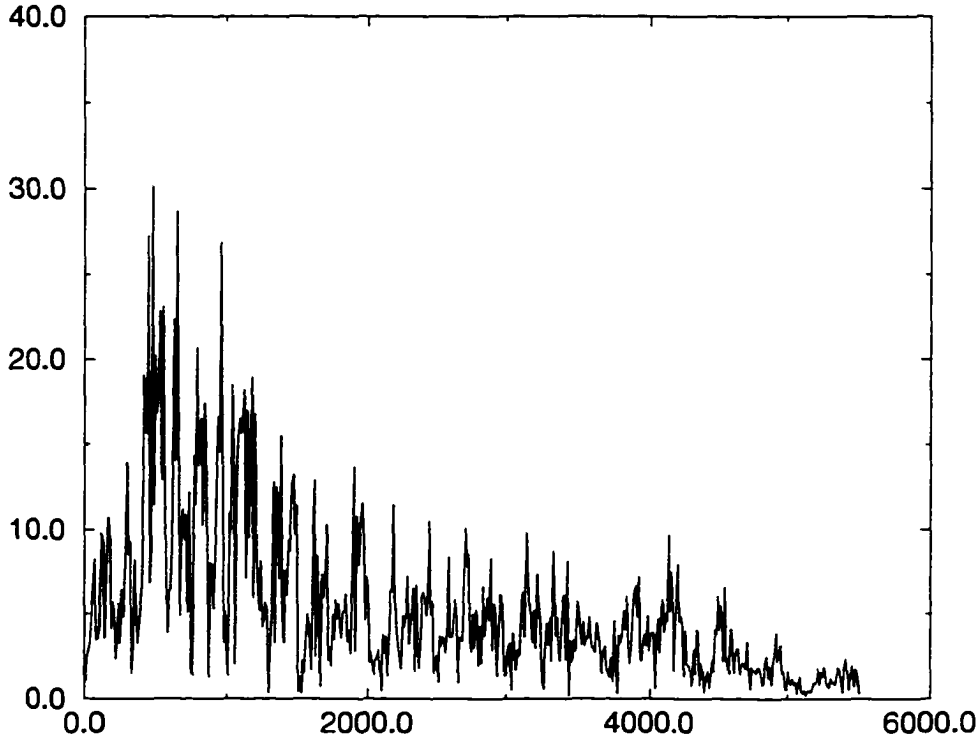


Figure 3.9: Frequency domain analysis of word "HELLO".

5. Step5: We again split the frequency domain of each set into 14 banks between 100-3000Hz frequency and find average of frequencies in each bank. So we get 14 values corresponding to each set of 1024 values. So we arrive at a vector of size 14 * 14 =196. This vector is what we use for pattern matching purpose.

6. Step6: We enter the pattern matching phase.

Merits and Demerits

1. The recognition process is accurate when compared to previous experiment because we retain time resolution .

2. The recognition process is real-time because we do not process amplitudes corresponding to noise. It is also fast because we use a small FFT of size 1024.

3. Threshold adjustment must be done before we run the application.

4. The Recognition process is accurate for a vocabulary of moderate size.

CHAPTER 4

VOICE RECOGNITION AND GRAPHICS MODULE

This chapter explains how the recognition module can be linked to a corresponding animation in the graphics module. Each of the three experiments along with its graphics module is explained.

Experiment1 and its Graphics Module

The following steps enumerated below describes how the whole application works.

1. Step1: As shown in the flow chart, we start our journey by creating a database of words we need. When we run the program described in the experiment it prompts the user to decide on updating the database or proceed with recognition phase.

2. Step2: If the answer to the above question is "no" the control is transferred to recognition else we proceed to step3.

3. Step3: We collect amplitudes in a buffer after digitization.

4. Step4: We apply Bartlett filter to reduce the noise.

5. Step5: We send the filtered values to a FFT for frequency domain analysis.

6. Step6: We store the power values in a buffer corresponding to a frequency range 100-3000Hz.

7. Step7: We store the vector obtained in the previous step with its associated word in a separate file for future reference and transfer control to step1 to check for the choice of updating or going to recognition.

8. Step8: If the answer to the question to run the recognition phase is "yes", we proceed to the next step else we exit the application.

9. Step9: We now repeat steps 3, 4, 5, 6, after we speak the word to be recognized.

10. Step10: We now compare the vector corresponding to the word to be recognized with all the vectors stored in the database. We basically find the linear distance between the vectors and find the minimum of all the distances. The vector with associated minimum distance is the recognized word. We write this word into a file.

11. Step11: A check function implemented in the graphics module repeatedly checks the file to find if a new word is written. Corresponding to the word read from the file, the control is transferred to an animation that is related to the word.

Illustration of Experiment1 using a flowchart

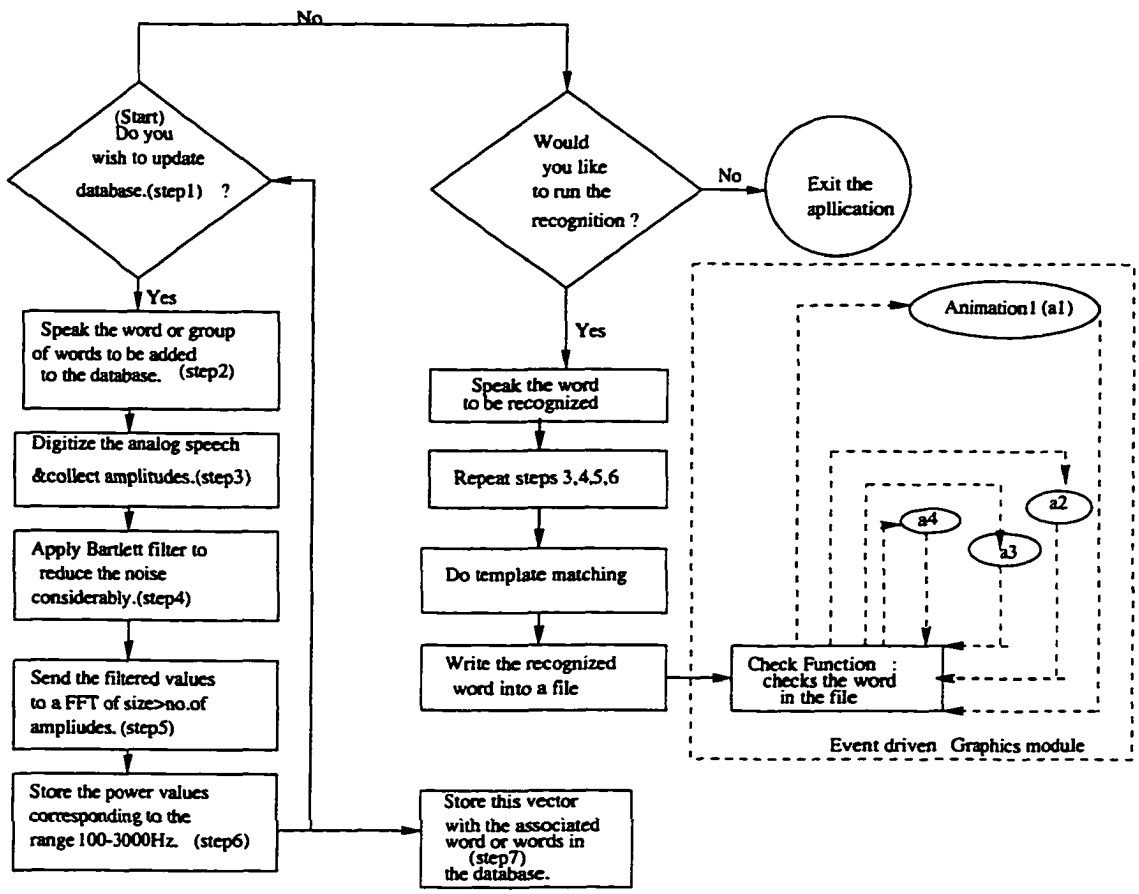


Figure 4.1: Algorithm 1 Flowchart

Experiment2 and its Graphics Module

The following steps enumerated below describes how the whole application works.

1. Step1: As shown in the flow chart we start our journey by creating a database of words we need. When we run the program described in the experiment it prompts the user to decide on updating the database or proceed with recognition phase.

2. Step2: If the answer to the above question is "no"the control is transferred to recognition else we proceed to step3.

3. Step3: We collect amplitudes in a buffer after digitization.

4. Step4: We apply a noise-gate to extract the exact speech signal.

5. Step5: We send the value obtained from the previous step and zeroes (corresponding to noise) to a FFT for frequency domain analysis.

6. Step6: We store the power values in a buffer corresponding to frequency range 100-3000Hz.

7. Step7: We store the vector obtained in the previous step with its associated word in a separate file for future reference and transfer control to step1 to check for the choice of updating or going to recognition process.

8. Step8: If the answer to the question to run the recognition phase is "yes" we proceed to the next step else we exit the application.

9. Step9: We now repeat steps 3, 4, 5, 6, after we speak the word to be recognized.

10. Step10: We now compare the vector corresponding to the word to be recognized with all the vectors stored in the database. We basically find the linear distance

between the vectors and find the minimum of all the distances. The vector with associated minimum distance is the recognized word. We write this word into a file.

11. Step11: A check function implemented in the graphics module repeatedly checks the file to find if a new word is written. Corresponding to the word read from the file, the control is transferred to an animation that is related to the word.

Illustration of Experiment2 using a flowchart

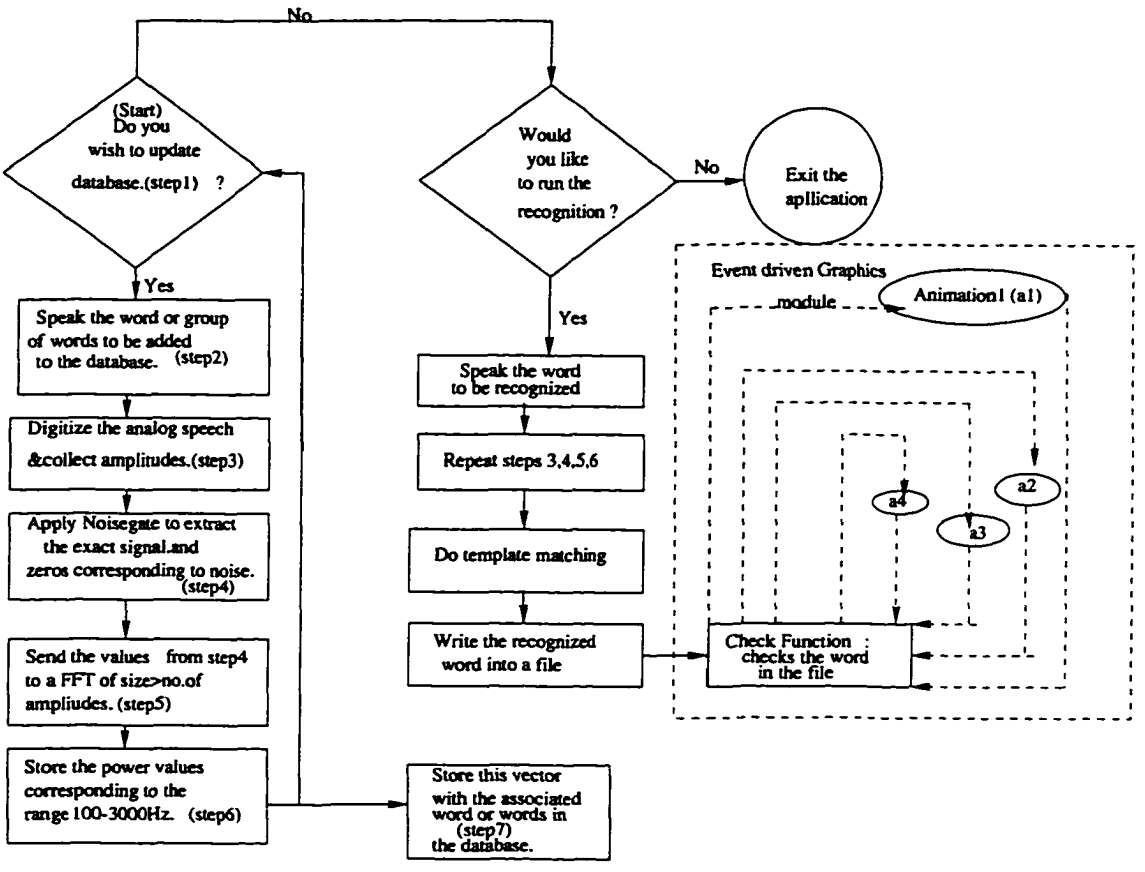


Figure 4.2: Algorithm 2 Flowchart

Experiment3 and its Graphics Module

The following steps enumerated below describes how the whole application works.

1. Step1: As shown in the flow chart we start our journey by creating a database of words we need. When we run the program described in the experiment it prompts the user to decide on updating the database or proceed with recognition phase.

2. Step2: If the answer to the above question is "no"the control is transferred to recognition else we proceed to step3.

3. Step3: We collect amplitudes in a buffer after digitization.

4. Step4: We apply a noise-gate to extract the exact speech signal and split it into 14 pieces.

5. Step5: We apply a hamming window on sets of 1024 values by shifting the window 14 times. it can be noted that we have overlapping windows.

6. Step6: We send 14 sets for frequency domain analysis.

7. Step7: We store the power values in a buffer corresponding to frequency range 100-3000Hz and then split it into 14 slices and then find the average of each slice.

8. Step7: We store the vector of size 14*14 obtained in the previous step with its associated word in a separate file for future reference and transfer control to step1 to check for the choice of updating or going to recognition.

9. Step8: If the answer to the question to run the recognition phase is "yes" we proceed to the next step else we exit the application.

10. Step9: We now repeat steps 3, 4, 5, 6, 7, after we speak the word to be recognized.

11. Step10: We now compare the vector corresponding to the word to be recognized with all the vectors stored in the database. We basically find the linear distance between the vectors and find the minimum of all the distances. The vector with associated minimum distance is the recognized word. We write this word into a file.

12. Step11: A check function implemented in the graphics module repeatedly checks the file to find if a new word is written. Corresponding to the word read from the file, the control is transferred to an animation that is related to the word.

Illustration of Experiment3 using a flowchart

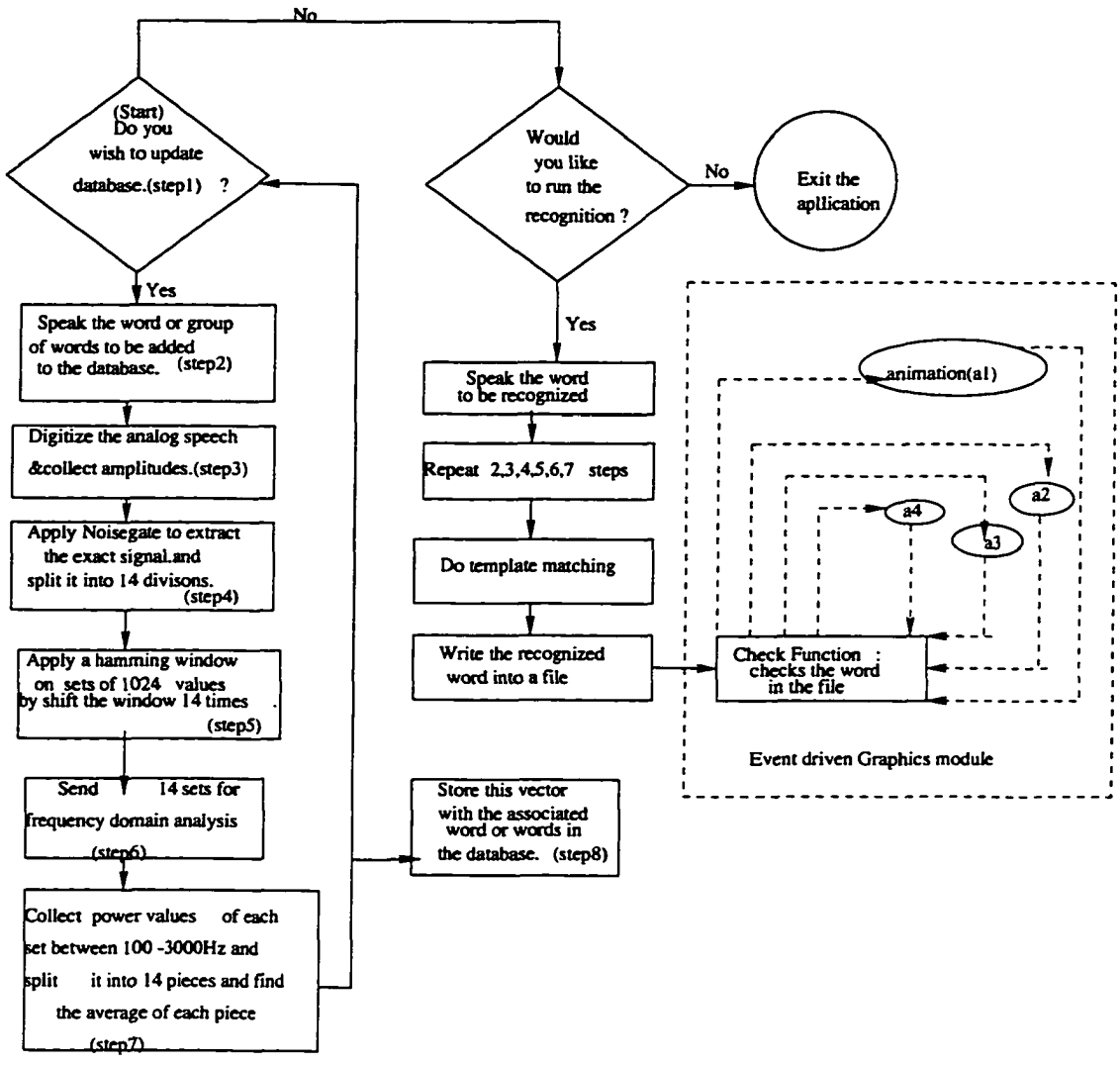


Figure 4.3: Algorithm 3 Flowchart

CHAPTER 5

CONCLUSION

The present thesis demonstrates the fact that the advances in voice recognition, natural language understanding, and other technologies are beginning to make human language access to information and services a reality. As a result of sophisticated Voice Technologies, we need not restrict ourselves in applying it to telecommunications and other information retrieval services, but also to some other fields like multimedia. Very soon we are going to witness this revolution.

Observations and Results

The first recognition algorithm implemented is neither very accurate, nor fast. The reasons are enumerated below

1. It is not very accurate because, when we use filters to reduce noise, they also filter some of the high frequencies which are necessary for recognition process.

2. The recognition process is slow because an FFT algorithm of size 32768 is used to cover all the amplitudes present in a duration of 2 secs. It becomes still slower when we increase the vocabulary size.

In the second algorithm we make an effort to increase the accuracy by finding the beginning and the end of the speech signal using a noise-gate. This algorithm is still not very efficient for the following reasons.

1. It is not very accurate because, when we send the whole speech signal along with some zeros corresponding to noise for frequency domain analysis, we lose time resolution.

2. The recognition process is slow because, an FFT algorithm of size 32768 is used to cover all the amplitudes present in a duration of 2 secs including zeros corresponding to noise.

The third algorithm solves the above problems by using short-time Fourier analysis and bank of filters. This algorithm is both accurate and real-time for the following reasons.

1. It is very accurate because, we retain the time resolution by doing short-time Fourier analysis on the speech signal.

2. Real-time performance is achieved by using an FFT algorithm of size 1024 to cover all the amplitudes present in each slice rather than the entire speech signal.

Future Research

One potential problem in all speech recognition systems is that, they are not accurate. There are several reasons for this inaccuracy. The most important reasons as described in chapter 2 are:

1. Isolated, connected, and continuous speech

2. Vocabulary size

3. Task and Language constraints

4. Speaker dependence or independence

5. Acoustic ambiguity, confusability

6. Environmental noise

One possible solution to some of the above problems is to provide *lip movement* or *gesture recognition* as a supplement to voice interface. Some research has been done in this field.

Human machine interaction by voice and gesture is very important because it emphasizes how gesture can be a good supplement to the recognition process. Jayant[10] explains how voice and gesture represent fundamental and universal modalities in inter-human communication. They also explain how less mature and deployed, gesture recognition by machine is becoming reliable enough to be considered as a serious supplement to the voice interface between humans and machines. Human machine interaction by voice is a well organized reality. As the qualities of our acoustic and visual transducers become increasingly fast and reliable, machine recognition of human gestures will become a reality.

In spite of advances in speech technologies, not much research has been done in applying it to *Multimedia*. Speech technology will play an increasingly important role in the coming Multimedia era.

While we are making steady progress towards a better understanding of the complex relationship between the speech signal and its underlying linguistic forms, there is no denying that our knowledge is still very limited. Speech knowledge can potentially improve recognition algorithms. The improvement may come in the form of better signal representation, more reliable methods to extract phonetically relevant acoustic measures, and perceptually based measures of phonetic similarity.

Algorithms enable us to solve the problems, *Knowledge* enables the algorithms to work better.

APPENDIX A

Source Code For Recognition Module

```
/*
  Nataraj Jeedigunta
  File:main.c
  This is the main program of the recognition module,
  which basicallly calls functions in "database module".
  The function calls are basically for updating the
  database and for comparing database for recognition
  process.
 */

#include <unistd.h>
#include <dmedia/audio.h>     /* audio library header */
#include <sys/types.h> /* for unix open() */
#include <sys/stat.h>  /* for unix open() */
#include <fcntl.h>     /* for unix open() */
#include <signal.h>    /* for signal handler */
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <math.h>
#include "sound.h"
```

```
#include "fft.h"
#include "database.h"

main()
{  char dword[256];
   char *fword;
   FILE *fh;
   printf("\n");
   /* bartletwindow();*/

   InitFFT();
   read_database();

  printf("\n Hai Welcome TO UNLV Voice Recognition Project\n");

   printf("\n If You Like To Update the Database [y/n]?");
   if( question() )
   {
     do
     {
       if( update_database() )
       {
      printf("\n You Have sucessfully updated the Database\n");
         write_database();
       }
       else
```

```
            printf( "\n The database is not updated\n" );

          printf("\n Do You Want to add another word [y/n]?");
        }while( question() );
        print_database();
      }

      printf("\n Do You Like To run recogntion [y/n]?");
      if( question() )
        do
        {
          printf("\n Say the word to  be recognized");

          if( (fword = compare_database()) != NULL )
          {
            sprintf( dword, "%s", fword );
            while( (fh = fopen( "result", "w" )) == NULL );
            fprintf( fh, "%s\n", dword );
            fclose( fh );
          }
          else
            printf( "\n The word was not recognized\n" );

          printf("\n Do You Want to say another word [y/n]?");
        }while( question() );
    }
```

```
/*
Nataraj Jeedigunta
  File:database.h
*/

#ifndef _DATA_H
#define _DATA_H


typedef struct{
   char word[256];
   char filename[256];
}RECORD;

int question();
void read_database();
void write_database();
void print_database();
void add_word( char *word, char *fname );
int find_word( char *word );
int update_database();
int search_database( float *t, int num );
char *compare_database();


#endif
```

```
/*
  Nataraj Jeedigunta
  File:database.c
  This is the database module, which basically has
  functions for reading database,comparing database,
  and also to search database. These functions
  further call functions in sound and FFT modules
  which are required for recording,analysing in
  amplitude domain and frequency domain,and also
  for pattern matching.

 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "database.h"
#include "sound.h"
#include "fft.h"

#define MAX_WORDS   256
#define NUM   DIV*DIV
#define DATABASE_FILE   "words/index.db"

int max_records;
RECORD database[MAX_WORDS];
```

```
float dbfft[FFT_SIZE/2];
float dog[2*DIV][2*DIV];
float *result;
 float gun[500];
extern float z[DIV][FFTSZ];

int question()
{
int ch = getchar();
while( ch != 'y' && ch != 'Y' && ch != 'n' && ch != 'N' )
ch = getchar();
return( ch == 'y' || ch == 'Y' );
}

void read_database()
{
  char line[512];
  int i;
  FILE *ifh = fopen( DATABASE_FILE, "r" );
  if( ifh == NULL )
  {

fprintf(stderr,"Could not find the 'index.db'
database file\a\n" );
    exit( -1 );
  }
```

```
  fgets( line, 512, ifh );
  sscanf( line, "%d", &max_records );
  for( i = 0; i < max_records; i++ )
  {
    fgets( line, 512, ifh );
 sscanf( line, "%s %s", database[i].word,
 database[i].filename );
  }
  fclose( ifh );
}

void write_database()
{
  char line[512];
  int i;
  FILE *ofh = fopen( DATABASE_FILE, "w" );

  fprintf( ofh, "%d\n", max_records );
  for( i = 0; i < max_records; i++ )
fprintf( ofh,"%s %s\n",database[i].word,
database[i].filename);

  fclose( ofh );
}

void print_database()
```

```
{
  char line[512];
  int i;

  printf( "\nThere are %d word in the database\n",
   max_records );
  printf( "----------------------------\n" );
  for( i = 0; i < max_records; i++ )
  printf( "Word: %s \t\tFile: %s\n", database[i].word,
  database[i].filename );
  printf( "----------------------------\n" );
}

void add_word( char *word, char *fname )
{
  strcpy( database[max_records].word, word );
  strcpy( database[max_records].filename, fname );
  max_records++;
}

int find_word( char *word )
{
  int i;
  for( i = 0; i < max_records; i++ )
    if( strcmp( database[max_records].word, word ) == 0 )
       return 1;
```

```
  return 0;
}

int update_database()
{
    char word[256];
    char filename[256];
    FILE *ofh;
    float fnarray[400];/*should be DIV*DIV*/

    printf( "\n Type the word to be added to database: " );
    scanf( "%s", word );
    if( find_word( word ) )
    {
       printf( "\n The word is in the database." );
       printf( "\n Do you want to replace this word [y/n]?" );
       if( !question() )
          return 0;
    }
    printf( "\n Recording..." );

    if( !analyze( fnarray ) )
    {
       printf( "Analisys failed!\n" );
       return 0;
    }
```

```
    sprintf( filename, "words/%s.fft", word );
    ofh = fopen( filename, "w" );

/*    fwrite( &num, sizeof( int ), 1, ofh );*/
    fwrite(fnarray, sizeof( float ), NUM, ofh );

    fclose( ofh );
    add_word( word, filename );
    return 1;
}

int analyze( float *fnarray )
{
    float fag[NSAMPS];
    float smog[NSAMPS];
    float *final;
    int sigsize=0;
    int id, gt, ufo, ifo;

    record(fag,&sigsize);
  /*returns the exact spoken signal values*/
    if(sigsize ==0)
      return 0;

    padding(smog,fag,sigsize);
```

```
    gt=sigsize/DIV;

    ufo = 0;
    for(id=0;id<DIV;id++)
    {
        hmwindow( &smog[gt*id] );
        final = send_to_FFT(&smog[gt*id]);
         /*14 vals are returned*/
        if( final == NULL )
       return 0;

    for(ifo=0;ifo<DIV;ifo++)
    {
       fnarray[ufo]=final[ifo];
    /*copy the 14 element slices to a big aay*/
           ufo++;
    }
    }
    return 1;
}

char *compare_database()
{
    int i;
    char word[256];
    char filename[256];
```

```
    FILE *ofh;
    float fnarray[400];
/*should be DIV*DIV*/

    printf( "\n Recording the word..." );


    if( !analyze( fnarray ) )
    {
       printf( "Analisys failed!\n" );
       return NULL;
    }

i = search_database( fnarray, NUM );
printf( "\n The recognized word is: %s\n", database[i].word );
return database[i].word;

}

int search_database( float *t, int num )
{
  int j, i = 0, min_index = 0;
  FILE *ifh;
  float min, sum;

  printf( "\n\n" );
```

```
   for( j = 0; j < max_records; j++ )
   {
      printf( " Comparing to word: \"%s\"  ",
      database[j].word );
      ifh = fopen( database[j].filename, "rb" );
/*      fread( &n, sizeof( int ), 1, ifh ); */
      fread( dbfft, sizeof( float ),NUM, ifh );
      fclose( ifh );

      sum = 0;
      for( i = 0; i < num; i++ )
         sum += fabs( t[i] - dbfft[i] );

      sum /= (float)num;
      printf( "\n\t--> Match: %f\n", sum );

      if( j == 0 )
      {
         min = sum;
         min_index = 0;
      }
      else if( sum < min )
      {
        min = sum;
        min_index = j;
      }
```

```
    }
    printf( "\n Best Match: %f\n", min );

    return min_index;
}
```

```
/*
Nataraj Jeedigunta
  File:sound.h
*/

#ifndef _SOUND_H
#define _SOUND_H
#define inc 0.90702948;
#define SAMPLE_RATE (11025)
#define NSAMPS (4*11025)

#define DIV  14
#define  FFTSZ  1024

/* these are the files which we call from main.c a different
module from sound.c so declare them here*/

void bartletwindow();
void record(float *fag ,int *sz);
void noisegate(float *fan,float *buf,int *k);
void padding(float *h,float *fag,int sz);
void hmwindow(float *z);
void normalize(float  *buf );
#endif
```

```
/*
  Nataraj Jeedigunta
  File:sound.c
  This is the sound module. This module has
  functions for sound recording and analysis
  in amplitude domain.
 */


#include <unistd.h>
#include <dmedia/audio.h>     /* audio library header */
#include <sys/types.h> /* for unix open() */
#include <sys/stat.h>  /* for unix open() */
#include <fcntl.h>     /* for unix open() */
#include <signal.h>    /* for signal handler */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <memory.h>
#include "sound.h"
unsigned int nwritten;
int nread; float  *fan;
float value;
float u;
int f,m,j,l,pj,p;
int k;
```

```
 int caught_sigint =  0;
int i=0;
int p=0;
int terminate;/* completion flag */
ALconfig config;/* port configuration structure  */
ALport inport;
ALport outport;

#define HALF_W  6
#define THRESHOLD   5
#define DELAY       (SAMPLE_RATE*.1)
#define PI       22/7
#define blk (HALF_W*2+1)
float buf[NSAMPS];
float window[blk];
float runbuf[NSAMPS];
float b[NSAMPS];
float sum;
long ji=0;
float orig[NSAMPS];
FILE *fp;
FILE  *fp2;
FILE  *fp3;
```

```
static  void catch_sigint();/*encapsulated ie can be
                              called from only this module*/
void catch_sigint()
{
  caught_sigint=1;
  fputs("Got SIGINT\n", stderr);
}

void record(float *fan ,int *sz)
{

  int n,r,q;


   setbuf(stdout, 0);
  config = ALnewconfig();


 /*  If nondefault values
are needed create a config object set them;
the defaultThe default ALconfig has:

a buffer size of 100,000 samples

stereo data
```

```
a two's complement sample format

a 16-bit sample width
          */



ALsetsampfmt ( config,AL_SAMPFMT_FLOAT);
ALsetfloatmax(config, 128.0);
/*VALS RAMGE FROM -10TO10 FLOAT*/
ALsetqueuesize(config, NSAMPS);
/* 4 sec of MONO */
/* ALsetwidth(config, AL_SAMPLE_32);
/* 32-bit  samples  */
ALsetchannels(config, AL_MONO);
 /* MONO port*/
    {
     signed long NewParams[2];
     NewParams[0] = AL_INPUT_RATE;
     NewParams[1] = SAMPLE_RATE;

 if (ALsetparams(AL_DEFAULT_DEVICE, NewParams, 2) < 0) {
    perror("Setting output frequency");
                                                          }
  }
```

```
   sigset(SIGINT, catch_sigint);
   inport = ALopenport("inport", "r", config);

    while (!caught_sigint)    {

     sighold(SIGINT);



     ALreadsamps(inport, buf, NSAMPS);
    /* read 4 sec of sound */
       noisegate( fan,buf,sz);

    /* normalize( fan );*/

    sigrelse(SIGINT);
    ALfreeconfig(config);
/* free the config object    */
    ALcloseport(inport);

    break;

                              }
}

void noisegate( float *orig ,float *buf,int *k )
```

```
{
  long gate = 0;
  long i,j=0;

  for( i = 0; i < NSAMPS; i++ )
   {
     if( abs( buf[i] ) >= THRESHOLD )
        gate = DELAY;

     if( gate <= 0 )
     {
        buf[i] = 0;
        gate = 0;
     }
     else
     {
        orig[j]=buf[i];
        /* printf("orig=%f\n", orig[j]);*/
        j++;
     }
     gate--;
   }
  *k = j;
}
```

```
void normalize( float *buf )
{
   float max = 0;
   float mult;

   for( i = 0; i < NSAMPS; i++ )
     if( abs( buf[i] ) > max )
         max = abs( buf[i] );

   mult = 128/max;
   for( i = 0; i < NSAMPS; i++ )
     buf[i] *= mult;
}


void padding(float *h,float *fag,int siz)
{
  int i, ix;

  for(i=0;i<(siz+FFTSZ);i++)
  {
     ix = i;
     if( ix > siz )
        ix = 2*siz-ix;
     h[i]=fag[ix];
  }
```

```
}

/*
/* collecting 1024 chunks in to 2d array
starts here .14 timesshifting is done.*/
/*  o=k/DIV;
for(i=0;i<DIV;i++)
  {
   for(s=0;s<FFTSZ;s++)
     {
      z[i][s]=h[x];
      x++;
     }
   if(i>0)
     {
      x=o*i;
     }
  }

}*/

void hmwindow( float *z )/*apply hamming for 2d array*/
{
  int r;

  for(r=0;r<FFTSZ;r++)
```

```
        z[r]=z[r]*(0.54-(0.46*cos(2.0*PI*r/FFTSZ)));
}

/*end of recording*/


/*redirect array  b values to a file for future fft use*/
```

```
/*
Nataraj Jeedigunta
  File:fft.h
*/

#ifndef _FFT_H
#define _FFT_H

#define FFT_SIZE    1024/*1024*32*/
void fft2( int k, int n, int no2, float *xr, float *xi,
       int *irb, float *co, float *si, int id );
void InitFFT();
/*float *send_amplitudes_to_FFT( int *num );*/
float *send_to_FFT(float *inter);

#endif
```

```
/*
  Nataraj Jeedigunta
  File:fft.c
We implement a fast fourier transform(FFT)
 in this module for doingfrequency domain analysis.
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "fft.h"

#define SAMPLE_RATE 11025
#define FREQDIV 14
FILE  *fp8;/*FILE ram.raw containing amplitudes*/
FILE  *fp9;/*for storing fft values*/

float pi,w,fir,a,co[FFT_SIZE/2],
si[FFT_SIZE/2],xr[FFT_SIZE],xi[FFT_SIZE];
int k,m,n,j,id,i,irb[FFT_SIZE/2],no2,pop;
short int sd;

float compare[FFT_SIZE/2],comp[FFT_SIZE/2],
sum[4000];/*Holds averaged freqs*/
int temp1;
float retarr[FREQDIV];
```

```
void InitFFT()
{
  n = FFT_SIZE;
  no2 = n / 2;

  pi = acos(-1.0);

  irb[0] = 0;
  k = 0;
  m = n;
  while( m > 1)
  {
    k++;
    m >>= 1;
  }

  for(j = 0; j < k; j++)
  {
    if(j == 0)
      id = 1;
    else
      id *= 2;

    for(i = 0; i < id; i++)
    {
      irb[i] = 2*irb[i];
```

```
      if( j < k-1)
        irb[i+id] = irb[i] + 1;
    }
   }

   w = 2.0*pi/n;

   for(i = 0; i < id; i++)
   {
     fir = irb[i] / 2.0;
     a = fir*w;
     co[i] = cos(a);
     si[i] = sin(a);
   }
}

void fft2( int k, int n, int no2, float *xr, float *xi,
       int *irb, float *co, float *si, int id )
{
   int nc,nb,lb,lb2,is,iff,i2,i,ir,ib,ii,l,ibi;
   float c,s,qr,qi,zr,zi;
   for(nc = 0; nc < k; nc++)
   {
     if(nc == 0) nb = 1;
     else nb = nb*2;
     lb = n/nb;
```

```
    lb2 = lb/2;

    for(ib = 0; ib < nb; ib++)
    {
     c = co[ib];
     s = si[ib];
     is = ib*lb;
     iff = ib*lb+lb2;

     for(i = is; i < iff; i++)
     {
      i2 = i+lb2;
      qr = xr[i2]*c-xi[i2]*s;
      qi = xr[i2]*s+xi[i2]*c;
      xr[i2] = xr[i]-qr;
      xi[i2] = xi[i] - qi;
      xr[i] = xr[i] + qr;
      xi[i] = xi[i] + qi;
     }
  }
}

for(i = 0; i < id; i++)
  for(l = 0; l < 2; l++)
  {
   ir = irb[i] + l;
```

```
      ii = i + l*id;
      if((ir - ii) > 0)
      {
        zr = xr[ir];
        zi = xi[ir];
        xr[ir] = xr[ii];
        xi[ir] = xi[ii];
        xr[ii] = zr;
        xi[ii] = zi;
      }
    }
}
float *send_to_FFT(float *inter)
{
  float freq, sum1;
  int   up,couch1,index,dg;
  int   temp1=0;
  int   count=0;
  int   ok=0;
  int   u=0;

  for( i = 0; i < FFT_SIZE; i++ )
  {
    xr[i]=inter[i];
    xi[i] = 0;
  }
```

```
fft2( k, FFT_SIZE, no2, xr, xi, irb, co, si, id );
ok = 0;
for( i = 0; i < FFT_SIZE/2; i++)
{
   compare[i] = sqrt( xr[i]*xr[i]+xi[i]*xi[i] );
   freq = SAMPLE_RATE*(float)i/(float)FFT_SIZE;


   if( (freq >= 199) && (freq <= 3000) )
   {
      comp[ok] = compare[i];
      ok++;
   }
}

up=0;
couch1=0;
sum1=0;
index=0;
dg = ok/FREQDIV;

for(up=0;up<ok;up++)
{
  couch1++;
  if( couch1==dg)
```

```
    {
      couch1=0;
      retarr[index]=sum1/(float)dg;
      index++;
      sum1=0;
    }
    else { sum1=sum1+comp[up]; }
  }
  return(retarr);
}

/*
float *send_amplitudes_to_FFT( int *num )
{
  float freq;
  int   temp1=0;
  int   count=0;
  int   ok=0;

  fp9 = fopen("FFT_DATA_FILE","wb");
  fp8 = fopen( "ram.raw", "rb" );

  while( !feof(fp8) )
  {
    for( i = 0; i < FFT_SIZE; i++ )
    {
```

```
    xr[i] = 0;
    xi[i] = 0;
}

for( i = 0; i < FFT_SIZE; i++ )
{
    fscanf( fp8, "%f", &xr[i] );
    xi[i] = 0;
}

fft2( k, FFT_SIZE, no2, xr, xi, irb, co, si, id );

ok = 0;
for( i = 0; i < FFT_SIZE/2; i++)
{
    compare[i] = sqrt( xr[i]*xr[i]+xi[i]*xi[i] );
    freq = SAMPLE_RATE*(float)i/(float)FFT_SIZE;

    fprintf( fp9 ,"%16.8f %16.8f\n", freq, compare[i] );
    if( (freq >= 300) && (freq <= 3000) )
    {
        comp[ok] = compare[i];
        ok++;
    }
}
```

```
      pop = i;

      *num = ok;

      break;
    }

  fclose( fp9 );
  fclose( fp8 );
  return( comp );
}

*/
```

APPENDIX B

Source Code For Graphics Module

```
/*
Nataraj Jeedigunta
  File:nox1.c
This file serves as a graphics module.
This program has functions for 5 different
 animations of a Dinosaur model depending upon
the recognized word.
*/

#include <stdio.h>
#include <sys/stat.h>  /* for unix open() */
#include <fcntl.h>     /* for unix open() */
#include <signal.h>    /* for signal handler */
#include <stdio.h>
#include <stdlib.h>
#include <stdlib.h>
#include <string.h>
#include <math.h> /* for cos(), sin(), and sqrt() */
#include <GL/glx.h>/* this includes X and gl.h headers */
#include <GL/glu.h
```

```
/* gluPerspective(), gluLookAt(), GLU popowlygon*/
#include "aux.h"
#include <GL/gl.h>
#include <math.h>
#include <dmedia/audio.h>     /* audio library header */

void jump_dino();
void stop_dino();
void rotate_dino();
void dance_dino();
void speak_greek_dino();
void check();
void climb();
void climbDisplay();
void climb_dino();

int an;
int caught_sigint =  0; /* completion flag */
ALconfig config;/* port configuration structure */
ALport outport; /* audio port structure */
int angle=0;
int ang=5;


typedef enum {
    RESERVED, BODY_SIDE, BODY_EDGE, BODY_WHOLE,
```

```
BODY_SIDE1, BODY_EDGE1, BODY_WHOLE1,

BODY_SIDE2, BODY_EDGE2, BODY_WHOLE2,

BODY_SIDE3, BODY_EDGE3, BODY_WHOLE3,

BODY_SIDE4, BODY_EDGE4, BODY_WHOLE4,

BODY_SIDE5, BODY_EDGE5, BODY_WHOLE5,

BODY_SIDE6, BODY_EDGE6, BODY_WHOLE6,

BODY_SIDE7, BODY_EDGE7, BODY_WHOLE7,

BODY_SIDE8, BODY_EDGE8, BODY_WHOLE8,

BODY_SIDE9, BODY_EDGE9, BODY_WHOLE9,

BODY_SIDE10, BODY_EDGE10, BODY_WHOLE10,

FRONTLEG_SIDE, FRONTLEG_EDGE, FRONTLEG_WHOLE,

BACKLEG_SIDE, BACKLEG_EDGE,BACKLEG_WHOLE,

EYE_SIDE, EYE_EDGE, EYE_WHOLE,
```

```
 UN_SIDE,UN_EDGE, UN_WHOLE,

N_SIDE,N_EDGE, N_WHOLE,


   DINOSAUR,
   DINOSAUR1,
   DINOSAUR2,
   DINOSAUR3,
   DINOSAUR4,
   DINOSAUR5,
 DINOSAUR6,
   DINOSAUR7,
   DINOSAUR8,
   DINOSAUR9,
   DINOSAUR10,

L_SIDE,L_EDGE, L_WHOLE,

V_SIDE,V_EDGE, V_WHOLE


  } displayLists;
```

```
static GLfloat spin = 0.0;
int move=0;
GLdouble bodyWidth = 2.0;
GLfloat body1[][2] =  { {0,3.3}, {1,1.3}, {6,1.3},
  {8.2,2.8},{9,3},
 {10,3}, {11,3}, {12,2.5}, {15,2.5},
 {16,3}, {17,3}, {18,4},
 {19,5},{21,7}, {22,8}, {25,8},
 {23,9}, {25,10}, {24,10}, {23,10.5},
{22,10}, {18,6.5}, {11,6.5},{10,6},{9,5},
{8.2,4.3}, {6,2.3}, {1,2.3}};


GLfloat body[][2] = { {0,3}, {1,1}, {6,1},
{8.2,2.5},{9,3},{10,3}, {11,3},
 {12,2.5}, {15,2.5},
 {16,3}, {17,3}, {18,4},
 {19,5},{21,7}, {22,8}, {25,8.1},
 {23,9}, {25,9.9}, {24,10}, {23,10.5},
{22,10}, {18,6.5}, {11,6.5},{10,6},{9,5}
,{8.2,4}, {6,2}, {1,2}};


GLfloat body2[][2] = { {0,3.6}, {1,1.6},
{6,1.6}, {8.2,3.1},{9,3},
```

```
 {10,3}, {11,3}, {12,2.5}, {15,2.5},
 {16,3}, {17,3}, {18,4},
 {19,5},{21,7}, {22,8}, {25,8.2},
 {23,9}, {25,9.8}, {24,10}, {23,10.5},
{22,10}, {18,6.5}, {11,6.5},{10,6},{9,5},
{8.2,4.6},  {6,2.6}, {1,2.6}};


GLfloat body3[][2] = { {0,3.9}, {1,1.9},
{6,1.9},{8.2,3.4},{9,3},
 {10,3}, {11,3}, {12,2.5}, {15,2.5},
 {16,3}, {17,3}, {18,4},
 {19,5},{21,7}, {22,8}, {25,8.3},
 {23,9}, {25,9.7}, {24,10}, {23,10.5},
{22,10}, {18,6.5}, {11,6.5},{10,6},{9,5},
{8.2,4.9},  {6,2.9}, {1,2.9}};


GLfloat body4[][2] = {{0,4.2}, {1,2.2},
 {6,2.2},{8.2,3.7},{9,3},
 {10,3}, {11,3}, {12,2.5}, {15,2.5},
 {16,3}, {17,3}, {18,4},
 {19,5},{21,7}, {22,8}, {25,8.4},
 {23,9}, {25,9.6}, {24,10}, {23,10.5},
{22,10}, {18,6.5}, {11,6.5},{10,6},{9,5},
```

```
{8.2,5.2},  {6,3.2}, {1,3.2}};


GLfloat body5[][2] = { {0,4.5}, {1,2.5},
{6,2.5},{8.2,4},{9,3},
 {10,3}, {11,3}, {12,2.5}, {15,2.5},
 {16,3}, {17,3}, {18,4},
 {19,5},{21,7}, {22,8}, {25,8.5},
 {23,9}, {25,9.5}, {24,10}, {23,10.5},
{22,10}, {18,6.5}, {11,6.5},{10,6},{9,5},
{8.2,5.5}, {6,3.5}, {1,3.5}};


GLfloat body6[][2] = { {0,4.8}, {1,2.8},
 {6,2.8},{8.2,4.3},{9,3},
 {10,3}, {11,3}, {12,2.5}, {15,2.5},
 {16,3}, {17,3}, {18,4},
 {19,5},{21,7}, {22,8}, {25,8.6},
 {23,9}, {25,9.4}, {24,10}, {23,10.5},
{22,10}, {18,6.5}, {11,6.5},{10,6},{9,5},
{8.2,5.8}, {6,3.8}, {1,3.8}};
```

```
GLfloat body7[][2] = { {0,5.1}, {1,3.1},
{6,3.1}, {8.2,4.5},{9,3},
 {10,3}, {11,3}, {12,2.5}, {15,2.5},
 {16,3}, {17,3}, {18,4},
 {19,5},{21,7}, {22,8}, {25,8.7},
 {23,9}, {25,9.3}, {24,10}, {23,10.5},
{22,10}, {18,6.5}, {11,6.5},{10,6},{9,5},
{8.2,6.1}, {6,4.1}, {1,4.1}};

GLfloat body8[][2] = { {0,5.4}, {1,3.4},
{6,3.4},{8.2,4.8},{9,3},
 {10,3}, {11,3}, {12,2.5}, {15,2.5},
 {16,3}, {17,3}, {18,4},
 {19,5},{21,7}, {22,8}, {25,8.8},
 {23,9}, {25,9.2}, {24,10}, {23,10.5},
{22,10}, {18,6.5}, {11,6.5},{10,6},{9,5},
{8.2,6.4}, {6,4.4}, {1,4.4}};

GLfloat body9[][2] = { {0,5.7}, {1,3.7},
{6,3.7},{8.2,5.1},{9,3},
 {10,3}, {11,3}, {12,2.5}, {15,2.5},
 {16,3}, {17,3}, {18,4},
 {19,5},{21,7}, {22,8}, {25,8.9},
 {23,9}, {25,9.1}, {24,10}, {23,10.5},
{22,10}, {18,6.5}, {11,6.5},{10,6},{9,5},
```

```
{8.2,6.7}, {6,4.7}, {1,4.7}};

GLfloat body10[][2] = { {0,6}, {1,4},
{6,4}, {8.2,5.4},{9,3},
 {10,3}, {11,3}, {12,2.5}, {15,2.5},
 {16,3}, {17,3}, {18,4},
 {19,5},{21,7}, {22,8}, {25,9},
 {23,9}, {25,9.0}, {24,10}, {23,10.5},
{22,10}, {18,6.5}, {11,6.5},{10,6},{9,5},
{8.2,7}, {6,5}, {1,5}};

GLfloat backleg[][2] = { {10,3}, {11,2},
{10,1}, {11,0},
 {13,0}, {11.5,1}, {12,2}, {11,3}};

GLfloat frontleg[][2] = { {16,3}, {17,2},
{16,1}, {17,0},
 {19,0}, {17.5,1}, {18,2}, {17,3}};


GLfloat eye[][2] = { {22,9.5}, {22.5,9.2},
{23,9.5}, {22.5,9.8}};

GLfloat un[][2] = { {10.2,4.2}, {10.2,5}, {10,5},

{10,4}, {11,4}, {11,5},{10.8,5},{10.8,4.2}};
```

```
GLfloat n[][2] = { {12,5}, {12,4}, {12.2,4},
{12.2,4.8}, {12.8,4}, {13,4},{13,5},{12.8,5},
{12.8,4.2}, {12.2,5}};

GLfloat l[][2] = { {14,5}, {14,4}, {15,4},
{15,4.2}, {14.2,4.2}, {14.2,5}};

GLfloat v[][2] = { {16,5}, {16.5,4}, {17,5},
{16.8,5}, {16.5,4.5}, {16.2,5}};



GLfloat lightZeroPosition[] = {10.0, 4.0, 10.0, 1.0};
GLfloat lightZeroColor[] = {.8, 1.,0.8,1.0};
/* green-tinted */
GLfloat lightOnePosition[] = {-1.0, -2.0,1.0, 0.0};
GLfloat lightOneColor[] = {0.6, 0.3, 0.2, 1.0};
/* red-tinted */
GLfloat skinColor[] = {0.0,1.0, 0.1, 1.0},
eyeColor[] ={0.0, 0.0, 0.0,1.0};
GLfloat unColor[] = {0.0, 0.0,1.0,1.0},
nColor[]={1.0,0.0,0.5,1.0},

lColor[]={1.0,0.0,1.0,1.0},
vColor[]={1.0,1.0,0.5,1.0};
```

```
void
extrudeSolidFromPolygon(GLfloat data[][2],
unsigned int dataSize,GLdouble thickness,
 GLuint side, GLuint edge, GLuint whole)
{
    static GLUtriangulatorObj *tobj = NULL;
    GLdouble  vertex[3], dx, dy, len;
    int  i;
    int  count = dataSize / (2 * sizeof(GLfloat));


    if (tobj == NULL) {
        tobj = gluNewTess();
/* create and initialize a GLU polygon
  * tesselation object */
  gluTessCallback(tobj, GLU_BEGIN, glBegin);
  gluTessCallback(tobj, GLU_VERTEX, glVertex2fv);
  gluTessCallback(tobj, GLU_END, glEnd);
    }
    glNewList(side, GL_COMPILE);
        glShadeModel(GL_SMOOTH);
/* smooth minimizes seeing tessellation */
        gluBeginPolygon(tobj);
            for (i = 0; i < count; i++) {
                vertex[0] = data[i][0];
```

```
                vertex[1] = data[i][1];
                vertex[2] = 0;
                gluTessVertex(tobj, vertex, &data[i]);
            }
        gluEndPolygon(tobj);
    glEndList();
    glNewList(edge, GL_COMPILE);
        glShadeModel(GL_FLAT);
/* flat shade keeps angular hands from being
* "smoothed" */
        glBegin(GL_QUAD_STRIP);
        for (i = 0; i <= count; i++) {
            /* mod function handles closing the edge */
            glVertex3f(data[i
 % count][0], data[i % count][1], 0.0);
            glVertex3f(data[i
% count][0], data[i % count][1], thickness);
 /* Calculate a unit normal by
dividing by Euclidean distance. We
* could be lazy and use glEnable
(GL_NORMALIZE) so we could pass in
* arbitrary normals for a very
slight performance hit. */

dx = data[(i + 1) % count][1] - data[i % count][1];
dy = data[i % count][0] - data[(i + 1) % count][0];
```

```
len = sqrt(dx * dx + dy * dy);

glNormal3f(dx / len, dy / len, 0.0);
        }
        glEnd();
    glEndList();
    glNewList(whole, GL_COMPILE);
        glFrontFace(GL_CW);
        glCallList(edge);
        glNormal3f(0.0, 0.0, -1.0);
/* constant normal for side */
        glCallList(side);
        glPushMatrix();
            glTranslatef(0.0, 0.0, thickness);
            glFrontFace(GL_CCW);
            glNormal3f(0.0, 0.0, 1.0);
/* opposite normal for other side */
            glCallList(side);
        glPopMatrix();
    glEndList();
}


void
makeDinosaur(void)
{
```

```
    GLfloat         bodyWidth = 3.0;

extrudeSolidFromPolygon(body, sizeof(body), bodyWidth,
        BODY_SIDE, BODY_EDGE, BODY_WHOLE);

extrudeSolidFromPolygon(frontleg, sizeof(frontleg),
 bodyWidth/4,FRONTLEG_SIDE,  FRONTLEG_EDGE,  FRONTLEG_WHOLE);
extrudeSolidFromPolygon(backleg, sizeof(backleg),
bodyWidth / 2,BACKLEG_SIDE, BACKLEG_EDGE,BACKLEG_WHOLE);
extrudeSolidFromPolygon(eye, sizeof(eye), bodyWidth + 0.2,
        EYE_SIDE, EYE_EDGE, EYE_WHOLE);
extrudeSolidFromPolygon(un, sizeof(un), bodyWidth + 0.2,
       UN_SIDE,UN_EDGE, UN_WHOLE);
extrudeSolidFromPolygon(n, sizeof(n), bodyWidth + 0.2,
       N_SIDE,N_EDGE, N_WHOLE);
extrudeSolidFromPolygon(l, sizeof(l), bodyWidth + 0.2,
       L_SIDE,L_EDGE, L_WHOLE);
extrudeSolidFromPolygon(v, sizeof(v), bodyWidth + 0.2,
       V_SIDE,V_EDGE, V_WHOLE);

extrudeSolidFromPolygon(body1, sizeof(body1), bodyWidth,
        BODY_SIDE1, BODY_EDGE1, BODY_WHOLE1);


extrudeSolidFromPolygon(body2, sizeof(body2), bodyWidth,
        BODY_SIDE2, BODY_EDGE2, BODY_WHOLE2);
```

```
extrudeSolidFromPolygon(body3, sizeof(body3), bodyWidth,
        BODY_SIDE3, BODY_EDGE3, BODY_WHOLE3);

extrudeSolidFromPolygon(body4, sizeof(body4), bodyWidth,
        BODY_SIDE4, BODY_EDGE4, BODY_WHOLE4);

extrudeSolidFromPolygon(body5, sizeof(body5), bodyWidth,
        BODY_SIDE5, BODY_EDGE5, BODY_WHOLE5);

extrudeSolidFromPolygon(body6, sizeof(body6), bodyWidth,
        BODY_SIDE6, BODY_EDGE6, BODY_WHOLE6);

extrudeSolidFromPolygon(body7, sizeof(body7), bodyWidth,
        BODY_SIDE7, BODY_EDGE7, BODY_WHOLE7);

extrudeSolidFromPolygon(body8, sizeof(body8), bodyWidth,
        BODY_SIDE8, BODY_EDGE8, BODY_WHOLE8);
```

```
extrudeSolidFromPolygon(body9, sizeof(body9), bodyWidth,
        BODY_SIDE9, BODY_EDGE9, BODY_WHOLE9);

extrudeSolidFromPolygon(body10, sizeof(body10), bodyWidth,
        BODY_SIDE10, BODY_EDGE10, BODY_WHOLE10);


    glNewList(DINOSAUR, GL_COMPILE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, skinColor);
        glCallList(BODY_WHOLE);
        glPushMatrix();
            glTranslatef(0.0, 0.0, bodyWidth);
            glCallList( FRONTLEG_WHOLE);
            glCallList(BACKLEG_WHOLE);
            glTranslatef(0.0, 0.0, -bodyWidth-bodyWidth/4);
            glCallList(FRONTLEG_WHOLE);
     glTranslatef(0.0, 0.0, -bodyWidth / 4);
            glCallList(BACKLEG_WHOLE);
    glTranslatef(0.0, 0.0, bodyWidth / 2 - 0.1);
            glMaterialfv(GL_FRONT, GL_DIFFUSE, eyeColor);
            glCallList(EYE_WHOLE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, unColor);
         glCallList(UN_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, nColor);
glCallList(N_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, lColor);
```

```
glCallList(L_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, vColor);
glCallList(V_WHOLE);
        glPopMatrix();
    glEndList();

 glNewList(DINOSAUR1, GL_COMPILE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, skinColor);
        glCallList(BODY_WHOLE1);
        glPushMatrix();
            glTranslatef(0.0, 0.0, bodyWidth);
            glCallList( FRONTLEG_WHOLE);
            glCallList(BACKLEG_WHOLE);
            glTranslatef(0.0, 0.0, -bodyWidth-bodyWidth/4);
            glCallList(FRONTLEG_WHOLE);
     glTranslatef(0.0, 0.0, -bodyWidth / 4);
            glCallList(BACKLEG_WHOLE);
    glTranslatef(0.0, 0.0, bodyWidth / 2 - 0.1);
            glMaterialfv(GL_FRONT, GL_DIFFUSE, eyeColor);
            glCallList(EYE_WHOLE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, unColor);
         glCallList(UN_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, nColor);
glCallList(N_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, lColor);
glCallList(L_WHOLE);
```

```
glMaterialfv(GL_FRONT, GL_DIFFUSE, vColor);
glCallList(V_WHOLE);
        glPopMatrix();
    glEndList();

 glNewList(DINOSAUR2, GL_COMPILE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, skinColor);
        glCallList(BODY_WHOLE2);
        glPushMatrix();
            glTranslatef(0.0, 0.0, bodyWidth);
            glCallList( FRONTLEG_WHOLE);
            glCallList(BACKLEG_WHOLE);
            glTranslatef(0.0, 0.0, -bodyWidth-bodyWidth/4);
            glCallList(FRONTLEG_WHOLE);
     glTranslatef(0.0, 0.0, -bodyWidth / 4);
            glCallList(BACKLEG_WHOLE);
    glTranslatef(0.0, 0.0, bodyWidth / 2 - 0.1);
            glMaterialfv(GL_FRONT, GL_DIFFUSE, eyeColor);
            glCallList(EYE_WHOLE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, unColor);
         glCallList(UN_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, nColor);
glCallList(N_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, lColor);
glCallList(L_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, vColor);
```

```
glCallList(V_WHOLE);
        glPopMatrix();
    glEndList();


 glNewList(DINOSAUR3, GL_COMPILE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, skinColor);
        glCallList(BODY_WHOLE3);
        glPushMatrix();
            glTranslatef(0.0, 0.0, bodyWidth);
            glCallList( FRONTLEG_WHOLE);
            glCallList(BACKLEG_WHOLE);
            glTranslatef(0.0, 0.0, -bodyWidth-bodyWidth/4);
            glCallList(FRONTLEG_WHOLE);
     glTranslatef(0.0, 0.0, -bodyWidth / 4);
            glCallList(BACKLEG_WHOLE);
    glTranslatef(0.0, 0.0, bodyWidth / 2 - 0.1);
            glMaterialfv(GL_FRONT, GL_DIFFUSE, eyeColor);
            glCallList(EYE_WHOLE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, unColor);
         glCallList(UN_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, nColor);
glCallList(N_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, lColor);
glCallList(L_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, vColor);
```

```
glCallList(V_WHOLE);
        glPopMatrix();
    glEndList();

glNewList(DINOSAUR4, GL_COMPILE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, skinColor);
        glCallList(BODY_WHOLE4);
        glPushMatrix();
            glTranslatef(0.0, 0.0, bodyWidth);
            glCallList( FRONTLEG_WHOLE);
            glCallList(BACKLEG_WHOLE);
            glTranslatef(0.0, 0.0, -bodyWidth-bodyWidth/4);
            glCallList(FRONTLEG_WHOLE);
     glTranslatef(0.0, 0.0, -bodyWidth / 4);
            glCallList(BACKLEG_WHOLE);
    glTranslatef(0.0, 0.0, bodyWidth / 2 - 0.1);
            glMaterialfv(GL_FRONT, GL_DIFFUSE, eyeColor);
            glCallList(EYE_WHOLE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, unColor);
         glCallList(UN_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, nColor);
glCallList(N_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, lColor);
glCallList(L_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, vColor);
glCallList(V_WHOLE);
```

```
        glPopMatrix();
    glEndList();

glNewList(DINOSAUR5, GL_COMPILE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, skinColor);
        glCallList(BODY_WHOLE5);
        glPushMatrix();
            glTranslatef(0.0, 0.0, bodyWidth);
            glCallList( FRONTLEG_WHOLE);
            glCallList(BACKLEG_WHOLE);
            glTranslatef(0.0, 0.0, -bodyWidth-bodyWidth/4);
            glCallList(FRONTLEG_WHOLE);
     glTranslatef(0.0, 0.0, -bodyWidth / 4);
            glCallList(BACKLEG_WHOLE);
    glTranslatef(0.0, 0.0, bodyWidth / 2 - 0.1);
            glMaterialfv(GL_FRONT, GL_DIFFUSE, eyeColor);
            glCallList(EYE_WHOLE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, unColor);
         glCallList(UN_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, nColor);
glCallList(N_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, lColor);
glCallList(L_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, vColor);
glCallList(V_WHOLE);
        glPopMatrix();
```

```
    glEndList();


glNewList(DINOSAUR6, GL_COMPILE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, skinColor);
    glCallList(BODY_WHOLE6);
        glPushMatrix();
            glTranslatef(0.0, 0.0, bodyWidth);
            glCallList( FRONTLEG_WHOLE);
            glCallList(BACKLEG_WHOLE);
            glTranslatef(0.0, 0.0, -bodyWidth-bodyWidth/4);
            glCallList(FRONTLEG_WHOLE);
     glTranslatef(0.0, 0.0, -bodyWidth / 4);
            glCallList(BACKLEG_WHOLE);
    glTranslatef(0.0, 0.0, bodyWidth / 2 - 0.1);
            glMaterialfv(GL_FRONT, GL_DIFFUSE, eyeColor);
            glCallList(EYE_WHOLE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, unColor);
         glCallList(UN_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, nColor);
glCallList(N_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, lColor);
glCallList(L_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, vColor);
glCallList(V_WHOLE);
        glPopMatrix();
    glEndList();
```

```
glNewList(DINOSAUR7, GL_COMPILE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, skinColor);
    glCallList(BODY_WHOLE7);
        glPushMatrix();
            glTranslatef(0.0, 0.0, bodyWidth);
            glCallList( FRONTLEG_WHOLE);
            glCallList(BACKLEG_WHOLE);
            glTranslatef(0.0, 0.0, -bodyWidth-bodyWidth/4);
            glCallList(FRONTLEG_WHOLE);
     glTranslatef(0.0, 0.0, -bodyWidth / 4);
            glCallList(BACKLEG_WHOLE);
    glTranslatef(0.0, 0.0, bodyWidth / 2 - 0.1);
            glMaterialfv(GL_FRONT, GL_DIFFUSE, eyeColor);
            glCallList(EYE_WHOLE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, unColor);
         glCallList(UN_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, nColor);
glCallList(N_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, lColor);
glCallList(L_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, vColor);
glCallList(V_WHOLE);
        glPopMatrix();
    glEndList();
```

```
glNewList(DINOSAUR8, GL_COMPILE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, skinColor);
    glCallList(BODY_WHOLE8);
        glPushMatrix();
            glTranslatef(0.0, 0.0, bodyWidth);
            glCallList( FRONTLEG_WHOLE);
            glCallList(BACKLEG_WHOLE);
            glTranslatef(0.0, 0.0, -bodyWidth-bodyWidth/4);
            glCallList(FRONTLEG_WHOLE);
     glTranslatef(0.0, 0.0, -bodyWidth / 4);
            glCallList(BACKLEG_WHOLE);
    glTranslatef(0.0, 0.0, bodyWidth / 2 - 0.1);
            glMaterialfv(GL_FRONT, GL_DIFFUSE, eyeColor);
            glCallList(EYE_WHOLE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, unColor);
         glCallList(UN_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, nColor);
glCallList(N_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, lColor);
glCallList(L_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, vColor);
glCallList(V_WHOLE);
        glPopMatrix();
    glEndList();
```

```
glNewList(DINOSAUR9, GL_COMPILE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, skinColor);
    glCallList(BODY_WHOLE9);
        glPushMatrix();
            glTranslatef(0.0, 0.0, bodyWidth);
            glCallList( FRONTLEG_WHOLE);
            glCallList(BACKLEG_WHOLE);
            glTranslatef(0.0, 0.0, -bodyWidth-bodyWidth/4);
            glCallList(FRONTLEG_WHOLE);
     glTranslatef(0.0, 0.0, -bodyWidth / 4);
            glCallList(BACKLEG_WHOLE);
    glTranslatef(0.0, 0.0, bodyWidth / 2 - 0.1);
            glMaterialfv(GL_FRONT, GL_DIFFUSE, eyeColor);
            glCallList(EYE_WHOLE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, unColor);
         glCallList(UN_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, nColor);
glCallList(N_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, lColor);
glCallList(L_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, vColor);
glCallList(V_WHOLE);
        glPopMatrix();
    glEndList();
```

```
glNewList(DINOSAUR10, GL_COMPILE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, skinColor);
    glCallList(BODY_WHOLE10);
        glPushMatrix();
            glTranslatef(0.0, 0.0, bodyWidth);
            glCallList( FRONTLEG_WHOLE);
            glCallList(BACKLEG_WHOLE);
            glTranslatef(0.0, 0.0, -bodyWidth-bodyWidth/4);
            glCallList(FRONTLEG_WHOLE);
     glTranslatef(0.0, 0.0, -bodyWidth / 4);
            glCallList(BACKLEG_WHOLE);
    glTranslatef(0.0, 0.0, bodyWidth / 2 - 0.1);
            glMaterialfv(GL_FRONT, GL_DIFFUSE, eyeColor);
            glCallList(EYE_WHOLE);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, unColor);
         glCallList(UN_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, nColor);
glCallList(N_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, lColor);
glCallList(L_WHOLE);
glMaterialfv(GL_FRONT, GL_DIFFUSE, vColor);
glCallList(V_WHOLE);
        glPopMatrix();
    glEndList();
```

```
}

   void display(void)
  {
      glClear(GL_COLOR_BUFFER_BIT |GL_DEPTH_BUFFER_BIT);
      glPushMatrix();
      glRotatef(spin, 0.0, 1.0, 0.0);
      glTranslatef(-8, -8, -bodyWidth / 2);
      glCallList(DINOSAUR1);
      glPopMatrix();
      glFlush();
      glXSwapBuffers(auxXDisplay(), auxXWindow());

   }


   void spinDisplay(void)
      {
    spin = spin + 2.0;
    if (spin > 360.0)
    spin = spin - 360.0;
      check();
     display();
      }
```

```
void myinit(void)
{
    glClearColor(0.0, 0.0, 1.0, 1.0);

    makeDinosaur();
/*execute the display lists*/
    glEnable(GL_CULL_FACE);
/* ~50% better perfomance than no back-face
* culling on Entry Indigo */
    glEnable(GL_DEPTH_TEST);
/* enable depth buffering */
    glEnable(GL_LIGHTING);
 /* enable lighting */

    glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, 1);
    glLightfv(GL_LIGHT0, GL_POSITION, lightZeroPosition);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, lightZeroColor);
    glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 0.1);
    glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.05);
    glLightfv(GL_LIGHT1, GL_POSITION, lightOnePosition);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, lightOneColor);
    glEnable(GL_LIGHT0);
    glEnable(GL_LIGHT1);          /* enable both lights */
    glShadeModel(GL_FLAT);
```

```
    /*  glMatrixMode(GL_MODELVIEW);
        glTranslatef(0.0,0.0,-50.0);*/


  }

 void myReshape(GLsizei w, GLsizei h)
    {
        glViewport(0, 0, w, h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();

   gluPerspective( /* field of view in degree */ 70.0,
/* aspect ratio */ 1.0,/* Z near */ 1.0, /* Z far */ 120.0);




glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(0.0, 0.0, 30.0,   /* eye is at (0,0,30) */
             0.0,0.0,0.0,    /* center is at (0,0,0) */
             0.0, 1.0, 0.0);
```

```
    /*   glMatrixMode(GL_MODELVIEW);

    glTranslatef(0.0,0.0,-20.0);
    */


     /*     glPushMatrix();

       /*  gluLookAt(0.0, 0.0,30.0,   /* eye is at (0,0,5.0) */
       /*    0.0,0.0,0.0,     /* center is at (0,0,0) */
       /*   0.0, 1.0, 0.0);

            glPopMatrix();*/


     }


void check()
{
   FILE *fp;
   char string[150];

   if( (fp = fopen( "result", "r" ) ) == NULL )
      return;
```

```
    fscanf(fp,"%s",string);
    fclose( fp );

    if(strcmp(string,"rotate")==0)
       rotate_dino();
    else
       if(strcmp(string,"stop")==0)
         stop_dino();

    else
       if(strcmp(string,"jump")==0)
          jump_dino();

    else
       if(strcmp(string,"vanish")==0)
        dance_dino();
    else
       if(strcmp(string,"speak_greek" )==0)
        speak_greek_dino();

 else
       if(strcmp(string,"climb_mountain" )==0)
        climb_dino();

}
```

```
void  jumpDisplay()
 {

     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
     glPushMatrix();
     glTranslatef(-8, -8, -bodyWidth / 2);
     glTranslatef(0.0,move,0.0);
     glCallList(DINOSAUR1);
     glPopMatrix ();
     glXSwapBuffers(auxXDisplay(),auxXWindow());
     check();

      }

void jump(void)
{
 move++;
  if(move==5)
   move=move-5;
   check();
  jumpDisplay();

}

void danceDisplay()
     {
```

```
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
     glXSwapBuffers(auxXDisplay(),auxXWindow());
     check();
       /* ang=ang-5;
     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
     glPushMatrix();
     glTranslatef(-8, -8, -bodyWidth / 2);
     glRotatef(ang,0.0,0.0,1.0);
     glCallList(DINOSAUR1);
     glPopMatrix ();
     glXSwapBuffers(auxXDisplay(),auxXWindow());
     ang=ang+5;

     check();
       */

     }


void climbDisplay()
{
glRotatef(angle,1.0,1.0,1.0);
glPushMatrix();
glTranslatef(-8, -8, -bodyWidth / 2);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
```

```
glCallList(DINOSAUR);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR1);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR2);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR3);
glPopMatrix();
```

```
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR4);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR5);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR6);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());
```

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR7);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR8);
glPopMatrix();
glFlush();

glXSwapBuffers(auxXDisplay(),auxXWindow());
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR9);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
```

```
    glCallList(DINOSAUR10);
 glPopMatrix();
glFlush();

glXSwapBuffers(auxXDisplay(),auxXWindow());

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR9);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR8);
glPopMatrix();
glFlush();

glXSwapBuffers(auxXDisplay(),auxXWindow());
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR7);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());
```

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR6);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR5);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR4);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());
```

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR3);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR2);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR1);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());
glPopMatrix();
check();

  }
```

```
void speakDisplay()
     {

glPushMatrix();
glTranslatef(-8, -8, -bodyWidth / 2);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR1);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR2);
glPopMatrix();
```

```
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR3);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR4);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR5);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());
```

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR6);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR7);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR8);
glPopMatrix();
glFlush();

glXSwapBuffers(auxXDisplay(),auxXWindow());
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```
glPushMatrix();
glCallList(DINOSAUR9);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
    glCallList(DINOSAUR10);
 glPopMatrix();
glFlush();

glXSwapBuffers(auxXDisplay(),auxXWindow());

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR9);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR8);
glPopMatrix();
glFlush();
```

```
glXSwapBuffers(auxXDisplay(),auxXWindow());
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR7);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR6);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR5);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());
```

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR4);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR3);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
glCallList(DINOSAUR2);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());


glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
```

```
glCallList(DINOSAUR1);
glPopMatrix();
glFlush();
glXSwapBuffers(auxXDisplay(),auxXWindow());
glPopMatrix();
system("pl<ram");
system("pl<greek");
check();
     }



void dance(void)
{

danceDisplay();
 check();
}
void climb()
{
angle=angle+5;
if(angle==90)
angle=angle-90;
climbDisplay();
check();
}
```

```
void talk()
{
check();
speakDisplay();

}
void jump_dino()
{

auxIdleFunc( jump);

}

void stop_dino()
{
   auxIdleFunc( check );
}

void rotate_dino()
{
 auxIdleFunc( spinDisplay );
}

void climb_dino()
{
```

```
 auxIdleFunc( climb );
}

void dance_dino()
{
   auxIdleFunc( dance );
}

void speak_greek_dino()
{
auxIdleFunc(talk);

}

  int main(int argc, char** argv)
{
     auxInitDisplayMode(AUX_DOUBLE | AUX_RGBA | AUX_DEPTH);
     auxInitPosition(0, 0, 700, 700);
     auxInitWindow("DIANA");
     myinit();
     auxReshapeFunc(myReshape);
     auxIdleFunc(check);
     auxMainLoop(display);
}
```

BIBLIOGRAPHY

[1] C. A. Kamm, S. Singhal, K. M. Yang , " Speech Recognition For Directory Assistance Applications " 2nd IEEE Workshop on Interactive Voice Technology for Telecommunications Application (IVTTA94).

[2] Hong C. Leung and Judith R. Spitz, "Interactive Speech And Language Systems For Telecommunication Applications At Nynex"2nd IEEE Workshop on Interactive Voice Technology for Telecommunications Application (IVTTA94).

[3] Toshihiro Isobe, Masatoshi Morishima, Fuminori Yoshitani"Voice Activated Home Banking System And Its Field Trail, "proceedings , Fourth International Conference on Spoken Language Processing, (IEEE96).

[4] Yasuhiro Yamazaki and Tsuyoshi Morimoto, "ATR Research Activites On Speech Recognition"2nd IEEE Workshop on Interactive Voice Technology for Telecommunications Application (IVTTA94).

[5] Raymond Lau, Giovanni Flammia, Christine Pao and Victor Zue "WebGALAXY Project"Spoken Language Systems Group MIT Laboratory for Computer Science.

[6] J. Glass, "Multilingual Speech-understanding for Human-computer Interaction"Proc. 3rd CRIM-FORWISS Workshop, Montreal, Quebec, October 1996.

[7] V. Zue , "Multilingual Human-computer Interaction" Proc. 3rd CRIM-FORWISS Workshop, Montreal, Quebec, (October 1996).

[8] Mazo, M. ; Rodriguez, F. J. ; Lazaro, J. L. ; Revenga, P. A, "Electronic Control Of Wheel Chair guided by voice commands" Control Engineering Practice v3 n5 May 1995.

[9] Beattie, P. ; Bishop, M; Katevas, N; Moignard, " New developments In Electric Wheelchairs for Disabled"IEE Colloquium(Digest)IEE Computing and Control Divison Colloquium on Mechatronic Aids for the disabled (May 17 1195).

[10] Jayant and Nikil, " Human machine interaction by voice and gesture" Lucent Technologies.

VITA

Graduate College
University of Nevada, Las Vegas

Nataraj Jeedigunta Graduate Student

Local Address:
4236 Grove Circle; #1
Las Vegas NV-89119

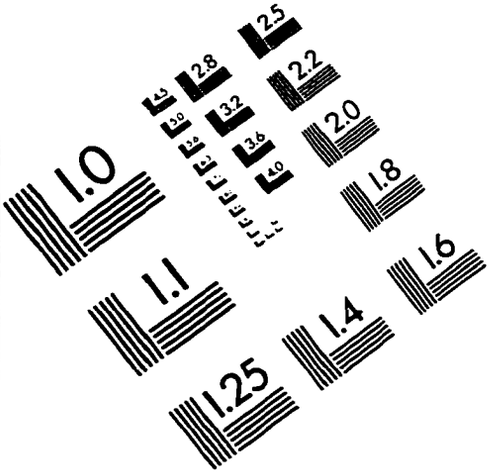
Home Address:
4236 Grove Circle; #1
Las Vegas NV-89119

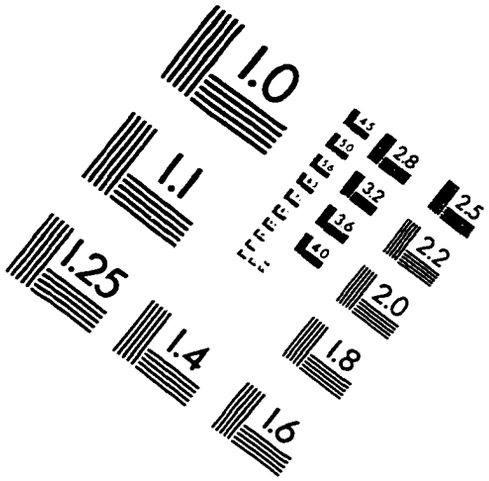
Degrees:
Bachelor of Science,Computer Science, 1992
Andhra University, India.

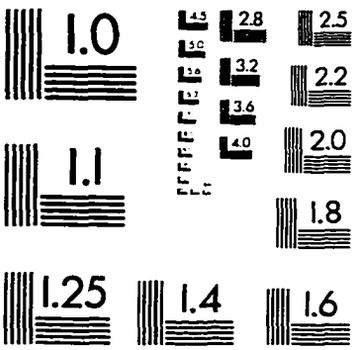
Thesis Title: Voice Interactive Computer Graphics and Games

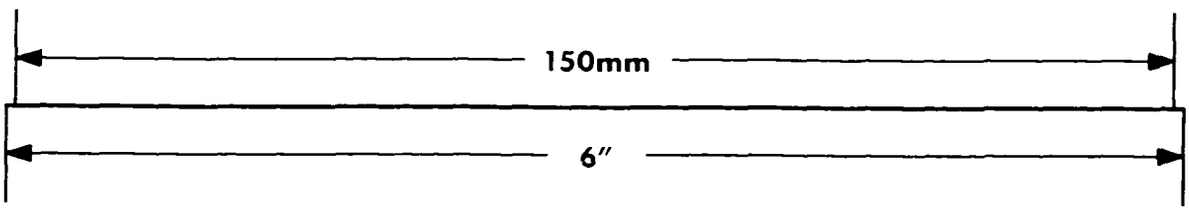
Thesis Examination Committee:
Chairperson, Evangelos A Yfantis, Ph.D.
Examining Committee Member, Roy H Ogawa, Ph.D.
Examining Committee Member, Laxmi P. Gewali, Ph.D.
Graduate Faculty Representative, Sahjendra Singh, Ph.D.

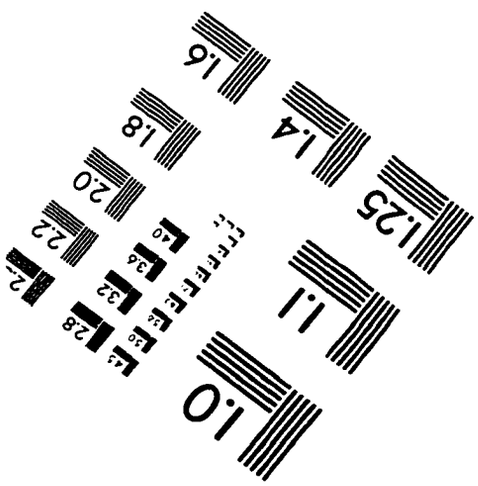
IMAGE EVALUATION TEST TARGET (QA-3)

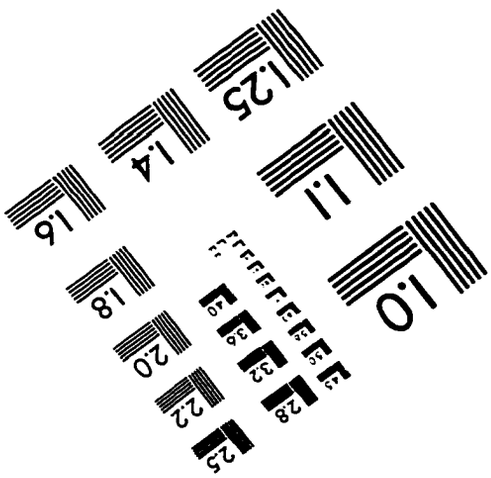












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989