FRACTAL MODELLING OF RESIDUAL IN LINEAR PREDICTIVE CODING OF SPEECH

by

EPIPHANY VERA

A Thesis

Submitted to the Faculty of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

University of Manitoba

Winnipeg, Manitoba, Canada

© E. Vera; April 1999

National Library of Canada

Bibliothèque nationale du Canada

Acquisitions and Bibliographic Services

Acquisitions et services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-41642-9

Canadä

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES
*****
COPYRIGHT PERMISSION PAGE

FRACTAL MODELLING OF RESIDUAL IN LINEAR PREDICTIVE CODING OF SPEECH

BY

EPIPHANY VERA

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University

of Manitoba in partial fulfillment of the requirements of the degree

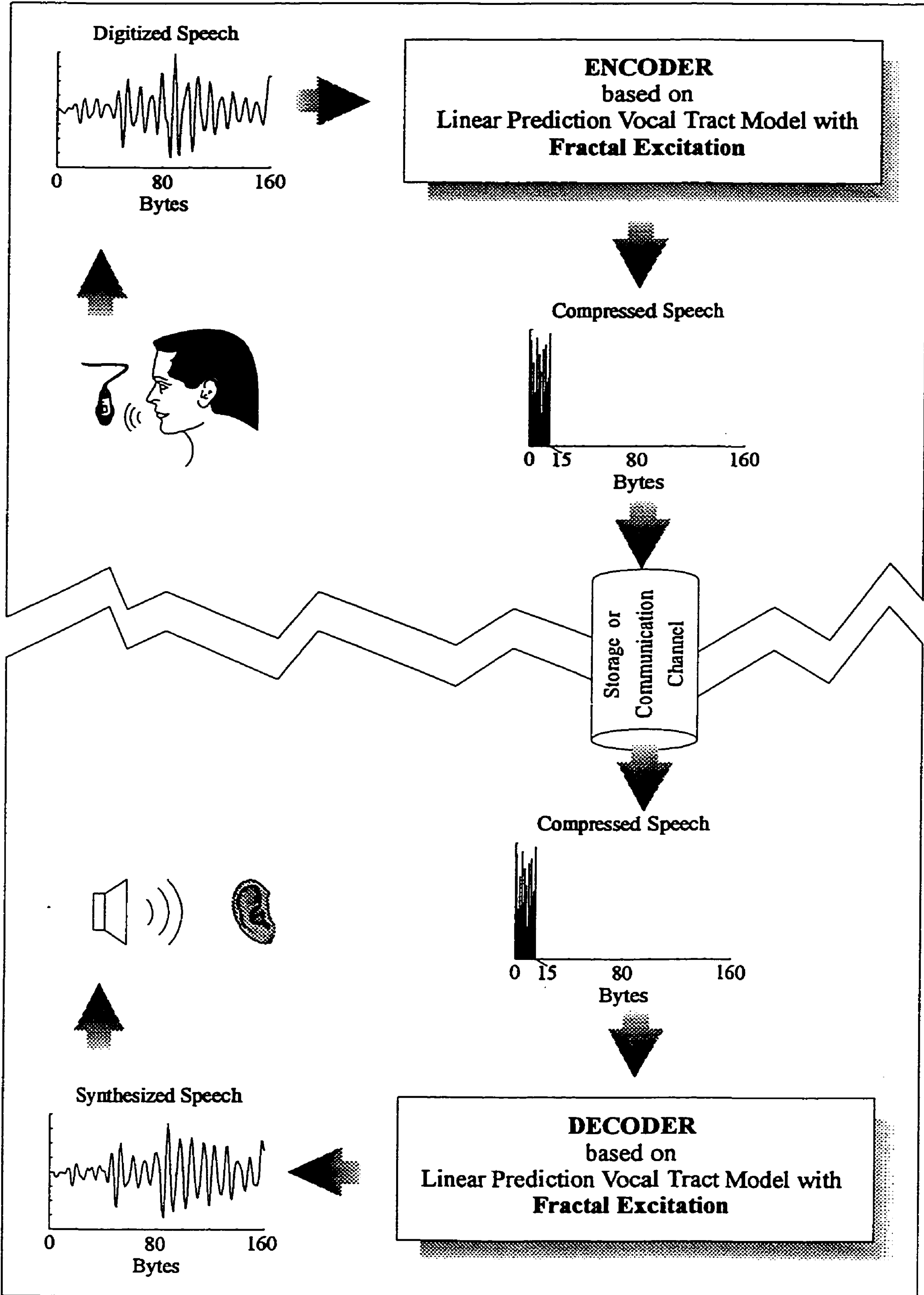
of

MASTER OF SCIENCE

EPIPHANY VERA ©1999

Permission has been granted to the Library of The University of Manitoba to lend or sell copies of this thesis/practicum, to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film, and to Dissertations Abstracts International to publish an abstract of this thesis/practicum.

The author reserves other publication rights, and neither this thesis/practicum nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.



Digitized Speech
0
80
160
Bytes
ENCODER
based on
Linear Prediction Vocal Tract Model with
Fractal Excitation
Compressed Speech
0
15
80
160
Bytes
Storage or
Communication
Channel
Compressed Speech
0
15
80
160
Bytes
Synthesized Speech
0
80
160
Bytes
DECODER
based on
Linear Prediction Vocal Tract Model with
Fractal Excitation

To my parents.

ABSTRACT

Linear predictive modelling of speech forms the basis of most low bit-rate speech coders,. including Linear Predictive Coder 10 (LPC-10), Code Excited Linear Prediction (CELP), Vector Sum Excited Linear Prediction (VSELP), Special Mobile Group (GSM) 06.10. The main difference between such codecs is in the coding of the excitation signal. LPC-10 uses a very compact method for representing the excitation. It produces bit-rates around 2 to 4 kilobits per second (kbps), but the speech quality is very poor and sounds machine-like. CELP and VSELP use a less compact method, but it is computationally more intensive. The resulting bit-rates are as low as 4 kbps, with a better speech quality. GSM uses an even less compact representation, with a bit rate above 10 kbps, but the reproduction quality is very good. In this thesis, a method for modelling speech excitations using fractal interpolation is developed. These fractal interpolation techniques are based on iterated function systems (IFS). Two IFS models are used: (i) the self-affine model and (ii) the piecewise self-affine model. The first model is found to be inefficient for the purpose of representing excitations, while the second model is found to provide a better representation for the speech excitations. Consequently, a 6 kbps speech coder was implemented using the piecewise self-affine fractal model. The coder has a signal-to-noise ratio of 10.9dB and an informal subjective measure found the perceptual quality to be comparable to that of the 13 kbps GSM coder.

ACKNOWLEDGEMENTS

I would like to thank my adviser Dr. Kinsner not only for introducing me to the topics covered in this thesis but also for enkindling my interest in many other topics in computing and signal processing. I also thank TR*Labs* for financial support for my studies. In particular, I am grateful to Len Dacombe and Clint Gibbler for helping me while I was at TR*Labs*. Working at both TR*Labs* and the Delta Research Group provided me with an excellent opportunity to acquire a diverse background in signal processing and telecommunications. Whenever I think of my experience in the Master of Science program, I will always remember my good friend Shamit Bal. I thank him for his support and encouragement. Many thanks go to Armin Langi who had to endure many questions about speech coding. I thank him for his patience. I am also grateful to my best friend Janice Hamilton and my fiancé Chengeto Mukonoweshuro for providing me with encouragement and for all the help they have both given me in preparing this thesis. I thank Bert and Kae Lowen and their children for "adopting" me into their family when I came to study in Winnipeg. I would like to acknowledge some people who helped me with technical advice on implementing some aspects of this thesis. First, Jutta Degener of the Technical University of Berlin for giving me advice on implementation of the GSM 06.10 codec and for obtaining a copy of the hard to get ETSI GSM standard publication for me. Secondly, Dr. Hayes of the Georgia Institute of Technology and Dr. Mazel of IBM for their comments on the implementation of the inverse algorithms for iterated function systems.

TABLE OF CONTENTS

Abstract iv

Acknowledgements v

Table of Contents vi

List of Figures ix

List of Tables x

List of Abbreviations xi

1. Introduction 1
    - Background 1
    - Problem Statement 2
    - Thesis Statement 4
    - Thesis Organization 5

2. Linear Prediction of Speech 7
    - Introduction 7
    - Human Speech Production Model 7
        - Excitation 8
        - Modulation 9
    - LPC Modelling of Speech 9
        - Discrete-Time Speech Production Model 10
        - Principles of Linear Predictive Analysis 12
            - Calculating the Prediction Coefficients 14
        - Linear Prediction Residual 17
        - Pitch Prediction 21
            - First Order Long-Term Predictor 24
        - Pre-emphasis and Post-emphasis 26
    - Summary of Chapter II 28

3. Iterated Function Systems 29
    - Introduction 29
    - Traditional Interpolation Functions 29
    - Iterated Function Systems (IFS) 30
        - The Attractor of an IFS 34
            - Proof of IFS Convergence 35

IFS Based Interpolation 38
Self-Affine Model 38
Piecewise Self-Affine Model 41
Synthesis of an IFS Attractor 43
The Collage Theorem 45
IFS Inverse Algorithm 46
IFS Map Minimal Parameter Set 46
Calculation of the contraction factor 48
Self-Affine Model 52
Piecewise Self-Affine Model 53
Summary of Chapter III 54

4. Fractal Modelling of Speech 55
Introduction 55
Fractal Modelling of Speech 55
Some Application of Fractals to Speech Modelling. 55
Map Coverage Cost Function 58
Review of Terminology 59
Motivation 59
Error Normalization. 60
Linear Coverage Cost Function 61
Non-Linear Map Coverage Cost Function 62
Fractal Modelling of Linear Prediction Excitations. 65
System Architecture 66
Self-affine Model Based Codec. 69
Piecewise Self-Affine Model Based Codec. 70
IFS Parameter Reduction 70
IFS Parameter Determination and Quantization 72
Summary of Chapter IV 73

5. Implementation of Fractal Excited Codec 74
Introduction 74
Test Program 74
The Encoder 75
LPC Analysis 76
LTP Analysis 77
IFS Analysis. 77
The Decoder 78
Speech Compression Lab 79
Summary of Chapter V 81

6. System Verification and Results 82
Introduction 82

Methods of Assessing Speech Quality 82
Signal -to-Noise Ratio 83
Mean Opinion Score 84
Informal MOS Test 86
System Performance Results 88
Performance of Self-Affine Model 88
Performance of Piecewise Self-Affine Model 89
Discussion of Results 93
Summary of Chapter VI 94

7. Conclusions and Recommendations 95

References 97

Appendices

A: Source Code for Piecewise Self-affine Fractal Excited Linear Predictive Codec . . A-1

B: Source Code for the GSM 06.10 Codec B-1

C: Source Code for Self-affine Fractal Excited Linear Predictive Codec C-1

LIST OF FIGURES

Fig. 2.1. The excitation plus modulation model of the human speech production. (After [Pars86]) . . . 7

Fig. 2.2. Discrete-time linear speech production model. . . . 10

Fig. 2.3. Discrete-time linear model with post-emphasis. . . . 12

Fig. 2.5. The power spectrum of the LP residual, (b), is much flatter than that of the original signal (a). . . . 20

Fig. 3.1. Graphs b, c, d, e, and f show the original object, a, after applying an affine transform that translates, scales, rotates, reflects, and shears respectively. . . 33

Fig. 3.2. Inverse algorithm for the self-affine fractal model. . . . 52

Fig. 3.3. Inverse algorithm for the piecewise self-affine model. . . . 54

Fig. 4.1. Graph of non-linear map coverage cost function. . . . 64

Fig. 4.2. Architecture of the encoder. The dotted arrow is needed for the piecewise self-affine model only. . . . 66

Fig. 4.3. Architecture of the decoder. . . . 69

Fig. 5.1. Structure of test program used to evaluate the codec. . . . 74

Fig. 5.2. The encode function. . . . 75

Fig. 5.3. The linear predictive coding analysis routine. . . . 76

Fig. 5.4. Long term prediction analysis routine. . . . 77

Fig. 5.5. IFS analysis and quantization routine for the piecewise self-affine model. . . . 78

Fig. 5.6. The decoder function. . . . 78

Fig. 5.7. Screen shots of the Speech Compression Lab. 80

LIST OF TABLES

Table 3.1. Change in notation to make equations easier to read. 47

Table 4.1. Set of example trial maps with their interpolation point, approximate error, E and normalized approximation error, E_0. 61

Table 4.2. Bit allocation for the log area ratios. 68

Table 6.1. Five point scale used for MOS tests. 85

Table 6.2. Speech quality as a function of map coverage in the self-affine based speech codec. 89

Table 6.3. Speech quality as a function of map coverage in the piecewise self-affine based codec. 90

Table 6.4. Speech quality as a function of analysis frame size for a map coverage of 40 samples. 91

Table 6.5. Speech quality as a function of bits used to quantize the contraction factor. 92

Table 6.6. Speech quality as a function of bits used to quantize the interpolation values. 92

Table 6.7. Bit-allocation of 6 kbps piecewise self-affine based codec. 93

LIST OF ABBREVIATIONS

CELP	Code-Excited Linear Predictive Coding
codec	encoder decoder
ETSI	European Telecommunications Standards Institute
IFS	Iterated Function Systems
GSM	Special Mobile Group (Groupe Spécial Mobile)
IEEE	Institute of Electrical and Electronic Engineering
ITU	International Telecommunication Union
kbps	kilo bits per second
LAR	Log Area Ratios
LP	Linear Prediction or Linear Predictive
LPC	Linear Predictive Coding
LTP	Long Term Prediction
MOS	Mean Opinion Score
PCM	Pulse Code Modulation
SNR	Signal-to-noise Ratio
STP	Short Term Prediction
VSELP	Vector Sum Excited Linear Prediction

CHAPTER I

INTRODUCTION

1.1 Background

The purpose of speech is to communicate. Information theory characterizes speech in terms of its message content or information. The message is generated in the brain which then controls the articulatory engine to produce the speech. The articulatory engine which consists of the lungs, vocal chords and vocal tract manipulates air to produce sound waves which carry the message. A listener receives this sound wave, the speech, and decodes it to obtain the message.

There are many other ways that people have devised for the purpose of transmitting the messages. In many cases there is a need to store the message or transmit the message to a recipient who is not within the hearing range of the speaker. Writing is one of the most widely used methods for this purpose. Writing and many other alternative methods for storing and transmitting the message have a disadvantage in that they lose some of the information that is embedded in speech.

A listener deduces a lot of information from speech. Some of this information includes the identity of the speaker. From the tone of the voice one may tell the emotional state of the speaker. The stress on certain words also conveys information to the listener. Speech also has purely aesthetical properties. When listening to singing, people are sensitive

to these aesthetical properties of speech.

Speech is a communication tool. It conveys to the listener an idea, thought or piece of information from the speaker. It also carries secondary information and possess aesthetic characteristics that are perceived by the listener.

1.2 Problem Statement

In the previous section we mentioned the desire to store messages for later retrieval and also for transmission to people outside the hearing range of the speaker. We also mentioned some of the characteristics and secondary information that speech conveys. The challenge is thus to keep the message in the form of speech but represent the speech in a compact and efficient way for the purpose of storage and transmission.

In digital storage and transmission systems, speech is converted into an electrical signal which in turn is converted into a string of ones and zeros. The compactness of the speech representation is thus measured by the number of ones and zeros (bits) required to represent each second of speech. Telephone quality speech requires 64,000 bits for every second of speech. This is an inefficient representation. Many techniques have therefore been developed to improve digital speech representation.

Speech compression or speech coding is the study of ways to find a more compact representation of speech on digital systems. Most low bit rate speech coding techniques are based on linear predictive (LP) modelling of speech. LP analysis estimates the parameters of

an all-pole filter representation of the vocal tract. It is based on the premise that during a stationary frame of speech it can be modelled by a pole-zero system function which is driven by some excitation sequence.

Code excited linear predictive coding (CELP) is one such technique that is based on linear predictive modelling [ScAt85]. CELP uses codebooks of waveform prototypes for the excitation of the LP filter. During analysis (compression) the parameters of the LP filter are calculated first. These parameters are used to form an inverse LP filter. The original speech is passed through this inverse filter to obtain the approximation error of the LP filter. The error signal obtained from the inverse LP filtering is called the LP-residual or LP-excitation.

In order to obtain high compression rates a very efficient representation of the residual is required. In CELP the LP-residual is represented using 104 bits for a 30 millisecond speech frame. The LP parameters are represented using 34 bits. This gives a total of 138 bits for a 30ms speech frame compared to 1,920 bits for the same speech frame without any compression.

CELP uses two codebooks to represent the residual signal. For the purpose of this discussion it suffices to consider only one codebook. The codebook is a collection of typical residual waveform prototypes. Both the analyser and the synthesizer have a copy of the codebook. Each prototype is given a unique index in the codebook. During compression, the analyser searches the codebook for a prototype that best matches the original residual signal. The index or address of the best matching prototype together with the LP parameters form the compressed speech bit-stream for the speech frame under analysis. During synthesis,

the index is used to retrieve the best matching prototype from the codebook. This prototype is used to excite the LP filter to obtain the synthesized speech.

A shortcoming of most linear prediction based codecs like CELP is that the residual signal is modelled as white noise. Studies have shown that this signal is actually not white noise but coloured noise [Kins94][LuCu92][MaYo90]. This means that the quality of speech produced by these codecs is lower than it could be. A technique is therefore required to give a better representation of the residual.

In this section we described the digitization of speech and linear predictive modelling of speech. Linear prediction was identified as being the most widely used technique in speech compression. A shortcoming of linear prediction based codecs was identified as being the modelling of the linear prediction residual as white noise. In the next section, the thesis objective is presented.

1.3 Thesis Statement

The objective of this thesis is to investigate the use of fractal interpolation for providing a better model for the linear prediction residual. By providing a better model for the residual, we hope to improve speech quality while lowering the bit-rate of linear prediction based codecs.

1.4 Thesis Organization

This chapter gives a brief introduction into speech communication and compression. The problem statement and motivation for the thesis are presented.

Chapter 2 gives a more in-depth introduction to speech modelling. It starts off with a description of the human speech production system. The modelling of speech production using linear prediction is then presented. An explanation is then given of how linear predictive (LP) coding is used for compressing speech.

Chapter 3 is an introduction to iterated function systems. Two iterated function system models are presented; the self-affine iterated function system and the piecewise self-affine iterated function system model. Techniques for finding the IFS parameters that best represent a given data set are then presented.

Chapter 4 starts with a literature review of fractal based speech coding techniques. A new technique for modelling the speech residual using iterated function systems is then presented. The self-affine and the piecewise self-affine iterated function systems are studied as candidates for modelling the residual. The architecture of a fractal based linear predictive codec is presented. Chapter 5 gives the software architecture of the implementation of this codec.

Chapter 6 gives a brief discussion of objective and subjective measures for evaluating speech quality. A 6 kbps speech codec based on the model presented in Chapter 5 is developed. The signal-to-noise ratio speech quality measure is used as a guide for selecting codec parameters. Evaluation of the fractal based systems based on signal-to-noise ratio

measure and mean opinion scores are presented. Chapter 7 gives the conclusions and recommendations for future work.

CHAPTER II

LINEAR PREDICTION OF SPEECH

2.1 Introduction

Linear prediction of speech is the basis for most low bit rate speech coders. It is used to form a model of the vocal and nasal tracts. The linear predictor itself is a finite impulse response (FIR) filter. It can also be viewed as a system that predicts samples of a time series based on a linear combination of the past samples [DePH87]. In this chapter we are going to describe the human speech production model. We will then introduce the vocal tract model based on linear prediction. The chapter will conclude with a brief description of techniques used to model the excitations in different speech codecs.

2.2 Human Speech Production Model

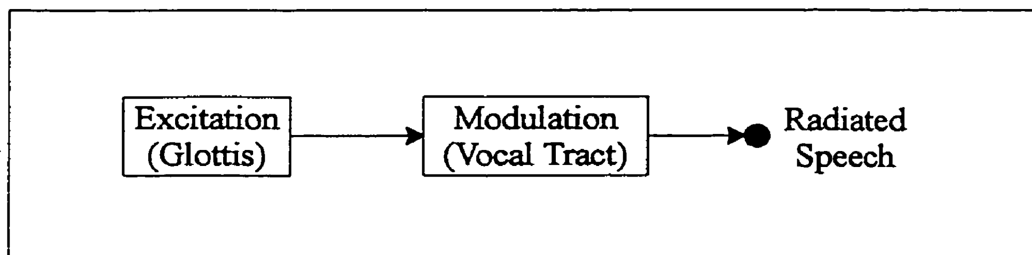


Fig. 2.1. The excitation plus modulation model of the human speech production. (After [Pars86])

The speech production process in human beings can be divided into two sub-

processes; excitation and modulation. Fig. 2.1 shows a block diagram of the model.

2.2.1 Excitation

There are two main types of excitation: voiced excitation and unvoiced excitation. In some cases four other types of excitation are considered. They include mixed, plosive, whisper, and silence. The first three are combinations of voiced and unvoiced. Silence is not really an excitation but is included for modelling purposes.

Voiced sounds are produced by forcing air through the opening between the vocal chords. The source of the compressed air is the lungs. When the abdominal muscles push the diaphragm up, they push air out of the lungs. This air then goes into the trachea and then through the glottis. If the vocal chords have some tension, they start to vibrate. Varying the tension in the vocal chords varies the frequency at which they vibrate.

The frequency at which the vocal folds vibrate is termed the fundamental frequency. This fundamental frequency depends on the size and tension in the vocal chords. The fundamental frequency is also referred to as the pitch. Most men have pitch in the range of 50-250 Hz whereas most women have pitch in the range 120-500 Hz. The reason for the difference is that the average size of vocal folds in men is larger than in women. Larger vocal folds vibrate with a lower pitch than the smaller ones. Each individual has a characteristic pitch which is determined by the physical characteristics of their larynx. Variation from the characteristic pitch is used in intonation and in stress.

Unvoiced sounds are made by forming a constriction along the vocal tract and forcing air through that constriction. This produces a turbulent noise like excitation.

2.2.2 Modulation

Modulation is performed on the excitation to produce speech. It occurs in the pharynx, the oral cavity and the nasal cavity; which are collectively known as the vocal tract. The vocal tract is a series of tubes whose size can be modified when speaking. Changing the size and shape of the vocal tract changes its resonant frequencies. The effect on the excitation as it passes through these tubes is modulation. The control of these resonant frequencies, also known as formant frequencies, is required to produce all the vowels and some of the consonants.

2.3 LPC Modelling of Speech

Various models have been devised to mimic the human speech production system. Thes include mechanical speaking machines by Kratzenstein (1779), von Kempelen (1791), Wheatstone (1835), Faber (1846) and many others [DePH87]. These machines typically had a compressed air source which forced air into a system of resonators. The resonators could then be manipulated to produce speech-like sounds. These early experiments with speech production led to electrical analog devices for speech production.

One of the earlier known analog systems was invented by J. Q. Stewart (1922). In 1939, Dudley, Riesz and Watkins demonstrated an all electrical speech synthesizer. This system had a random noise generating circuit that was used for unvoiced excitation and an oscillator circuit used for voiced excitation. The excitation was used to drive resonance

circuits which mimicked the vocal tract. In 1950, H. K. Dunn designed an improved system that had a vibrating energy source that was used to drive a series of low-pass analog filter sections. These analog systems inspired the digital models for speech production.

2.3.1 Discrete-Time Speech Production Model

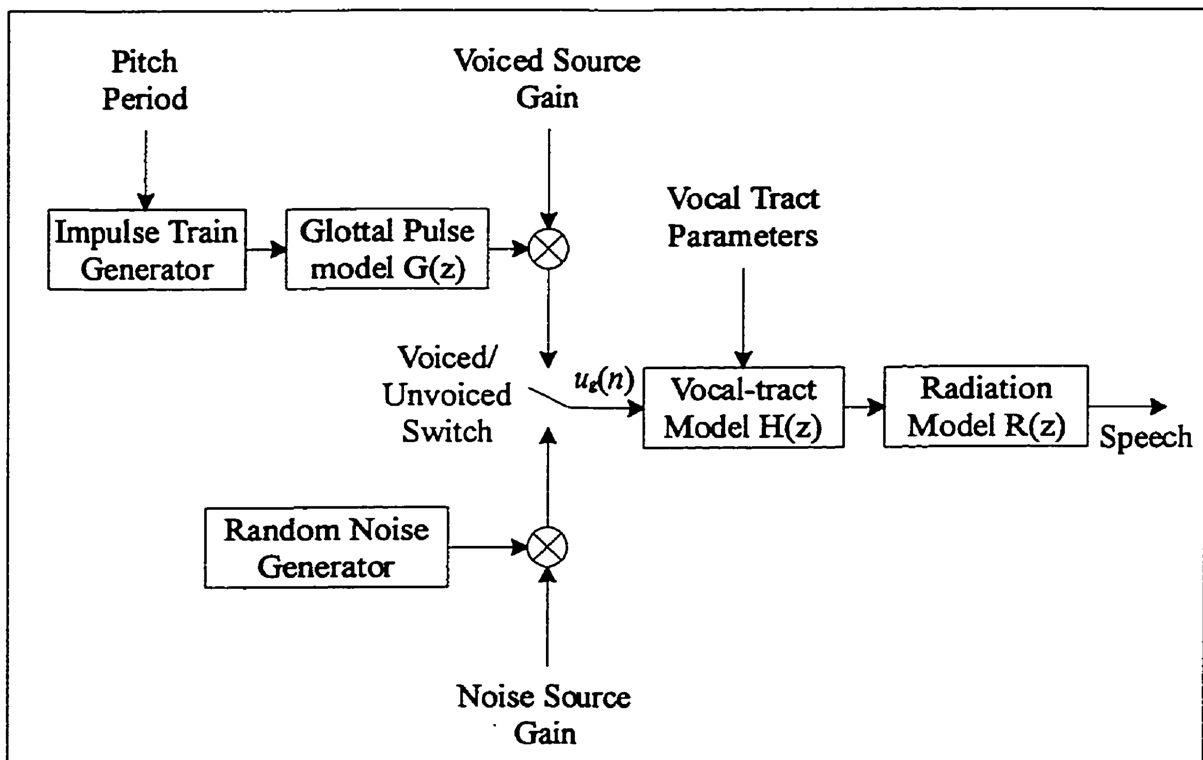


Fig. 2.2. Discrete-time linear speech production model.

Figure 2.2. shows a discrete-time linear model for speech production. The signals in this system are only superficially analogous to the true human speech production model. This system does not provide for non-linear effects between subsystems in the model. This makes it simpler to analyse but makes it inadequate for representing all aspects of speech. Notice

for instance that plosive excitation can not be modelled in this system. The system does not allow for the mixing of voiced and unvoiced excitation either. Despite these limitations, this model produces good quality speech.

The vocal tract model, $H(z)$, is excited by a discrete-time glottal excitation signal, $u_g(n)$. During unvoiced speech segments the excitation is a random noise generator. The random noise generator is a white noise generator thus its output has a flat power spectrum. During voiced segments the glottal excitation is the output of the glottal pulse model $G(z)$.

The glottal pulse shaping filter, $G(z)$, is excited by an impulse train generator. The impulse spacing is based on an estimate of the local pitch period. The filter $G(z)$ produces the required glottal wave shape. The glottal pulse has been shown to introduce a low-pass filtering effect. A two-pole low-pass filter model has been shown to be a good model [RaSc78].

The radiation model, $R(z)$, represents radiation effects at the lips. This is a high-pass filtering operation. It can be adequately represented by a one zero high-pass filter.

As the system in Fig. 2.2. is linear, it can be rearranged such that the glottal model and the lip radiation model are combined into one post-emphasis filter as shown in Fig. 2.3. This is a convenient arrangement because both the glottal model and the lip radiation are time-invariant. This reduces the system to two filters, the post-emphasis filter and the time varying filter $H(z)$. The next section describes the time varying filter, $H(z)$, which models the vocal tract.

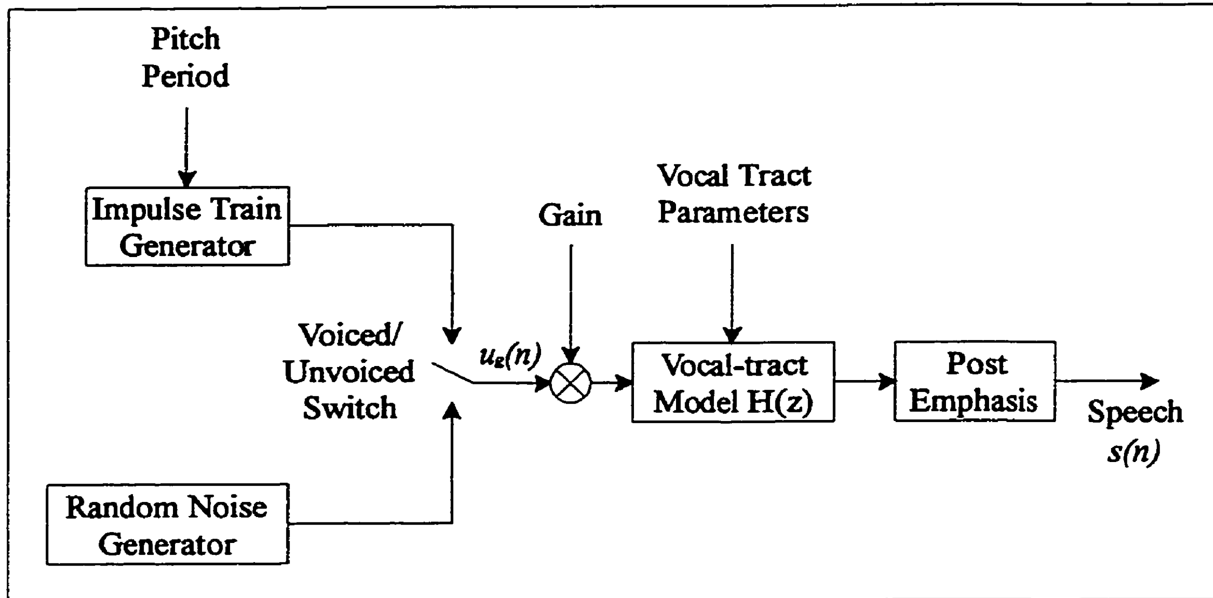


Fig. 2.3. Discrete-time linear model with post-emphasis.

2.3.2 Principles of Linear Predictive Analysis

In the last section, a discrete-time model of speech production was introduced. An important requirement for this model to be practical is that speech can be treated as a stationary signal within some finite period of time. The physical constraints that limit the rate at which our vocal tracts can change shape make it possible for this constraint to be satisfied. Speech can be considered to be stationary for up to approximately 40 milliseconds.

In linear prediction, speech is analysed in segments that are generally 30 milliseconds or less. $H(z)$ is an all-pole steady-state filter which represents the vocal tract frequency response during that fixed time period. The transfer function of this filter has the form shown in Eq. 2.1.

$$H(z) = \frac{G}{A(z)} = \frac{G}{1-\sum_{k=1}^{p} a_k \cdot z^{-k}} \tag{2.1}$$

The parameters in this model, the gain G and the coefficients $\{a_k\}$, can be estimated using linear predictive analysis. The definition of a linear predictor with coefficients α_k is given by

$$x`(n) = \sum_{k=1}^{p} \alpha_k \cdot x(n-k) \tag{2.2}$$

where $x(n)$ represents the sampled signal under analysis, $x`(n)$ is a prediction of the sample $x(n)$ based on the previous p samples and p is the order of the linear predictor. The prediction error of a linear predictor is defined as

$$e(n) = x(n) - x`(n) \tag{2.3}$$

To show how linear prediction is used to model speech production, we refer back to Fig. 2.3. The input of the time-varying filter is $u(n)$ with the output $s(n)$, if (for simplicity) we ignore the post-emphasis filter. The post-emphasis filter will be discussed later in this chapter. Given $u(n)$ and $s(n)$ as the input and output of the filter $H(z)$, Eq. 2.1 can thus be written as

$$H(z) = \frac{S(z)}{U(z)} = \frac{G}{1-\sum_{k=1}^{p} a_k \cdot z^{-k}} \tag{2.4}$$

This equation can be rearranged to give the following difference equation

$$s(n) = \sum_{k=1}^{p} a_k \cdot s(n-k) + G \cdot u(n) \tag{2.5}$$

If *s(n)* is substituted for the *x(n)* in Eq. 2.2, we obtain

$$s`(n) = \sum_{k=1}^{p} a_k \cdot s(n-k) \tag{2.6}$$

A(z) is the prediction filter and $\{a_k\}$ is the set of prediction coefficients. Since the speech signal is assumed to obey Eq. 2.5, then by Eq. 2.3 and 2.6 we can deduce that $G \cdot u(n)$ is the prediction error of the linear predictor *A(z)*.

2.3.2.1 Calculating the Prediction Coefficients

The main issue in linear prediction analysis is solving for the predictor coefficients. The set of predictor coefficients $\{a_k\}$ have to be obtained directly from the speech signal. This has to be done such that the resulting predictor provides a good estimate of the spectral properties of the speech signal. The approach used is based on finding a set of predictor coefficients that minimize the mean-squared prediction error over a short segment of the digitized speech waveform. For a given predictor order, *p*, the mean-squared prediction error is

$$E = \sum_{n=-\infty}^{\infty} e^2(n) = \sum_{n=-\infty}^{\infty} [s(n) - s`(n)]^2 \tag{2.7}$$

Substituting Eq. 2.6 into Eq. 2.7, yields

$$E = \sum_{n=-\infty}^{\infty} [s(n) - \sum_{k=1}^{p} a_k \cdot s(n-k)]^2 \tag{2.8}$$

As the analysis has to be done on a segment of the speech waveform over which speech can be considered to be stationary we take this window to be of length N samples. To simplify the model, the speech waveform is assumed to be identically zero outside the window under analysis. If we define the analysis window starting at n=0, the limits of summation in Eq. 2.8 can be changed to include only non-zero values of the mean-square error.

$$E = \sum_{n=0}^{N+p-1} [s(n) - \sum_{k=1}^{p} a_k \cdot s(n-k)]^2 \tag{2.9}$$

To find the optimal predictor, the mean squared error has to be minimized with respect to each predictor coefficient. This is done by taking partial derivatives of the error with respect to each coefficient, equating each partial derivative to zero and solving the resulting system of equations for the coefficients. This system of equations is obtained by taking p partial derivatives on Eq. 2.9 which gives

$$\frac{\delta E}{\delta a_i} = 0 = 2 \cdot \sum_{n=0}^{N+p-1} [(s(n) - \sum_{k=1}^{p} a_k \cdot s(n-k)) \cdot s(n-i)] \qquad 1 \le i \le p \tag{2.10}$$

Equation 2.10 can be simplified to

$$\sum_{k=1}^{p} a_k \cdot [\sum_{n=0}^{N+p-1} s(n-k) \cdot s(n-i)] = \sum_{n=0}^{N+p-1} s(n) \cdot s(n-i) \qquad 1 \le i \le p \tag{2.11}$$

Equation 2.11 can further be rewritten as

$$\sum_{k=1}^{p} a_k \cdot [\sum_{n=0}^{N-1-k} s(n) \cdot s(n+i-k)] = \sum_{n=0}^{N} s(n) \cdot s(n-i) \qquad 1 \le i \le p \tag{2.12}$$

Finally, Eq. 2.12 can be further simplified to

$$\sum_{k=1}^{p} a_k \cdot R(|i-k|) = R(i) \qquad 1 \le i \le p \tag{2.13}$$

where R the autocorrelation function is defined as

$$R(k) = \sum_{n=-\infty}^{\infty} s(n) \cdot s(n-k) = \sum_{n=-\infty}^{\infty} s(n) \cdot s(n+k) \tag{2.14}$$

The system of equations given by Eq. 2.13 can be expressed in matrix form as

$$\begin{bmatrix} R(0) & R(1) & R(2) & \cdots & R(p-1) \\ R(1) & R(0) & R(1) & \cdots & R(p-2) \\ R(2) & R(1) & R(0) & \cdots & R(p-3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ R(p-1) & R(p-2) & R(p-3) & \cdots & R(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \cdots \\ a_p \end{bmatrix} = \begin{bmatrix} R(1) \\ R(2) \\ R(3) \\ \vdots \\ R(p) \end{bmatrix} \tag{2.15}$$

The matrix of autocorrelation values is a Toeplitz matrix. This means it is symmetric and all the elements along any given diagonal are equal. This property is used for solving for the coefficients. The most widely used technique for solving for the coefficients is the Levinson-Durbin algorithm.

2.3.3 Linear Prediction Residual

In the previous section, it was stated that there is a prediction error associated with the linear prediction model. This error was defined as

$$e(n) = s(n) - s^{`}(n) \tag{2.16}$$

This error is also referred to as the LP (linear prediction) residual. In the discrete-time linear speech production model presented in Fig. 2.3, the LP residual gives a good approximation of the excitation signal. Thus the LP residual is analogous to the vocal tract excitation in the human speech production model.

In an LP system, speech is analysed to obtain the coefficients, $\{a_k\}$, that minimize the residual error. After obtaining these coefficients, they can be used to obtain an inverse filter

with the transfer function $H^{-1}(z)$. Using the original speech signal and this inverse filter, the LP residual can be calculated.

Since

$$S(z) = H(z) \cdot U(z) \tag{2.17}$$

it follows that

$$U(z) = H^{-1}(z) \cdot S(z) \tag{2.18}$$

The inverse function is given by

$$H^{-1}(z) = 1 - \sum_{k=1}^{p} a_k \cdot z^{-k} \tag{2.19}$$

The LP residual is thus calculated using the following difference equation

$$u(n) = s(n) - \sum_{k=1}^{p} a_k \cdot s(n-k) \tag{2.20}$$

The LP residual is found to have a much lower spectral variation than the original signal. This is because the LP inverse filter removes the effects of the formants from the speech signal. For this reason, the LP inverse filter is sometimes referred to as a whitening filter. Fig. 2.4 shows a speech sample in the time domain, and the corresponding LP residual. Fig. 2.5 shows the power spectra of the original speech signal and that of the LP residual. It is clear

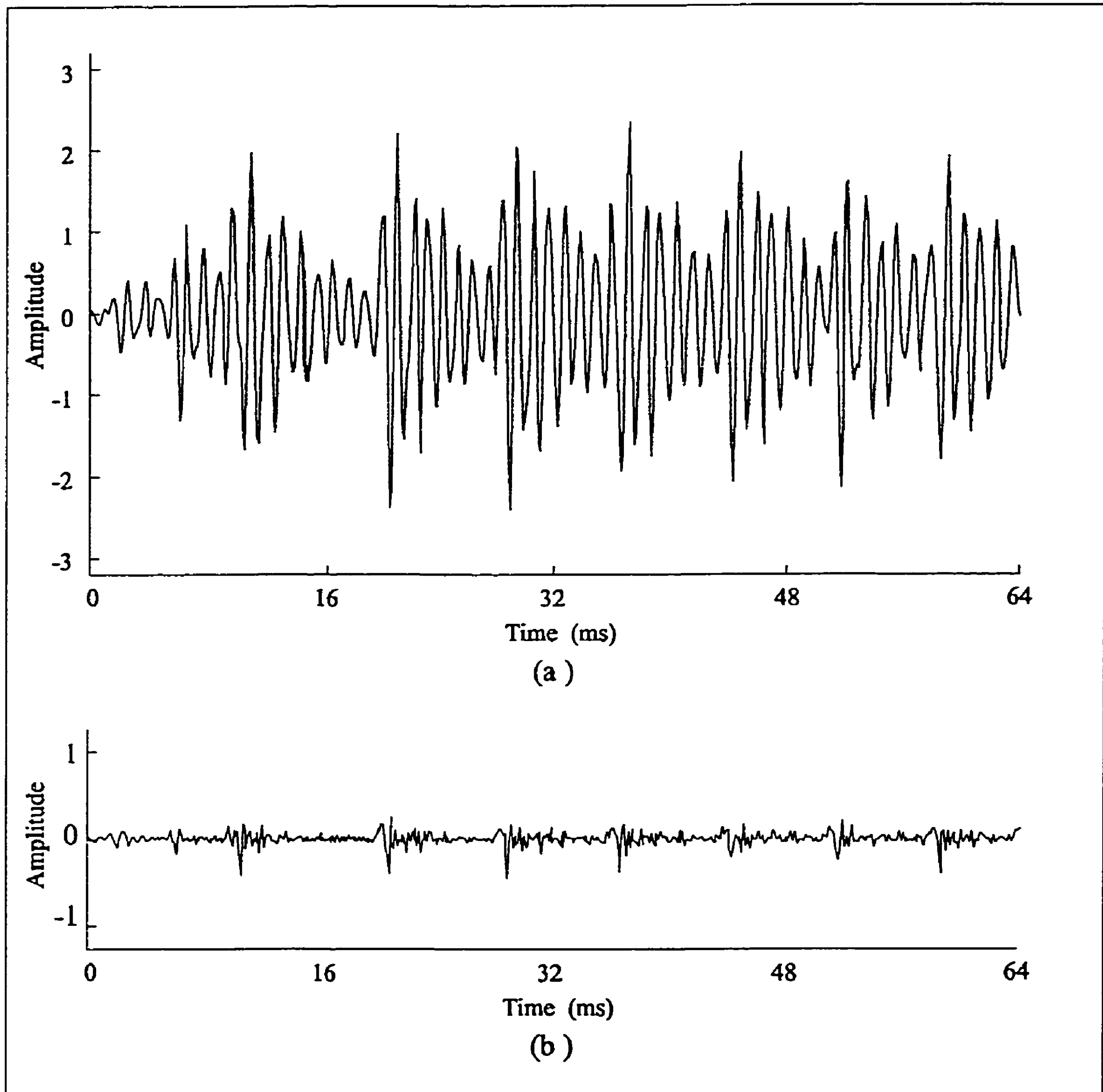


Fig. 2.4. Time domain plots of (a) a speech sample and (b) its LP residual signal.

from the two graphs the power spectrum of the residual is much flatter. The graph of the residual in Fig. 2.4 shows that there are still some long-term correlations in the signal. The next section describes pitch prediction which is used to further flatten the error signal.

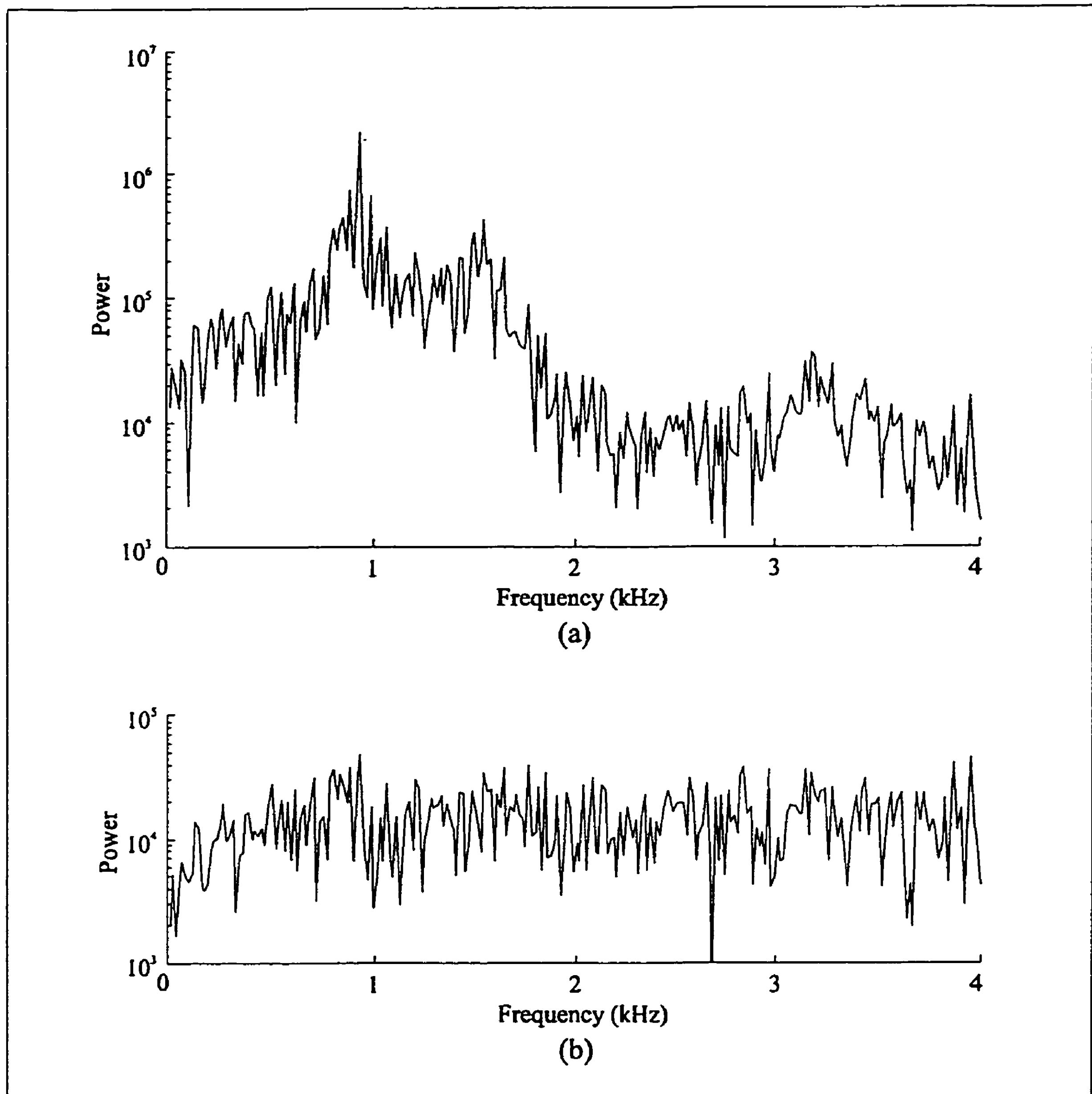


Fig. 2.5. The power spectrum of the LP residual, (b), is much flatter than that of the original signal (a).

2.3.4 Pitch Prediction

Linear prediction analysis is used to remove short-term correlations in the speech signal. These correlations are related to the formant frequencies of the vocal tract. There is another strong form of correlation that is found in voiced speech. These correlations are between samples that are one pitch or multiple pitch periods apart.

A linear predictor similar to the one described for the vocal tract model can be used to model the long-term correlation left the in LP speech residual. This pitch prediction filter is usually referred to as the long-term prediction (LTP) filter. The filter termed the LP predictor in previous sections is sometimes called the short-term predictor (STP). The LTP has the following transfer function

$$P(z) = \frac{1}{1 - \sum_{j=-I}^{I} b_j \cdot z^{-(j+\tau)}} \qquad (2.21)$$

where τ is the pitch period and b_j are the pitch gain coefficients. The pitch period, τ, is also called the lag of the pitch prediction filter. The order of the LTP filter is related to the variable I by:

$$P_L = 2 \cdot I + 1 \qquad I \geq 0 \qquad (2.22)$$

where P_L is the order of the long-term predictor.

In modelling the LP residual, the LTP has an error associated with it. Based on the prediction error of a linear predictor defined in Eq. 2.3, the error of the LTP can be defined

as

$$e(n) = u(n) - \sum_{j=-I}^{I} b_j \cdot u(n-\tau-j) \tag{2.23}$$

Assuming that the lag, τ, is known, the same procedure used in Section 2.3.2 to solve for the LPC coefficients $\{a_k\}$ can be used to formulate a system of equations to solve for the LTP gain coefficients $\{b_j\}$. The mean square error, is obtained from Eq. 2.23

$$E = \sum_{n=0}^{M-1} [u(n) - \sum_{j=-I}^{I} b_j \cdot u(n-\tau-j)]^2 \tag{2.24}$$

By minimizing the mean squared error with respect to each coefficient, the following system of equations is obtained.

$$\begin{bmatrix} V(-I,-I) & V(-I,-I+1) & V(-I,-I+2) & \cdots & V(-I,I) \\ V(-I+1,-I) & V(-I+1,-I+1) & V(-I+1,-I+2) & \cdots & V(-I+1,I) \\ V(-I+2,-I) & V(-I+2,-I+1) & V(-I+2,-I+2) & \cdots & V(-I+2,I) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ V(I,-I) & V(I,-I+1) & V(I,-I+2) & \cdots & V(I,I) \end{bmatrix} \begin{bmatrix} b_{-I} \\ b_{-I+1} \\ b_{-I+2} \\ \cdots \\ b_I \end{bmatrix} = \begin{bmatrix} R(\tau-I,0) \\ R(\tau-I+1,0) \\ R(\tau-I+2,0) \\ \vdots \\ R(\tau+I,0) \end{bmatrix} \tag{2.25}$$

where

$$R(\tau+i,0) = \sum_{n=0}^{M-1} u(n-\tau-i) \cdot u(n) \tag{2.26}$$

and

$$V(i,j) = \sum_{n=0}^{M-1} u(n-\tau-i) \cdot u(n-\tau-j) \tag{2.27}$$

The LTP coefficients can be obtained by solving Eq. 2.25. In Eqs. 2.26 and 2.27, M refers to the size of the frame used for the LTP analysis. The size of the window from which these M samples are taken from is required to be longer than M. This is because the pitch period, τ, is approximated to fall in the following range

$$2ms \leq T \leq 20ms \tag{2.28}$$

This means that for speech sampled at 8,000 samples per second, the pitch lag, τ, falls in the range

$$16 \ samples \leq \tau \leq 160 \ samples \tag{2.29}$$

The analysis window is thus required to be length $(M+\tau_{max})$ such that it contains more than one complete pitch period.

This section has presented general formulation for LTP analysis. The analysis depends on a known value of the pitch lag. Section 2.3.4.1 shows how the pitch lag can be estimated for a first order LTP filter.

2.3.4.1 First Order Long-Term Predictor

In this section, the LTP formulation developed in the previous section is used to estimate the parameters of a first order pitch predictor.

A first order long-term predictor has the variable I, of Eq. 2.21, equal to zero. This reduces the Eq. 2.21 to

$$P_1(z) = \frac{1}{1 - b \cdot z^{-(j+\tau)}} \tag{2.30}$$

From Eq. 2.25, the solution for the filter coefficient, b, is defined as

$$b = \frac{R(\tau,0)}{V(0,0)} = \frac{\sum_{n=0}^{M-1} u(n) \cdot u(n-\tau)}{\sum_{n=0}^{M-1} [u(n-\tau)]^2} \tag{2.31}$$

If the optimal coefficients for the LTP filter are known, Eq. 2.24 can be used to find the mean squared error corresponding to these optimal coefficients. In the case of first order LTP, if the minimum mean squared error is known, the filter coefficient, b, can be calculated using Eq. 2.31.

To estimate the minimum mean squared energy, Eq. 2.32 is used. It is obtained by substituting Eq. 2.31 into 2.24.

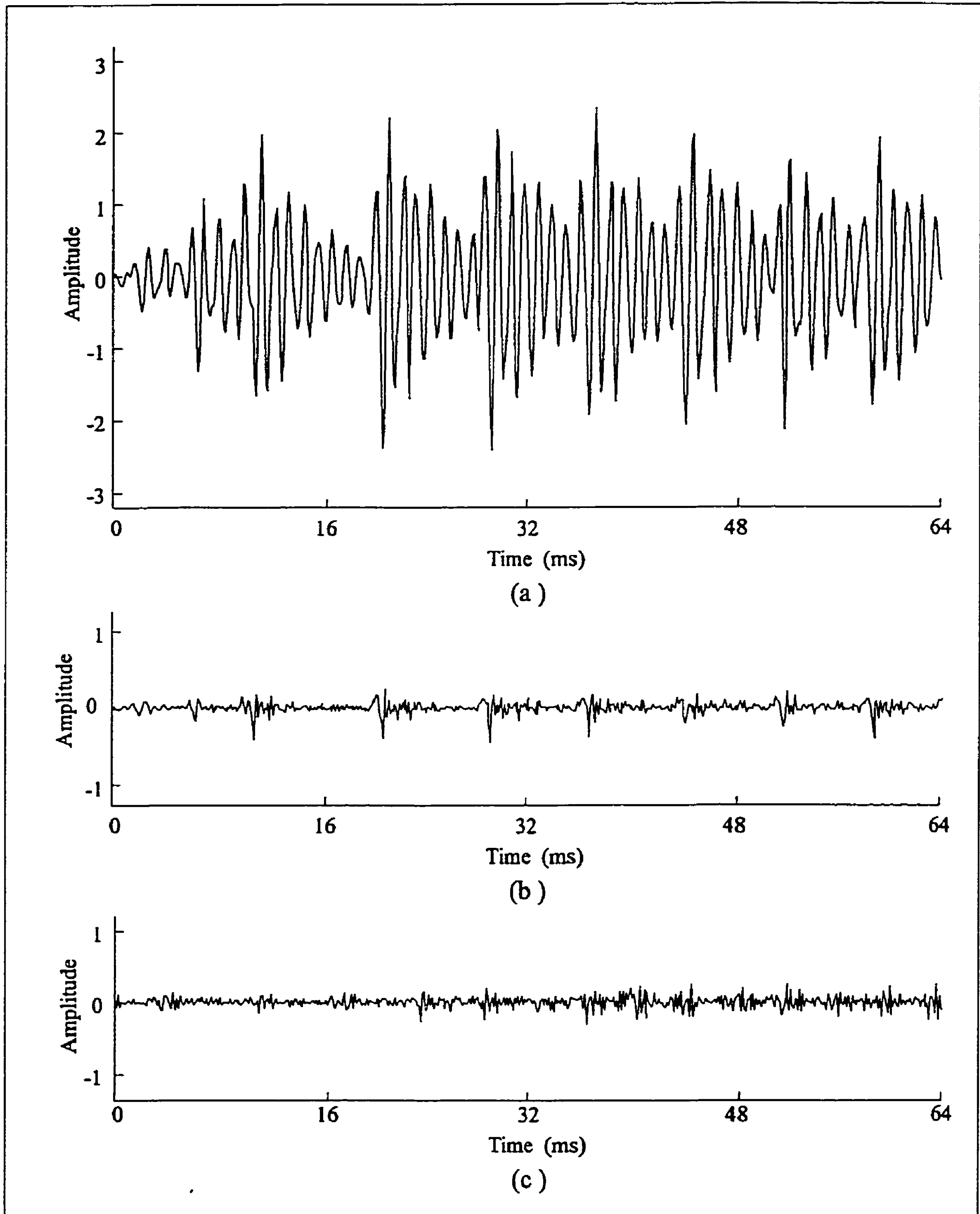


Fig. 2.6. Time domain plots of (a) original speech, (b) LP residual and (c) the pitch residual.

$$E = \sum_{n=0}^{M-1} [u(n) - \frac{\sum_{n=0}^{M-1} u(n) \cdot u(n-\tau)}{\sum_{n=0}^{M-1} [u(n-\tau)]^2} \cdot u(n-\tau-j)]^2 \tag{2.32}$$

Since the range of the pitch lag is known, Eq. 2.29, all the integer values in this range can be substituted into Eq. 2.32. The one that gives the minimum mean squared error, is the best estimate for the pitch lag for the frame under analysis. This value is substituted into Eq. 2.31 to obtain the best estimate for the filter coefficient.

Figure 2.6 illustrates how inverse LTP filtering further whitens the residual. This is to be expected since the LTP inverse filter removes the long-term correlations.

2.3.5 Pre-emphasis and Post-emphasis

The Linear Predictive Coding (LPC) model presented here has a post-emphasis filter as shown in Fig. 2.3. This filter represents the combined effect due to the glottal spectrum and lip radiation. This filter can be modelled as a low-pass filter with a spectral slope ranging from minus four to minus twelve dB/octave [Swan87]. Typically, it is approximated by a low-pass all-pole time-invariant filter with minus six dB/octave roll-off.

The inclusion of the post-emphasis filter reduces the order required for the LP filter to achieve a given level of speech quality. This is because the post-emphasis filter relieves the

LP filter from having to model the effect of the glottal and lip radiation on the frequency. The post-emphasis filter has to be used in conjunction with a pre-emphasis filter applied on the original speech before LP analysis. The pre-emphasis filter is the inverse of the post-emphasis filter.

In most applications, the pre-emphasis filter is implemented as a first order high-pass filter. Eq. 2.33 gives the difference equation.

$$y[n] = x[n]-u\cdot x[n-1] \tag{2.33}$$

where

$$u = \frac{R(0)}{R(1)} \tag{2.34}$$

and R is the autocorrelation function defined in Eq. 2.14. The corresponding post-emphasis filter has the form

$$y[n] = x[n]+u\cdot y[n-1] \tag{2.35}$$

Equation 2.34 gives a filter coefficient, u, which varies from frame to frame. It tends to be close to one for voiced speech due to higher degree of correlation. For unvoiced speech this parameter is very small thus making the effect of the pre-emphasis and post-emphasis filters negligible.

In most systems, the pre-emphasis filter has a constant value for the parameter u. This value lies in the range 0.9 to 1.0. This means that one less parameter has to be transmitted

to the synthesizer. The disadvantage is that the parameter is not optimized for unvoiced speech.

2.4 Summary of Chapter II

This chapter presented linear prediction theory as it is used for the modelling of speech in digital systems. Linear predictive coding of speech is based on the excitation plus modulation speech production model. The excitation is considered as being either voiced or unvoiced. Unvoiced excitation is noise-like whereas voiced excitation is periodic. LP analysis works by first deriving the vocal tract model parameters; the LPC coefficients. An inverse filter based on these coefficients is used to remove the modulation added to the excitation by the vocal tract. Pitch prediction is used to model the periodicity introduced by the vibration of the vocal cords. Equations for solving for the LPC coefficients and the pitch prediction coefficients were presented in this chapter.

Chapter 3 presents iterated function system (IFS) theory. In chapter 4, IFS are used to model the residual of the long term predictor.

CHAPTER III

ITERATED FUNCTION SYSTEMS

3.1 Introduction

In many areas of science, models of systems are derived from analysis of the input and corresponding output of the systems. The process takes two stages; (i) experimental observation and (ii) data analysis and modelling. In experimental observation, the system under study is subjected to different input states. A record is kept of the output of the system for each input state. In the data analysis phase, the objective is to analyze the experimental data and come up with a mathematical representation of the system. The mathematical representation is in the form of a function that best approximates the data. This function is known as the interpolation function.

This chapter introduces the concept of fractal interpolation. Before discussing fractal interpolation a discussion of the limitations of Euclidean geometry based interpolation methods is presented. Fractal interpolation is then introduced and contrasted with Euclidean geometry based methods.

3.2 Traditional Interpolation Functions

Suppose we observe a system that has a single input x and a single output y. From experimental observation, we obtain a collection of data in the form

$$\{(x_i, y_i): \quad i = 0, 1, 2, \ldots, N\} \tag{3.1}$$

where N is a positive integer, x_i are real numbers such that $x_0 < x_1 < x_2 < .... < x_N$. These data can be represented graphically as a subset of $\mathbb{R}^2$ plotted on the Cartesian plane. A polynomial of as low degree as possible whose graph provides a good fit to that of the data is sought. A single polynomial function might be used for the entire interval $[x_0, x_N]$ or a set of polynomials might be used for specific sub-intervals of $[x_0, x_N]$. Instead of using polynomials only, linear combinations of elementary functions such as sine, cosine and natural log might be used. The end result is that the data is represented by a classical geometrical entity, which can be represented by a simple mathematical formula.

The elementary functions used in traditional interpolation come from Euclidean geometry. When one sufficiently zooms in on a portion of a graph of one of these functions, the curve looks like a straight line. This means that these curves are differentiable everywhere except at a "few" discontinuities. There are cases in nature whereby this Euclidean based model fails. For instance, traditional interpolation cannot be used to describe functions that are continuous everywhere but not differentiable anywhere. An example of such systems are functions that are self-similar. Zooming into the curves of such functions does not produce straight lines but curves similar in shape to the original. Fractal interpolation provides one possible way of representing these self-similar functions.

3.3 Iterated Function Systems (IFS)

Iterated Function Systems (IFS) provide a method of describing or modelling self-

affine structures. An iterated function system is defined as a finite set of contraction mappings, w_i, defined over a metric space X, with a distance function d [Barn88]

$$\begin{gathered} \{ w_i : X \rightarrow X \; for \; i = 1,2, ...M \} \\ d(w_i(v), w_i(z)) \le s_i d(v,z) \quad for \quad v,z \in X \\ 0 \le s_i < 1 \end{gathered} \tag{3.2}$$

In a two-dimensional Cartesian plane, each affine map has the form

$$w_i(\mathbf{z}) = \begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \end{bmatrix} \tag{3.3}$$

The part of the affine transform with the coefficients e and f gives a translation along the horizontal and vertical axis. The matrix with coefficients a, b, c, d is a linear operator which performs four different transformations; scaling, rotation, reflection and shearing. The degree to which any of these transformations is performed depends on the relative values of the four coefficients.

To illustrate the effect of affine transformations, we take a set of five affine maps:

$$w_1(\mathbf{x}) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} -1 \\ -1 \end{bmatrix} \tag{3.4}$$

$$w_2(\mathbf{x}) = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{3.5}$$

$$w_3(\mathbf{x}) = \frac{\sqrt{2}}{2} \cdot \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{3.6}$$

$$w_4(\mathbf{x}) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{3.7}$$

$$w_5(\mathbf{x}) = \begin{bmatrix} 1 & 0.5 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{3.8}$$

The map in Eq. 3.4 gives a translation. The other maps in Eqs. 3.5, 3.6, 3.7 and 3.8 give a scaling, rotation, reflection and a shear respectively. Figure 3.1 shows the effect of applying each of the affine transformations in Eqs. 3.4 to 3.8 to an object.

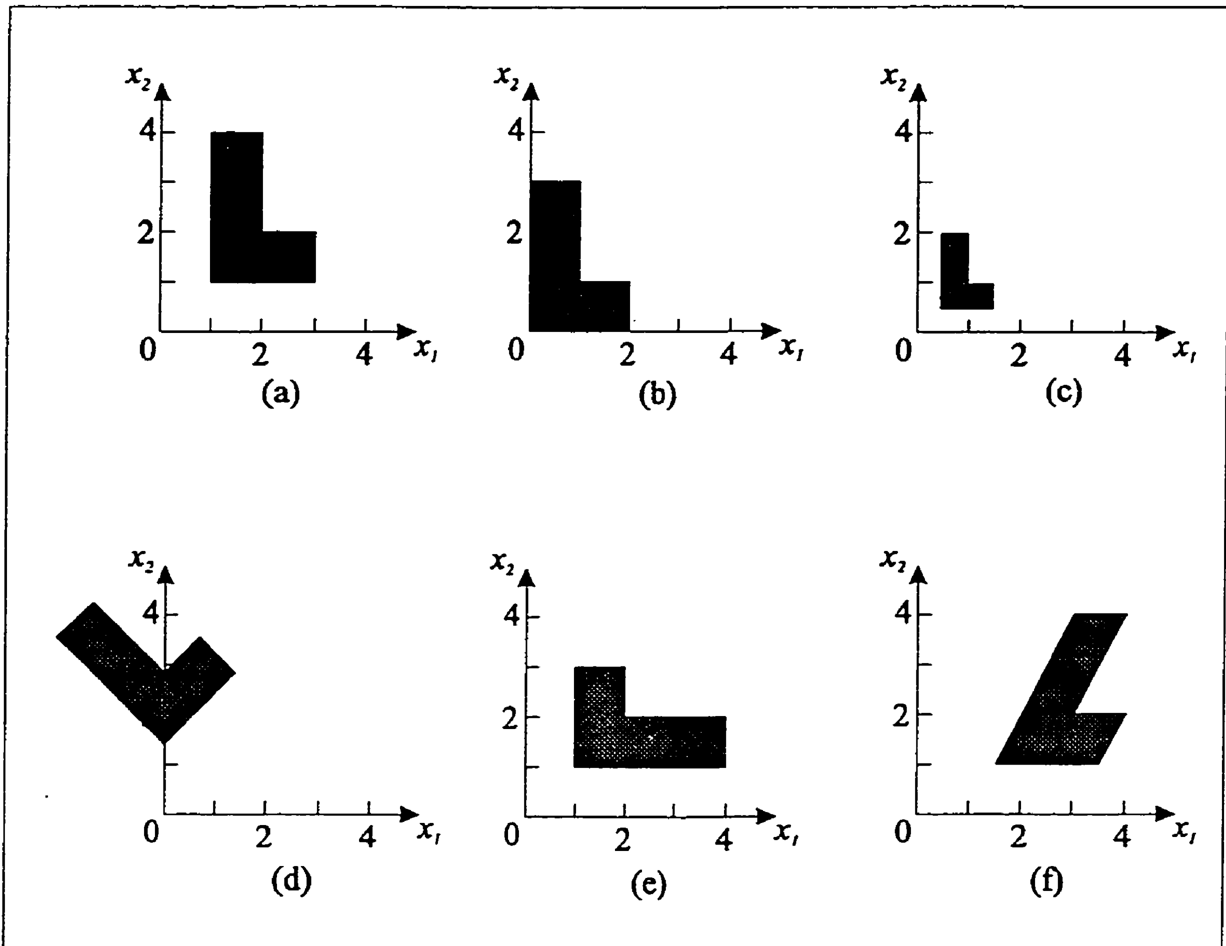


Fig. 3.1. Graphs b, c, d, e, and f show the original object, a, after applying an affine transform that translates, scales, rotates, reflects, and shears respectively.

The affine transforms given as examples above perform only one of the five transformations. Typically an affine transform performs more than one of these transforms. From Eq. 3.2, an iterated function system is a collection of such transforms.

The other requirement that Eq. 3.2 states for IFS is that each affine transform must be contractive with respect to some specified distance function defined on the metric space. If we again consider the same metric space as our previous examples, $\mathbb{R}^2$, the most common

distance metric is the Euclidian metric:

$$d(v,z) = [(v_1 - z_1)^2 + (v_2 - z_2)^2]^{\frac{1}{2}} \tag{3.9}$$

If a transform is contractive in $\mathbb{R}^2$ with the Euclidian metric, then the straight line joining any two points will be shorter after both have been mapped by that transform. The requirement that the affine maps be contractive is to ensure convergence of the IFS. Convergence of IFS is discussed next.

3.3.1 The Attractor of an IFS

So far we have described the components that make up an iterated function system. In this section we introduce concepts that make IFS useful.

Equation 3.2 specifies that an IFS is a collection of contractive affine transforms. Another mapping, W, can be defined based on this set of affine maps.

$$W(x) = \bigcup_{i=1}^{M} w_i(x) \tag{3.10}$$

The mapping W is thus a union of all the affine maps that make up the IFS. It can be shown that the mapping W is a contractive mapping too, and its contraction factor is given by

$$s = \max\{s_i :\ i = 1,2,...,M\} \tag{3.11}$$

Thus W obeys the relationship

$$d(W(\boldsymbol{u}), W(\boldsymbol{v})) \leq sd(\boldsymbol{u}, \boldsymbol{v}) \tag{3.12}$$

where $s<1$ and $\boldsymbol{u}$, $\boldsymbol{v} \in X$. We can thus analyze the iterated function system as a single contractive transform.

We choose an arbitrary set, $\boldsymbol{x}_0$, in the space X in which the IFS is defined. We then create a sequence $(\boldsymbol{x}_n)$ by iteratively applying the contractive transform W.

$$\boldsymbol{x}_1 = W(\boldsymbol{x}_0), \quad \boldsymbol{x}_2 = W(\boldsymbol{x}_1), \quad \ldots \quad \boldsymbol{x}_{n+1} = W(\boldsymbol{x}_n) \tag{3.13}$$

By IFS theory, this sequence is convergent. It converges to a unique set called the attractor of the IFS. Thus, regardless of what the initial set $\boldsymbol{x}_0$ is, the sequence always converges to the same set. This is the beauty of IFS.

3.3.1.1 Proof of IFS Convergence

We are going to provide two proofs. The first one is that the sequence in Eq. 3.13 is indeed convergent. The second one is that the sequence converges to the same limit regardless of what the initial point is.

It can be shown that if we take any two points in the sequence, say x_n and x_m, the following relationship is true

$$d(x_m, x_n) \leq \frac{s^m}{1-s} d(x_0, x_1) \tag{3.14}$$

where s is the contraction factor of W. Since $s<1$, Eq. 3.13 means that if we take any point x_m in the sequence and some arbitrarily small value, $\epsilon>0$, we can find another value x_n such that

$$d(x_m, x_n) < \epsilon \qquad for\ some\ n>m \tag{3.15}$$

Any sequence that satisfies Eq. 3.14 is a Cauchy sequence.

A metric space (X,d) is said to be complete if every Cauchy sequence in X converges. A sequence (x_n) in X converges if it has a limit which is an element of X [Krey78]. This means that the sequence in Eq. 3.13 is convergent if it is defined in a complete metric space.

One of the axioms of a metric, the triangle inequality, states that given a metric space (X,d) and any three vectors v, w, $z \in X$ the following relationship is always true

$$d(v, w) \le d(v, z)+d(z, w) \tag{3.16}$$

We have established that the sequence in Eq. 3.13 has a limit, let us say that limit is x. Let us also choose some arbitrary point in the sequence; say x_m. By the triangular inequality we have the following relationship

$$d(x, W(x)) \le d(x, x_m) + d(x_m, W(x)) \tag{3.17}$$

By Eq. 3.12 and 3.13 we have

$$d(x_m, W(x)) \le d(x_{m-1}, x) \tag{3.18}$$

Substituting 3.18 into 3.17 produces

$$d(x, W(x)) \leq d(x, x_m) + d(x_{m-1}, x) \tag{3.19}$$

Since x is the limit of the sequence (x_n) then the right hand side of Eq. 3.19 can be made arbitrarily small by choosing sufficiently large m. This implies that $d(x, W(x)) = 0$, which in turn implies that $x = W(x)$. Any point that is mapped into itself by some transform is called a fixed point of that transform. The limit of the sequence in Eq. 3.13 is thus a fixed point of the transform W. The next question to ask is; is this fixed point unique.

To prove that the fixed point is unique, we assume that there is at least one more fixed point, v. Since v is another fixed point; then $v = W(v)$. Using Eq. 3.12 and the knowledge that v and x are both fixed points of W, we get

$$d(x, v) = d(W(x), W(v)) \leq s \cdot d(x, v) \tag{3.20}$$

Since $s<1$, then solving Eq. 3.20 gives $d(x,v) = 0$. This in turn implies $x = v$. Thus, x is a unique fixed point.

In iterated function systems, the fixed point of the system is called the attractor for that IFS. In this section we showed that an IFS is guaranteed to converge as long as the contraction factor is less that one, and the IFS is defined in a complete metric space. The IFS converges to an attractor, which is a unique fixed point of the system. The next section shows how the attractor is used in interpolation.

3.4 IFS Based Interpolation

The interpolation problem can be stated as follows: given time series data over a specific interval, find a function, $F(\bullet)$, that best matches the data over the interval. In this section we represent the time series as

$$\{(u_j, v_j): \; j = 0,1,2,.......,N\} \tag{3.21}$$

where $u_j \in \mathbb{R}$ *s.t.* $u_0<u_1<u_2< ... < u_N$, and N is the total number of points in the time series. The time series data could be speech, stock market data, delay in packets being sent over the Internet, etc. The variable u_j is an index that represents the relative time when each measurement, v_j, is taken. Thus for stock market data u_j could represent days and v_j could be the daily closing price of a specific stock. N represents the length of the period over which the analysis is being done.

The interpolation task involves finding a function that best fits the given data, over the interval $[u_0, u_N]$. In this section we discuss two models of iterated function systems and their application in approximating functions of a given time series data. The two models are the self-affine model and the piece-wise self-affine model.

3.4.1 Self-Affine Model

The self-affine fractal model is an iterated function system whose attractor is self-affine over the interval $[u_0,\; u_N]$. This means that portions of the series may be obtained by applying affine transforms on the whole series. This section shows how to find a self-affine

IFS whose attractor closely approximates the graph of the time series of Eq. 3.21.

The definition of an IFS was given in Eq. 3.2. It consists of M affine transforms. In the self-affine model the time is divided into M sections. Each one of the M affine transforms maps the whole series , the interval $[u_0, u_N]$, into one of these sections. We call the section onto which a map scales the whole signal the range space of the map.

The range spaces of the M maps are non-overlapping. Thus if the map w_i maps the whole signal to the range $[x_{i-1}, x_i]$ then the map w_{i+1} maps the whole signal to the range $[x_i, x_{i+1}]$. The endpoint of the range of each map is called the interpolation point of that map. The interpolation point for the map w_i is thus denoted (x_i, y_i). Since every map has an interpolation point associated with it, we thus have a set of interpolation points given by

$$\{(x_i,y_i):\ i = 0,1,2,......M;\ M \le N\} \tag{3.22}$$

Eq. 3.22 defines M+1 interpolation points even though we only have M maps in the IFS. The first interpolation point, (x_0, y_0) does not have a map associated with it. It is needed for defining the interval for w_0, the first map. The set of interpolation points is a subset of the points in the time series. As the time series has N data points, the number of interpolation points , M cannot be more than N. To ensure complete coverage of the interval represented by the time series, the first and last interpolation points are required to be equal to the first and last points of the given data set. Thus

$$(x_0, y_0) = (u_0, v_0) \qquad and \qquad (x_M, y_M) = (u_N, v_N) \tag{3.23}$$

As mentioned earlier, each map, w_i, maps data over the whole interval, $[x_0, x_M]$, into its range space, $[x_{i-1}, x_i]$. This requires that

$$w_i\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix} \qquad \text{and} \qquad w_i\begin{bmatrix} x_M \\ x_M \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{3.24}$$

At the beginning of this chapter, the general form of each affine transform was given in Eq. 3.3. For the case of time series data, this equation is slightly simplified. Since time series data is single valued, to ensure that the IFS only generates single valued data the coefficient b, is set to zero [Vine93]. This simplifies the equation to

$$w_i\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_i & 0 \\ c_i & d_i \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \end{bmatrix} \tag{3.25}$$

Each one of the affine maps is a two dimensional mapping. It is a contraction in both x and y directions. There is a contraction in the x-direction since each map transforms points in the interval $[x_0, x_M]$ to the smaller interval $[x_{i-1}, x_i]$. The contraction in the y-direction is controlled by the parameter d_i, of Eq. 3.25. The map will be a contraction mapping if $|d_i|<1$. The parameter d, called the contraction factor, is independent of the interpolation points and is used to control the shape of the interpolating function [Vine93].

Given time series data, the problem of finding the IFS that best represents that data is called the inverse problem. If we know the interpolation points, then for each map we only need to find the contraction factor, d_i. Given contraction factors, and the interpolation points,

(x_i, y_i), the rest of the coefficients for each map can be calculated by substituting Eqs. 3.24 into 3.25. This gives four linear equations with four unknowns; a_i, e_i, c_i, and f_i. The problem of finding the parameters of each map thus reduces to a problem of solving for two parameters; the contraction factor d_i and the interpolation point x_i. Details of an inverse algorithm are given in Section 3.7.3.

3.4.2 Piecewise Self-Affine Model

The self-affine model described in the previous section, consists of affine maps, $\{w_i\}$, that map the entire function $[u_0, u_N]$, to a portion of the function in the interval $[x_{i-1}, x_i]$. The resultant function is thus self-affine throughout its domain. The piecewise self-affine model generalizes this concept. Instead of imposing the condition that each affine map should have the same domain as the function, affine maps may have domains which are subsets of the function's domain. The whole function is thus composed of sections which are affine transforms of other sections of the function. Such a function is called piecewise self-affine. Sections of the function are not self-affine with the whole function as in the self-affine model. They are self-affine with specific portions of the function.

In the self-affine model described in Section 3.4.1, each affine map has an interpolation point associated with it. The interpolation point is needed to define the range space of the map and its domain is assumed to be the same as that of the function. In the piecewise self-affine model, maps may have domains which are subsets of the function. We therefore require for each map a set of two points that define its domain. In keeping with

Mazel and Hayes [MaHa92], we shall call these points the addresses associated with each map. The set of addresses for all the maps is described as:

$$\{ \ (u'_{i,j}, v'_{i,j}); \quad i=1,2,\ldots,M; \quad j=1,2 \ \} \tag{3.26}$$

Thus the map w_3 would have the addresses $(u'_{3,1}, v'_{3,1})$ and $(u'_{3,2}, v'_{3,2})$ associated with it. The endpoint constraints given in Eq. 3.24 were based on affine transforms whose domain space was that of the function. As the piece-wise self-affine model has a different domain for each map; the endpoint constraints for each map, w_i, are thus given by:

$$w_i \begin{bmatrix} u'_{i,1} \\ v'_{i,1} \end{bmatrix} = \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix} \quad \text{and} \quad w_i \begin{bmatrix} u'_{i,2} \\ v'_{i,2} \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{3.27}$$

In the self-affine model, the inverse problem was reduced to finding the interpolation points and the corresponding contraction factor for each map. In the piece-wise model, the inverse problem involves solving for the interpolation points, the map addresses and the contraction factor. Once these values are known, the rest of the parameters may be obtained by solving the linear equations formed by substituting Eqs. 3.27 into 3.25. A solution for the inverse problem is presented in Section 3.7.4.

3.5 Synthesis of an IFS Attractor

In the last section, a description of an IFS was provided. Given an IFS, how do we find the interpolation function that it represents. This section describes how this interpolation function, also known as the attractor of the IFS, may be generated.

As was shown in Section 3.3.1, the attractor of an IFS is the unique fixed point G such that

$$W(G) = \bigcup_{i=1}^{M} w_i(G) = G \tag{3.28}$$

In other words, the set G is mapped onto itself by the IFS. There are two techniques for generating the IFS; the deterministic technique and the random iteration technique [Barn88].

In the random iteration, the iteration is started by selecting one of the interpolation points. A map is then chosen at random and applied to this point. The resulting point is plotted. Another map is chosen at random and applied to this new point to give yet another point. Every time a new point is obtained it is plotted and a map is randomly selected and applied to it to give a new point. This method will work for the self-affine model but not for the piecewise model. We therefore discuss the deterministic method in more detail because it can be used for both methods.

The deterministic method is based on the property that if enough iterations are performed on a given non-empty set A_0, using a set of affine maps, then the result is the attractor of the IFS that consists of that set of affine maps. We define the result of the nth

iteration on A_0 as

$$A_n = \bigcup_{i=1}^{M} w_i(A_{n-1}) \tag{3.29}$$

The only requirement on A_0 is that it should be non empty single valued function in the x×y plane. The iteration on A_0 produces a sequence of sets $\{A_0, A_1, A_2, \dots\}$ which, according to the proof provided in Section 3.3.1, converges to the attractor of the IFS. During the iteration, we ascertain that the attractor has been found when $A_{n+1}=A_n$. The important steps of the synthesis algorithm are as follows:

i. Select any non-empty set and call it A_0
ii. Use Eq. 3.29 to obtain the next approximation to the attractor. Call it A_{n+1}
iii. If $A_{n+1} = A_n$ go to (iv) otherwise go to (ii)
iv. The attractor, $G=A_n$.

We have presented in this section a method of generating the attractor, or interpolation function, associated with an IFS. The question of how we find the IFS that best matches the underlying function of a given time-series has not yet been addressed. This question is the inverse problem. Before we discuss the inverse problem, the next section presents a theorem that provides a means of measuring the goodness of fit of an attractor associated with an IFS and a given function or time-series.

3.6 The Collage Theorem

The Collage theorem provides a way of assessing how well the attractor of an IFS approximates a given function. The theorem is as follows:

Let (X,d) be a complete metric space. Let H be a given function in X. Let ϵ be a given scalar quantity greater than or equal to zero. Choose an IFS system with contraction factor s, where $0 \leq s < 1$, such that

$$d(H, \bigcup_{i=1}^{M} w_i(H)) \leq \epsilon \tag{3.30}$$

Then

$$h(H,G) \leq \frac{\epsilon}{1-s} \tag{3.31}$$

where G is the attractor of the IFS. A proof of the Collage theorem is given in [Barn88]. When solving the inverse problem, we need to find an IFS that minimizes the distance between its attractor and the underlying function of the given time-series. In other words, we need to minimize $h(H,G)$ where H is the time series under analysis. This means that an inverse algorithm would have to identify a possible IFS candidate, generate its attractor and then compute its distance from the function H. The collage theorem offers a shortcut which, as we shall show later on in this chapter, significantly reduces the computational complexity of inverse algorithms.

Equations 3.30 and 3.31 in the Collage theorem imply that a minimization problem

in $h(H,G)$ is equivalent to a minimization problem in $d(H,W(H))$. Formulating the inverse problem as a minimization problem in $d(H,W(H))$ simplifies the inverse problem.

3.7 IFS Inverse Algorithm

The IFS inverse problem can be stated as follows: given the time series of Eq. 3.21 find an IFS whose attractor best matches the time series. The minimal set of parameters that define an IFS are the contraction factors, d_i, and the interpolation points, (x_{i-1}, x_i), in the case of the self-affine model. In the case of the piecewise self-affine model, the addresses of each map are also required. One method of finding these parameters is an exhaustive search over all possible combinations of the parameters. Unfortunately, this is an NP-complete problem. Alternative algorithms are therefore required.

In this section, two solutions for the inverse algorithm developed by Mazel and Hayes are presented [MaHa92]. One is for the self-affine model and the other is for the piecewise self-affine model. First we show how the rest of the IFS parameters are calculated given the minimal set of parameters. We then derive a least squares formula for calculating the contraction factor given the interpolation points and addresses for each map. The inverse algorithms are then presented.

3.7.1 IFS Map Minimal Parameter Set

In this section and Section 3.7.2 we will make derivations based on the piecewise model. Since the self-affine model can be considered a special case of the piecewise self-

affine model, the formulas obtained can be applied to both. To make the equations a little easier to read and write, we are going to change the notation. The change will only be for this section and Section 3.7.2. Table 3.1 summarizes the change in notation.

Table 3.1. Change in notation to make equations easier to read.

Entity	Notation Elsewhere	Notation for Derivations
Time series	$\{(u_j, v_j): j=0,1,2, \ldots N\}$	$\{(n, y_n): n=0,1,2, \ldots N\}$
The ith map in an IFS	w_i	w
Interpolation points	(x_{i-1}, y_{i-1}) and (x_i, y_i)	(p, y_p) and (q, y_q)
Address points of ith map	$(u'_{i,1}, v'_{i,1})$ and $(u'_{i,2}, v'_{i,2})$	(I, y_I) and (F, y_F)

For the time series, the x-coordinate is given by an index n which ranges from zero to $(N-1)$. N is the size of the analysis window for the time series. The y-coordinate is depicted as a function of the index n. This is done for all points, thus if the x-coordinate of a point is p, then its y-coordinate is y_p. We also drop the subscript i that depicts the ith map and its parameters.

Now that we have defined the notation used in this section, we shall proceed to show how all the parameters required to construct the attractor of an IFS can be obtained . The minimal set of parameters was identified to be

(i) the contraction factor d

(ii) the interpolation points (p, y_p) and (q, y_q) and

(iii) the address points (I, y_I) and (F, y_F)

By substituting these values into Eq. 3.25, the equation of an affine transform, as well as into

Eq. 3.27, the end point constraints, we get a system of four linear equations. The unknowns are the four parameters, a, c, e, and f. Solving the equations yields the following formulas

$$a = \frac{q-p}{F-I} \tag{3.32}$$

$$b = \frac{F\cdot q - I\cdot p}{F-I} \tag{3.33}$$

$$c = \frac{(y_q - y_p) - d(y_F - y_I)}{F-I} \tag{3.34}$$

$$f = \frac{(F\cdot y_q - I\cdot y_p) - d(F\cdot y_I - I\cdot y_F)}{F-I} \tag{3.35}$$

We have thus shown that the minimal set as defined is sufficient to represent each map in an IFS. The inverse algorithm thus needs only search for these parameters.

3.7.2 Calculation of the contraction factor

The last item that we have to address before presenting the inverse algorithm is a problem that may be formulated as follows: given (i) a time series $\{(n, y_n): n=0,1,2, \ldots N\}$

(ii) the interpolation points (p, y_p) and (q, y_q) and (iii) the address points (I, y_I) and (F, y_F); find the contraction factor of the map, in an IFS system, that gives the attractor the best match to the time series in the interval $[p, q]$. A least squares fit approach is presented.

We start by defining a new point (j, y'_j), which is the result of mapping the point (n, y_n) by the map w. Thus

$$\begin{bmatrix} j \\ y'_j \end{bmatrix} = w\begin{bmatrix} n \\ y_n \end{bmatrix} = \begin{bmatrix} a & 0 \\ c & d \end{bmatrix}\begin{bmatrix} n \\ y_n \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} \tag{3.36}$$

The x-coordinate is defined as

$$j = int(a \cdot n + e) \tag{3.37}$$

where the int() function represents either a truncation or rounding to the nearest integer. Since w maps the larger interval of width $(F-I+1)$ to the smaller interval of width $(q-p+1)$, for the same x-coordinate j, we may get more than one y-coordinate. The y-coordinate y'_j is defined as the average value of all the y-coordinates of all points whose x-coordinate is j.

The next thing is to find the parameter d that minimizes the approximation error between the attractor of the resultant IFS and the time series in the interval $[p, q]$. The advantage of using the Collage theorem is that one does not have to first calculate the attractor in order to minimize the error. The mean squared error function is thus given by

$$E = \sum_{j=p}^{q} (y'_j - y_j)^2 \tag{3.38}$$

From Eq. 3.37 we note that for j to range from $j=p$ to $j=q$; the index n ranges from $n=I$ to $n=F$. We thus rewrite Eq. 3.38 in terms of the index n.

$$E=\sum_{n=I}^{F}(y'_j-y_j)^2 \tag{3.39}$$

The variable y'_j is the result of mapping the point $(n; y_n)$ and, from 3.35, can be written as

$$y'_j=c\cdot n+d\cdot y_n+f \tag{3.40}$$

If we substitute parameters c and f in Eq. 3.40 with Eqs. 3.34 and 3.35 we can express y'_j in terms of the minimum parameter set as in 3.41.

$$y'_j=\frac{y_p-y_q}{F-I}\cdot n+\frac{Fy_p-Iy_q}{F-I}-d\cdot\left(\frac{y_f-y_I}{F-I}\cdot n+\frac{Fy_I-Iy_F}{F-I}-y_n\right) \tag{3.41}$$

For convenience, we define the following variable

$$\xi_n=\frac{F-n}{F-I} \tag{3.42}$$

This simplifies 3.41 to

$$y'_j=y_q\cdot(1-\xi_n)+y_p\cdot\xi_n-d\cdot(\xi_n\cdot y_I+(1-\xi_n)\cdot y_F-y_n) \tag{3.43}$$

Substituting into Eq. 3.39 gives

$$E=\sum_{n=I}^{F}[y_q\cdot(1-\xi_n)+y_p\cdot\xi_n-d\cdot(\xi_n\cdot y_I+(1-\xi_n)\cdot y_F-y_n)]^2 \tag{3.44}$$

We again define variables to make the equations more manageable.

$$\alpha_n=y_n-[\xi_n y_I+(1-\xi_n)y_q] \tag{3.45}$$

$$\beta_n=y_j-[\xi_n y_p+(1-\xi_n)y_q] \tag{3.46}$$

This simplifies the error function to

$$E=\sum_{n=I}^{F}(\alpha_n d-\beta_n)^2 \tag{3.47}$$

The error function can be minimized with respect to d by taking a partial derivative of Eq. 3.47 and equating it to zero. This yields Eq. 3.48 which is the formula for calculating the contraction factor when the following are known (i) time series data, (ii) the interpolation points (p, y_p) and (q, y_q), and (iii) the address points (I, y_I) and (F, y_F).

$$d=\frac{\sum_{n=I}^{F}\alpha_n\beta_n}{\sum_{n=I}^{F}\alpha_n^{\,2}} \tag{3.48}$$

Now that we have established a formula for calculating the contraction factor we will present the inverse algorithms.

3.7.3. Self-Affine Model

When the minimal set that is needed to describe each map in our IFS system was described earlier, it included the contraction factor, the interpolation points and the address points. In the self-affine model, all the maps have the same address points because they all have the same domain, $[0, N\text{-}1]$, as the entire interpolation function. The inverse algorithm only needs to find the optimal contraction factors and interpolation points. The algorithm is presented in Fig. 3.2.

i. Choose the point (u_0, v_0) from the time series as the interpolation point (x_0, y_0). Set the map index i to 0.
ii. Increment the map index i by one and set the trial map index k to 0.
iii. Increment the trial map index k by one. (We are now trying the kth trial map.) Take its interpolation point to be the point in time series after the interpolation point of the last trial map if $k > 1$. If $k = 1$ take it to be the point after the interpolation point x_{i-1}. Call this point (x_k, y_k).
iv. Calculate the contraction factor d for the kth trial map.
v. If $|d| \geq 1$ go to (iii) else go to (vi).
vi. Compute the parameters, a, c, e, and f to form the trial map w_k. Apply this map to each point of the function to yield $w_k(H)$.
vii. Take a subset of the original function between x_{i-1} and x_k and call it H_k. Compute and store $E = d(H_k, w_k(H))$.
viii. Repeat (iii) - (vii) until $x_k = u_N$.
ix. The trial map that produced the least error in (viii) is the best candidate. Save its interpolation point and contraction factor as (x_i, y_i) and d_i respectively. If more than one trial map produces the least error, then the one with the largest x is chosen.
x. Go to (ii) and continue until $x_i = u_N$.

Fig. 3.2. Inverse algorithm for the self-affine fractal model.

3.7.4. Piecewise Self-Affine Model

The inverse algorithm for the self-affine model searched for interpolation points and the corresponding contraction factor. The piecewise model also requires the address intervals to be determined by the inverse algorithm. In this algorithm, constraints are imposed on the map parameters in order to simplify the search algorithm. The distance between the x-coordinates of both the interpolation points and the address points are required to be constant. Thus if δ is the distance between interpolation points and ψ is the distance between address points then

$$x_i - x_{i-1} = \delta \qquad i = 1, 2, \ldots, M \tag{3.49}$$

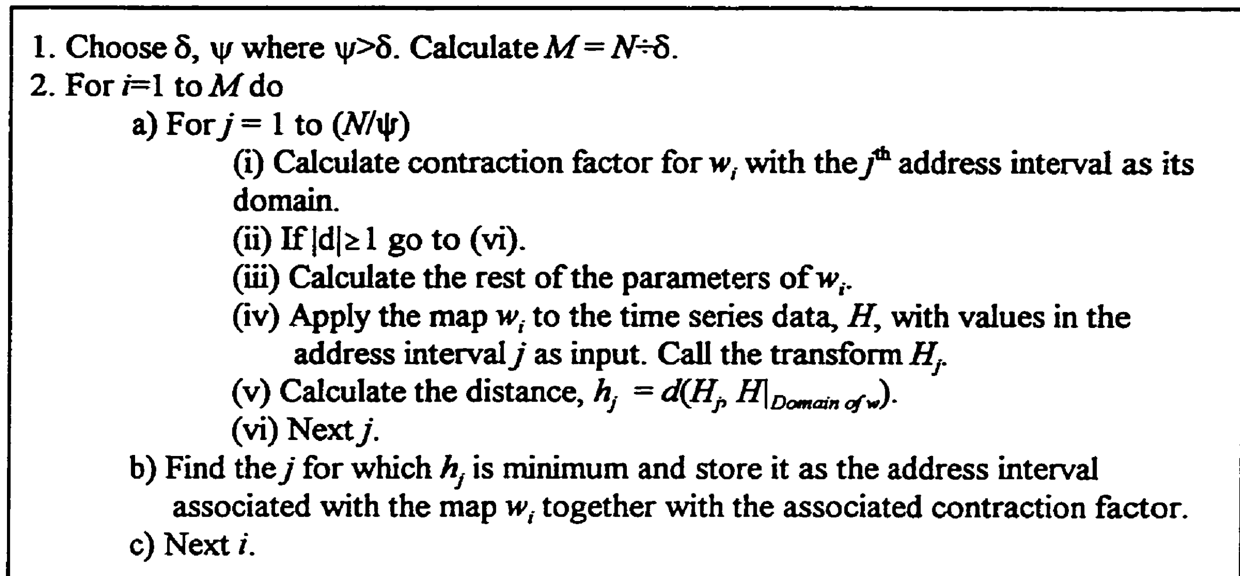
and

$$u`_{i,2} - u`_{i,1} = \psi \qquad i = 1, 2, \ldots, M \tag{3.50}$$

Since there should be contraction in the horizontal direction it is required that $\psi > \delta$. The values of δ and ψ have to be chosen by the designer either by experiment of from knowledge of the data under analysis. The IFS will have as many interpolation sections as there are maps but the number of address intervals will be less than the number of maps. This means that some of the maps will share address intervals. With the restrictions stated, the Mazel and Hayes inverse algorithm is given in Fig. 3.3.

It is important to point out that this algorithm does not determine the best IFS system for the given data. This is because the search space is limited by the restrictions placed on ψ

and δ. What the algorithm does provide is the optimal match between the given interpolation sections and the given address intervals.



1. Choose δ, ψ where ψ>δ. Calculate $M = N \div \delta$.
2. For i=1 to M do
    a) For j = 1 to (N/ψ)
        (i) Calculate contraction factor for w_i with the jth address interval as its domain.
        (ii) If $|d| \geq 1$ go to (vi).
        (iii) Calculate the rest of the parameters of w_i.
        (iv) Apply the map w_i to the time series data, H, with values in the address interval j as input. Call the transform H_j.
        (v) Calculate the distance, $h_j = d(H_j, H|_{Domain\ of\ w})$.
        (vi) Next j.
    b) Find the j for which h_j is minimum and store it as the address interval associated with the map w_i together with the associated contraction factor.
    c) Next i.

Fig. 3.3. Inverse algorithm for the piecewise self-affine model.

3.8 Summary of Chapter III

This chapter introduced fractal interpolation based on iterated function systems. The iterated function system model was discussed with particular attention to its use in modelling time series data. Algorithms developed by Mazel and Hayes for solving the inverse problems were presented. The next chapter presents a new approach for modelling excitations with IFS in linear predictive coding of speech.

CHAPTER IV

FRACTAL MODELLING OF SPEECH

4.1 Introduction

This chapter develops a system for modelling of linear prediction (LP) residual using iterated function systems. A low bit-rate speech codec based on fractal modelling is also developed. Fractal modelling refers to the use of fractals to characterize signals or systems. Before fractal modelling of the LP residual is presented, a review of other research in the application of fractals to speech coding is presented.

4.2 Fractal Modelling of Speech

Fractals have been applied to the modelling of speech. In this section we review some of these applications and subsequently propose a new approach of using fractals in the modelling of speech.

4.2.1 Some Application of Fractals to Speech Modelling.

Most of the work done in the area of fractal modelling of speech is based on waveform coding. Fractal interpolation functions are used to represent the waveform of the speech signal. Sampled speech signals are windowed and analyzed to find IFS that best represent them. The speech signal is then generated as the attractor of the IFS.

In one such application of IFS to speech coding Mazel and Hayes used the piecewise self-affine fractal model described in Chapter 3. They were able to obtain a compression ratio of 6:1 with a signal to noise ratio of 17dB [MaHa92]. Similar approaches have been presented in [ZhCT94]. Vehel et al make a variation on this approach by using sinusoidal iterated function systems and report that very good functional representations of speech were obtained [VeDL94].

One issue that is raised in fractal modelling of speech is whether voiced speech and unvoiced speech should be represented by the same model. The waveform for the voiced sections generally looks smoother and sinusoidal. The unvoiced sections look more noise-like. Daoudi and Vehel present a system that makes those distinctions [DaVe95]. The system is based on local regularity analysis as measured by the Holder exponent. For voiced parts the signal is smooth except at a few point of sharp variations. The wavelet maxima method of Mallat and Zhong[MaZh92], is used to determine Holder exponents for these isolated singularities. For the unvoiced sections, the signal is considered to be irregular at all scales and thus the local regularity has to be controlled not only at isolated points but everywhere. This is achieved by using IFS to control the local regularity [DaVe95][DaLM95].

Another class of techniques for modelling speech is based on modelling speech using chaotic dynamical systems. One such technique was developed by Langi and Kinsner and applied to the characterization of speech consonants [LaKi95]. The scheme employs correlation fractal dimension and Takens embedding theorem to measure the fractal dimension of the excitation. Later work by Kubin show that vowels can also be characterized using chaotic dynamical systems [Kubi97]. Maragos and Potamianos were able to reduce the error

rate of an Markov model based speech recognition system by 18%. This was done by adding a fractal feature vector to augment the standard feature vectors of energy and cepstrum [MaPo97].

A technique more closely related to the one developed in this thesis was developed by Lupini and Cuperman [LuCu92]. They investigated a method for improving CELP codecs at low bit-rates. Code Excited Linear Prediction (CELP) codecs are made up of three main parts

i) a short term predictor (LPC filter)

ii) a long term predictor for pitch modelling

iii) a stochastic codebook for the excitation.

Lupini and Cuperman argued that CELP coders degraded in quality significantly at low data rates mainly because of the inaccurate representation of the excitation by the white stochastic model. Instead of characterizing the excitation signal as white noise, they investigated an approach introduced by Maragos and Young [MaYo90], which characterizes the excitation as 1/f type of fractal signal. These signals can be characterized as having a power spectral density of the form

$$S(f) = c \cdot f^{-\beta} \tag{4.1}$$

where c is a constant. In the investigation, the parameter β of a residual signal was calculated. A codebook of residuals with the same power spectral density but with

randomized phases, was then searched to find the best candidate vector. The results of this investigation demonstrated only marginal improvements over using a fixed stochastic codebook. Our approach characterizes the excitation as 1/f fractal signals as well. The 1/f class of fractal signals is a family of statistically self-similar random processes. They have statistics in the time domain [Worn96]. The parameter β of Eq. 4.1 which is related to the fractal dimension provides useful but limited information when modelling the signal. The results of Lupini and Cuperman show that this information is not sufficient for giving significant improvement of speech quality in CELP coders. We thus propose using iterated function systems (IFS) for modelling the excitation. Unlike simply using β, which only preserves the spectral tilt, IFS are able to preserve the phase information as well. IFS should therefore provide a better model for speech excitations.

4.3 Map Coverage Cost Function

Section 3.7.3 presented an algorithm for solving the inverse problem for the self-affine IFS model. In this section a modification to that algorithm is proposed. The proposed modification gives the designer more control over the solution obtained by the inverse algorithm.

4.3.1 Review of Terminology

Before we describe the proposed changes we will present a very brief review of some of the terms that will be used. The interpolation problem was described in Section 3.4 as the problem of finding a function that best matches given time series data. Iterated function systems can be used to approximate such time series data. In the self-affine fractal model, an IFS consists of a set of M affine maps which are used to iteratively map the whole set of data points into a subset of the data points. Section 3.4.1 described this subset of data points as the range space of the map w_i. Since the range spaces of the M affine maps are non-overlapping, they can be sufficiently defined by the set of interpolation points, $\{(x_i, y_i)): i = 0,1,2 \ldots M\}$ as given in Eq. 3.22. Thus each map w_i takes all the points in the time series and maps them into the interval $[x_{i-1}, x_i]$. We define the value (x_i-x_{i-1}) as the map coverage of the ith affine map.

4.3.2 Motivation

The inverse algorithm presented in Section 3.7.3 was applied to LPC residual signals. It was discovered that the map coverage for each map in the resultant IFS was very small; less than two samples. It was conjectured that using the given algorithm on any data which is only statistically self affine similar results would be obtained.

Having a small map coverage is not very useful for signal compression applications. Small map coverage values mean that there are more maps in the IFS. Having more maps results in more parameters required to model the system. Increasing the number

of parameters generally reduces the compression ratio. In signal compression it is sometimes desirable to have a sub-optimal solution if it results in greater compression ratios.

The reason for obtaining small map coverage values lies in steps (vii) and (ix) of the algorithm in Fig. 3.2. Step (vii) calculates the approximation error of a given trial map and step (ix) selects the map with minimum error. The error function is such that if the data is not strictly self affine then the resulting solution could very well use the entire set of points in the time series as its interpolation points. We call this the trivial solution because it provides the best "interpolation" function for any data. The modifications proposed in this section provide a means to not only avoid the trivial solution but also allow the designer to control the average map coverage.

4.3.3 Error Normalization.

The first change proposed is normalization of the error. We redefined the approximation error in step (vii) to be

$$E_0 = \frac{h(H_k, w_k(H))}{x_k - x_{i-1}} \tag{4.2}$$

We illustrate the importance of this change with an example. Suppose for a certain data set, there are three trial maps as shown in Table 4.1.

Table 4.1. Set of example trial maps with their interpolation point, approximate error, E and normalized approximation error, E_0.

Trial Map	x_{k-1}	x_k	E	E_0
w_A	0	5	20	4
w_B	0	10	40	4

The approximation error, E, and normalized approximation error, E_0, for each of the trial maps is shown in the table. Using the inverse algorithm described, w_A and w_B could be among the candidates for the ith map. Using the original error measure, w_A would be chosen over w_B because its approximation error is only 20 as opposed to 40. On the other hand, if the normalized error is used, map w_B would be chosen instead. If the main target is to have high compression ratios map w_B is the better choice because it would result in fewer maps.

This section introduced normalization of the error function used in selecting the best candidate maps. Normalization allows comparison on a per sample basis as opposed to comparing errors on sections of different length.

4.3.4 Linear Coverage Cost Function

It is desirable to have maps with fairly large coverage. In the previous section, map coverage was increased by normalizing the error function. This produced less maps per IFS with little or no increase in approximation error.

In some cases, it might be required to reduce the number of maps by more than what is achieved by normalizing error. This can be achieved by modifying the error function

even further. For this purpose we introduce the coverage cost function. It forgives some of the error of large coverage maps while having little or no forgiveness for small coverage maps. The new error function thus becomes a combination of the normalized approximation error and the coverage cost function as in Eq. 4.3.

$$E = E_0 - g(z) \tag{4.3}$$

where z is the coverage of the map whose error is being calculated and

$$g(z) = b \cdot z \tag{4.4}$$

To forgive large coverage maps, the coverage cost function was used in the inverse algorithm for modelling speech excitations. The constant b was varied in an effort to obtain optimal results, It was found that, depending on the size of b, the map width was either too large or too small. It was not possible to obtain more intermediate values for the map width. It was thus decided to search for a cost function that gave more control on the number of maps.

4.3.5 Non-Linear Map Coverage Cost Function

It appears that the most direct method of increasing control over the solution of the inverse algorithm is by manipulating the error function. In the previous section, a linear map coverage cost function was added to modify the error. This simple cost function was found to be ineffective because it still did not give the designer control over the number of maps per IFS.

Many different non-linear cost functions were experimented with in an attempt to find one that provides the designer with the most control. The main characteristic required of each function was that its curve should rise very steeply at small map coverage and then level off at large map coverages. Most of the cost functions experimented with would work very well for a certain desired average map coverage but would fail when a different desired average map coverage was introduced. It thus seemed like we were doomed to searching for a new cost function every time experimentation with different compression ratios was performed. A universal cost function that would expedite experimentation was required.

An excellent cost function candidate was developed in 1913 in the field of enzyme chemistry by Leonor Michaelis and Maud Menten. The equation they developed, known as the Michaelis-Menten equation is used to model kinetics of many enzyme reactions [Stry88]. After changing the variables to meet our application we propose the following equation as the map coverage cost function:

$$g(z) = P_{max} \cdot \frac{k \cdot z}{(1-k)z_0 + k \cdot z} \tag{4.5}$$

A graph of this function is shown in Fig. 4.1.

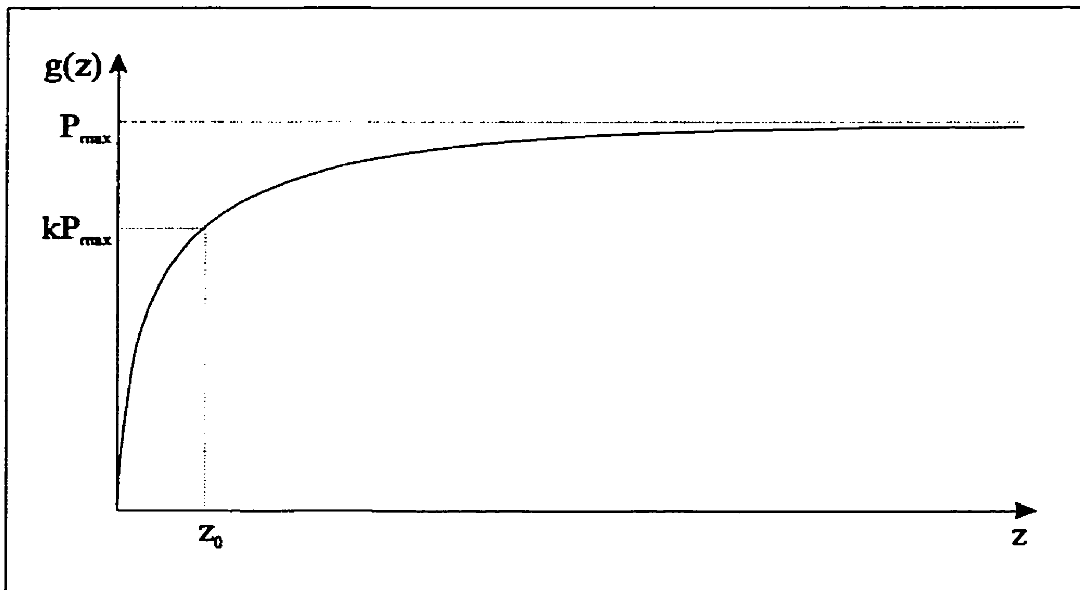


Fig. 4.1. Graph of non-linear map coverage cost function.

As can be seen from the graph of the cost function, it has the required characteristic of having a very steep slope for small map coverage but it levels off after the point z_0. The value z_0, denotes the most favourable map width.

A selection of z_0 is made by the designer based on desired compression ratios. After determining z_0, there are two more values that have to be determined by the designer, P_{max} and k. P_{max} gives the upper limit of the cost function. This is useful for ensuring that the cost function does not become so large as to increase the inaccuracy of the resultant IFS to unacceptable levels. The variable k determines how large the value of the cost function is for maps with the desired coverage. The values of P_{max} and k are determined by experiment, with the objective of optimizing the solutions found using the inverse algorithm of a given z_0.

This section introduced the map coverage cost function. The function is used to

modify the inverse algorithm for solving for parameters of the self-affine IFS model. The next section describes an application that uses this modified inverse algorithm. We will also present an application that uses the inverse algorithm for the piecewise self-affine IFS model as well.

4.4 Fractal Modelling of Linear Prediction Excitations.

In Chapter 2, linear prediction and its application in speech coding was discussed. A linear prediction based speech codec was described as having three main components; a short term predictor, a long term predictor and a module for modelling the excitation. In this section we present a new approach for modelling the excitation signal. As the main objective of the thesis is to investigate the excitation modelling, the design of both the short term and long term predictors was adopted from the GSM 06.10 full rate codec [ETSI91]. The excitation modelling presented is based on fractal interpolation. Two different interpolation techniques were investigated and implemented. The next section presents the architecture of the encoder and decoder used in both systems. Details of the two models, the self affine model and the piecewise self affine model are presented in Section 4.4.2 and 4.4.3 respectively.

4.4.1 System Architecture

The system of architecture of the fractal interpolation based encoder and decoder are presented in this section. The encoder is presented first.

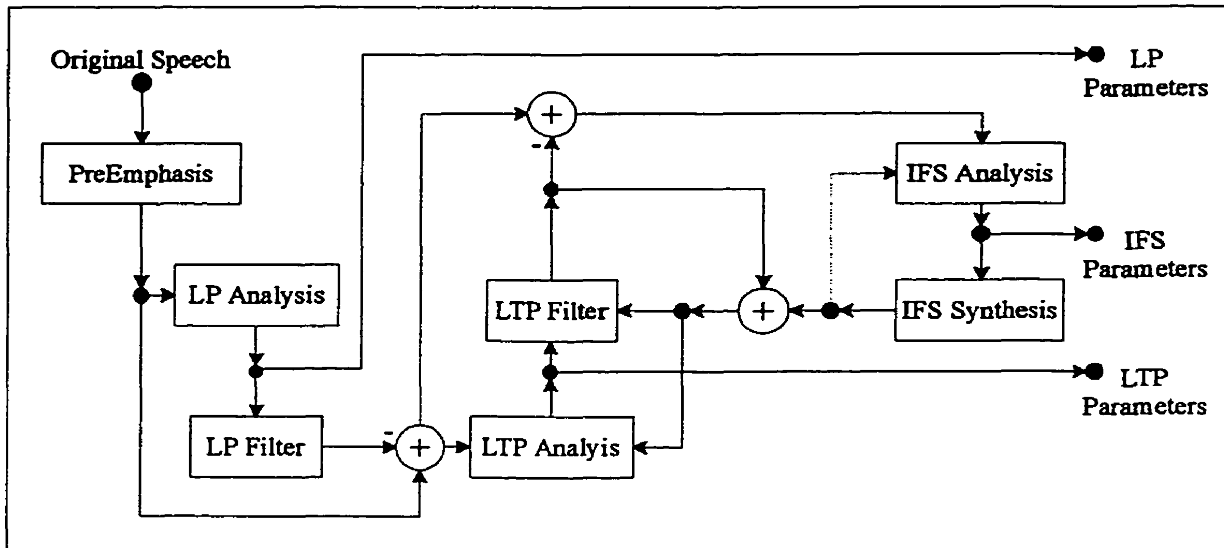


Fig. 4.2. Architecture of the encoder. The dotted arrow is needed for the piecewise self-affine model only.

The block diagram of the fractal interpolation based encoder is shown in Fig. 4.2. Both the self-affine based and the piecewise self-affine based encoders have the same basic architecture. The objective of this work is to investigate modelling of the linear prediction residuals using iterated function systems. For that reason not much effort was expended in the design of other modules of the encoder and decoder. We thus give only a brief description of these other modules.

The input speech is split into frames of 20 milliseconds each. This represents 160 samples of speech sampled at 8000 samples per second. For reasons presented in Section 2.3.5, the frame of speech is passed through a pre-emphasis filter.

After filtering, the 160 samples of speech are then analyzed using the Schur recursion algorithm to obtain the linear prediction filter coefficients. An eighth order linear prediction filter is used. As the LP coefficients are sensitive to quantization noise, they are converted into log area ratios which are less sensitive to quantization. The log area ratios (LAR) are then coded using scalar quantization.

A scalar quantizer is a many-to-one mapping of either the whole real-axis or a subset into a finite set of N real numbers [Cupe91][GeGr92]. We call the finite set Q. Each element of Q may be given one of the labels 0, 1, 2, ... (N-1). The labels are referred to as the indices of the N real numbers. The quantizer takes a real number as its input and gives the index of one of the elements of Q as output. An inverse quantizer takes an index as its input and outputs the corresponding member of Q. For example, to quantize values from 100 to 1,000 using only 10 values, the set Q may have the elements {100, 200, 300, ... 1000}. The respective indices for the elements could be {0, 1, 2, ... 9}. The quantizer, using a round to the nearest hundred operation, would thus map 230 to 1; 275 to 2 and so forth. The inverse quantizer would map 1 to 200; 2 to 300. The inverse operation is not a true inverse in that it does not return the original value that was sent into the quantizer. What it returns is a best approximation of it based on the elements of Q. The number of bits available to represent the quantized values determines the number of elements in Q. Table 4.2 gives the number of bits used to quantize each LAR.

The LP filter module in the diagram, decodes the quantized LP parameters into log area ratios. It then converts the log area ratios into prediction filter coefficients. The

impulse response of the LP filter is determined. The difference of the impulse response and the original input speech is calculated. This difference is called the LP residue.

Table 4.2. Bit allocation for the log area ratios.

Log Area Ratio Coefficient	**Number of Bits**
1	6
2	6
3	5
4	5
5	4
6	4
7	3
8	3
All coefficients	*36*

As mentioned earlier, the LP residue has most of the vocal tract information extracted from it. However, it still contains pitch information. The next stage in the encoder is thus the determination and extraction of the pitch information. This is done using a long term predictor. The pitch analysis is performed on 5 millisecond, 40 sample, windows. The LP residual is thus subdivided into four sub-frames for the analysis. The current sub-frame under analysis and the reconstructed residuals of the previous three sub-frames are used as input to the long term prediction (LTP). A first-order LTP is used, thus the equations developed in Section 2.3.4.1 are used for calculating the lag and gain for the current sub-frame. The lag is linearly scalar quantized using 7 bits and the gain is logarithmically scalar

quantized using 2 bits for each 5 ms sub-frame.

The LTP filter module does inverse quantization of the gain and lag. They are used together with the reconstructed versions of the previous three sub-frames, in order to obtain an approximation of the current sub-frame. This approximation is subtracted from original LP residuals to give the LTP residual also called the LP excitation. An iterated function system is used to model the LTP residue. The IFS parameters are then coded using scalar quantization. The attractor of the IFS is the LTP residue. More details on the IFS analysis and coding module are given in the next two sections.

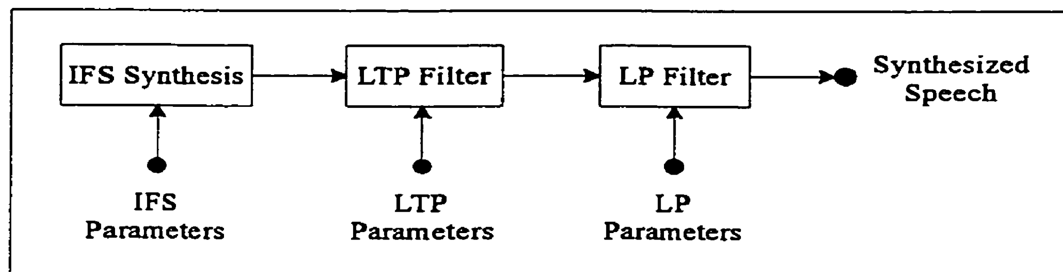


Fig. 4.3. Architecture of the decoder.

The decoder begins by reconstructing the LTP residual. This is done by first inverse quantizing the IFS parameters and then using them to synthesize the attractor of the IFS. This residual is used by the LTP filter to calculate the LP residual. The LP filter provides the final stage of the speech synthesis.

4.4.2 Self-affine Model Based Codec.

The system architecture of the self-affine model based codec was presented in the previous section. The IFS analysis and coding module takes LTP residue as input. It is

treated as time series data with 40 samples per frame and no overlap. The frame is analyzed to find a set of IFS parameters whose attractor best approximates the excitation. In experiments, both the original inverse algorithm by Mazel and Hayes and the modified algorithm presented in this chapter were used. For each map, w_i, in the IFS, the following parameters are needed: contraction factor, d_i ; interpolation point (x_i, y_i).

4.4.3 Piecewise Self-Affine Model Based Codec.

The piecewise self-affine model based codec is obtained by replacing IFS modules of figures 4.2 and 4.3 with a piecewise self-affine based IFS system. In our experiments, the inverse algorithm presented in Section 3.7.4 was used.

4.4.3.1 IFS Parameter Reduction

To represent each map w_i in a piecewise model, seven parameters are required: contraction factor d_i; interpolation point,(x_i, y_i) and address interval markers $(u`_{i,1}, v`_{i,1})$ and $(u`_{i,2}, v`_{i,2})$. In the proposed codec, only three parameters are required per map. This was achieved by making some simplifications on the model.

In the inverse algorithm used, the distance between the x-coordinates for both the interpolation points and address interval markers is made constant. These are represented by δ and ψ respectively. This means that the interpolation point for the ith map can be written

as $(i \cdot \delta, y_i)$. Since δ is fixed for the codec and the value of i is explicitly defined by the ordering of the maps, the encoder needs only save y_i for each interpolation point. This leaves six parameters.

Given that the address interval is constant, further reduction in the required parameters can be obtained. As there is a fixed number of possible address intervals, each one is given an index number; j. Then the address interval given by $[0, \psi]$ has index 0; the address interval given by $[\psi, 2\psi]$ has index 1; and so forth. We denote the address index for the ith map as j_i. This means that for each map we only need to store three parameters; d_i, y_i, and j_i. The number of parameters for each map is now three but we also need a set of y-coordinate values for the endpoint of each address interval. Thus if the size of the analysis window is N, there will be a set of $(N \div \psi + 1)$ y-coordinate values.

In the piecewise affine model based encoder, we imposed another constraint that made it unnecessary to store the y-coordinate values for each address interval. Recalling the discussion in Section 3.7.4, it was required that the address interval, ψ, be greater than the interpolation interval δ. ψ was restricted to be an integer multiple of δ. This therefore means that the endpoint of each address interval coincides with that of an interpolation point. The set of y-coordinates of address interval endpoints is thus a subset of the set of y-coordinate values of the interpolation points. We define two new variables for the map w_i.

$$\begin{aligned} s(i) &= \frac{\psi}{\delta} \cdot j_i \\ t(i) &= s(i) + \frac{\psi}{\delta} \end{aligned} \tag{4.6}$$

The address interval for the map w_i is thus given by the points in Eq. 4.7

$$(j_i\psi,\ y_{s(i)}) \qquad and \qquad ((j_i+1)\cdot\psi,\ y_{t(i)}) \tag{4.7}$$

The y in Eq. 4.7 refers to an element from the set of y-coordinates of the interpolation points. Thus by sharing these y-coordinate values between the interpolation interval and address intervals, the number of parameters required for each IFS is three per map.

4.4.3.2 IFS Parameter Determination and Quantization

The previous section showed how the parameters required to fully define each map in the iterated function system can be reduced to three: the interpolation factor, d_i; the address interval, j_i; and the y-value at the interpolation point, y_i. This completes the model for a fractal excitation based codec. To implement such a codec more issues, which have to do with the goals for developing such a codec, have to be dealt with. These issues include:

i. How large should the coverage for each map, δ, be?

ii. How large should the address interval, ψ, be?

iii. How large is each analysis frame, N, for IFS? This value determines the number of address intervals in the IFS, and therefore the number of bits required to represent the address interval index, j_i.

iv. How should the interpolation factor be quantized and how many bits should be used for the quantization?

v. How should the interpolation point, y_i, be quantized and how many bits should be used?

A systematic process of answering these questions is developed in chapter six. This process involves a number of experiments that are directed towards the development of a 6 kbps codec.

4.5 Summary of Chapter IV

This chapter started with a brief review of some of the literature on the application of fractals to speech coding. A new technique for modelling the excitation in linear prediction based codecs was developed. As the focus of the design was on developing an excitation model, the parameters of the other models in the system are the same as those of the ETSI standard; GSM 06.10. Besides allowing for focus on the excitation model, developing a system whose other parameters match those of the GSM codec also makes this codec a good reference for quality evaluation. In this chapter, we also developed a method of improving the inverse algorithm developed by Mazel and Hayes for the affine IFS model. This was done through the introduction of a cost function which can be used to influence the outcome of the inverse algorithm.

Chapter 5 describes the implementation of the codec presented in this chapter. Chapter six puts the proposed codec model to the test by developing a 6 kbps codec.

CHAPTER V

IMPLEMENTATION OF FRACTAL EXCITED CODEC

5.1 Introduction

In the previous chapter the system architectures of the fractal excited based codecs were introduced. One was based on modelling of the prediction residual using a self-affine model. The second codec was based on modelling the prediction residual using a piecewise affine fractal model. These two systems were implemented using the C programming language. This chapter gives the details of the implementation.

5.2 Test Program

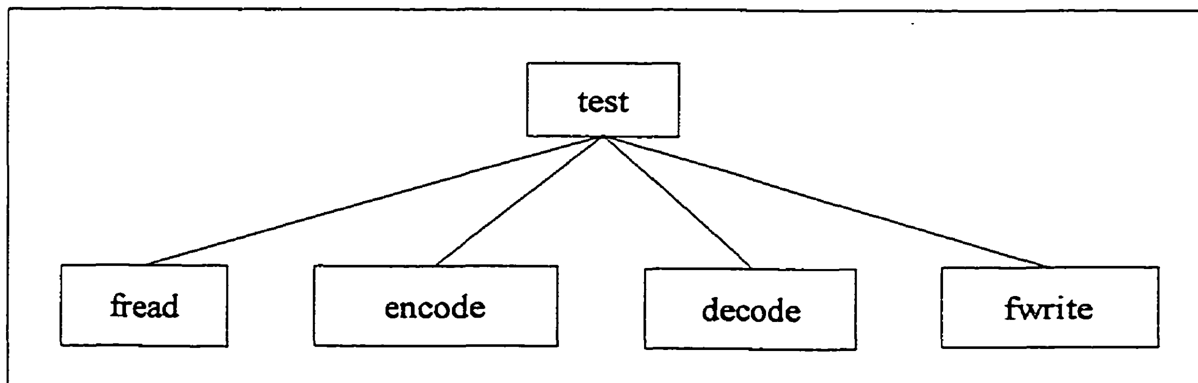


Fig. 5.1. Structure of test program used to evaluate the codec.

A test program is required to run both encoder and decoder. The purpose of the program is to take uncompressed speech from a file, compress it and then decompress it. The decompressed speech is then stored in another file. The stored file can be used to evaluate the quality of the codec's output. Figure 5.1 shows a structure chart of the test program.

The fread function reads speech in PCM format from a file and the fwrite function writes synthesized speech in PCM format to file.

5.3 The Encoder

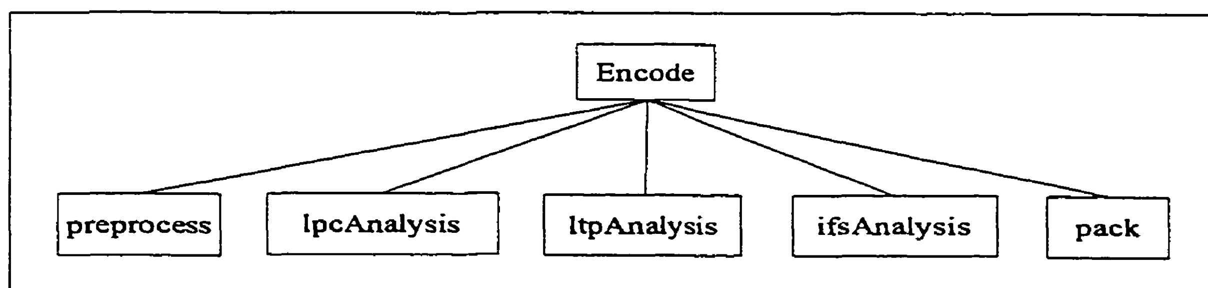


Fig. 5.2. The encode function.

Figure 5.2 shows a structure chart of the encoder. The preprocessing and LPC analysis is done on the full speech frame. The output of the LPC analysis are the log area ratios and the linear prediction residual. The log area ratios are sent to the pack routine to be put into the compressed bit-stream. The residual is sent to the LTP analysis routine which calculates the optimal autocorrelation lag and its associated gain. This routine also calculates a residual which is modelled by the IFS analysis routine. The LTP and IFS analysis routines are run on a sub-frame basis. There are four sub-frames for each 20 millisecond speech frame. The pitch lag, gain and IFS parameters for all four sub-frames are packed together with the log area ratios into one bit-stream. Before packing, each parameter is represented as a 16 bit value. The parameter will typically require only a subset of these16 bits making

the rest redundant. For example, if a parameter varies from 0 to 31, it will use only 5 of the 16 bits assigned to it. The pack routine takes only the required bits for each parameter and forms a stream of bits with no redundancies. This tightly packed stream of bits can then be sent to the decoder which unpacks them into 16 bit parameters again.

5.3.1 LPC Analysis

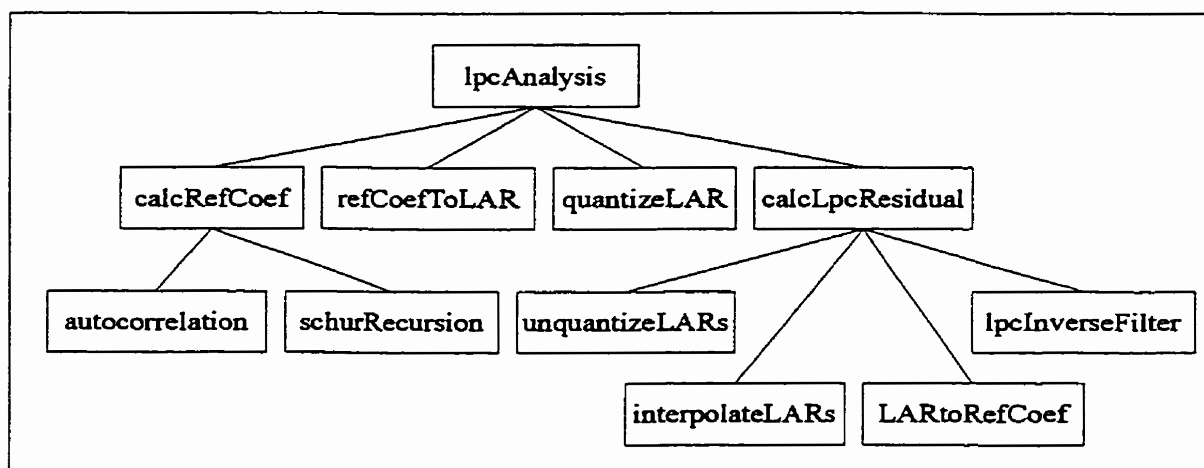


Fig. 5.3. The linear predictive coding analysis routine.

The LPC analysis routine uses the Schur recursive algorithm to calculate reflection coefficients. The reflection coefficients are then converted to log area ratio coefficients. The log area ratio coefficients are less sensitive to quantization effects than reflection coefficients [Makh75], [MaGr76], thus the former are quantized instead.

In addition to calculating the LP coefficients, the second function of the LPC analysis routine is to calculate the LP residual. This is done using the quantized log area ratios to mimic the decoder. The LP residual is used by the LTP analysis routine.

5.3.2 LTP Analysis

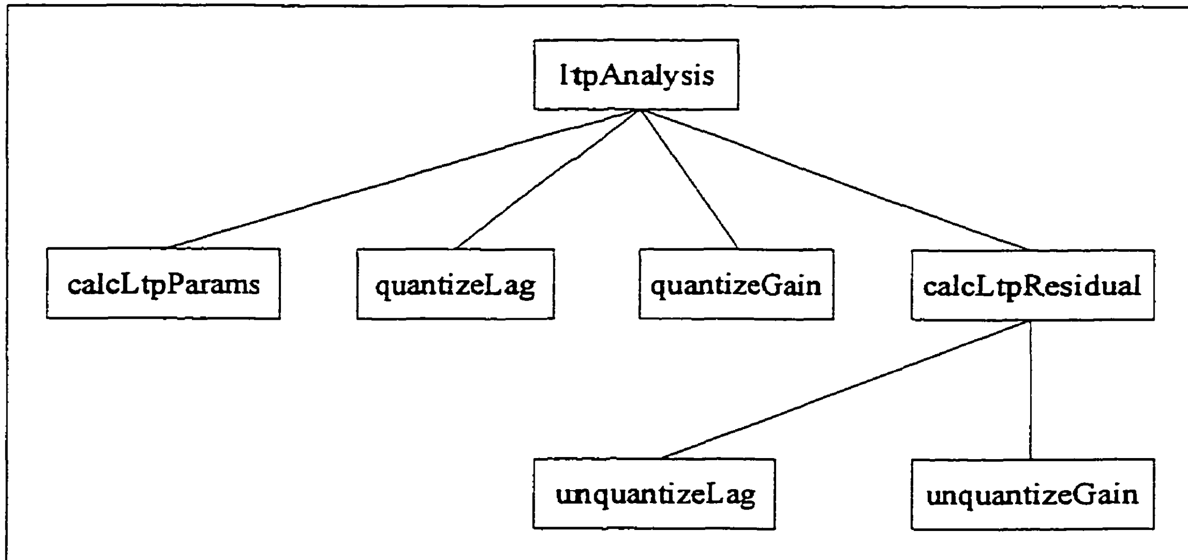


Fig. 5.4. Long term prediction analysis routine.

The LTP analysis routine takes the residual from LPC analysis and calculates a long term correlation lag and its associated gain. These values are quantized for sending to the decoder. The quantized values are also used for calculating the long term prediction residual.

5.3.3 IFS Analysis.

The IFS analysis routine implements the fractal interpolation technique used to model the LTP residue. It calculates parameters for an affine transform map for each sub-frame using the inverse algorithms presented in Figs. 3.2 and 3.1. The parameters are quantized using a linear scalar quantizer.

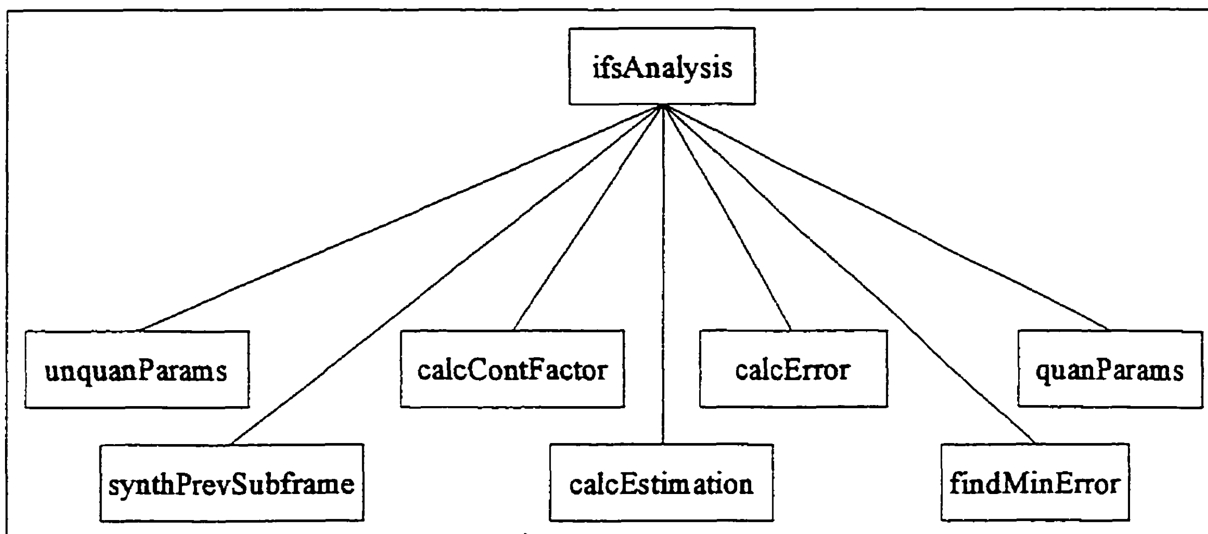


Fig. 5.5. IFS analysis and quantization routine for the piecewise self-affine model.

5.4 The Decoder

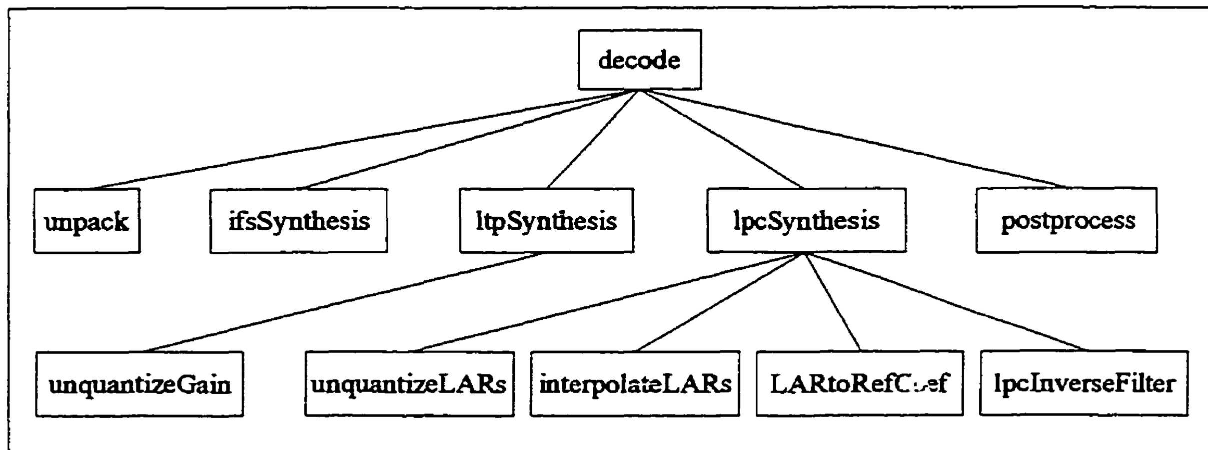


Fig. 5.6. The decoder function.

The purpose of the decoder function is to take the bit-stream from the encoder and synthesize it into speech. It unpacks the parameters from the bit-stream and then passes them to each one of the three synthesis modules. The post processing routine implements the

inverse of the pre-processing filter applied at the encoder.

5.5 Speech Compression Lab

An application, named the Speech Compression Lab, was developed for demonstrating the principles of speech compression. The application was developed for Microsoft Windows 3.1. It allows users to load speech files in Microsoft .wav file format. It displays a graphical representation of the speech waveform. The scale used to draw the waveform can be customized by the user. Once the waveform has been loaded, the user can use the `Compress` command from the `Action` menu. This will use the current codec algorithm to compress the speech and save the results in a file. The file with the compressed speech can be loaded into the Speech Compression Lab as well. As the compressed file is loaded, it is passed through the decoder. The uncompressed speech's waveform is displayed upon completion of the decoding. Different compression algorithms can be added to the Speech Compression Lab using MS Windows shared libraries; .dll files.

Figure 5.7 shows screen shots of the Speech Compression Lab. One screen shot shows the start-up screen and the other shows a speech file and its corresponding uncompressed waveform.

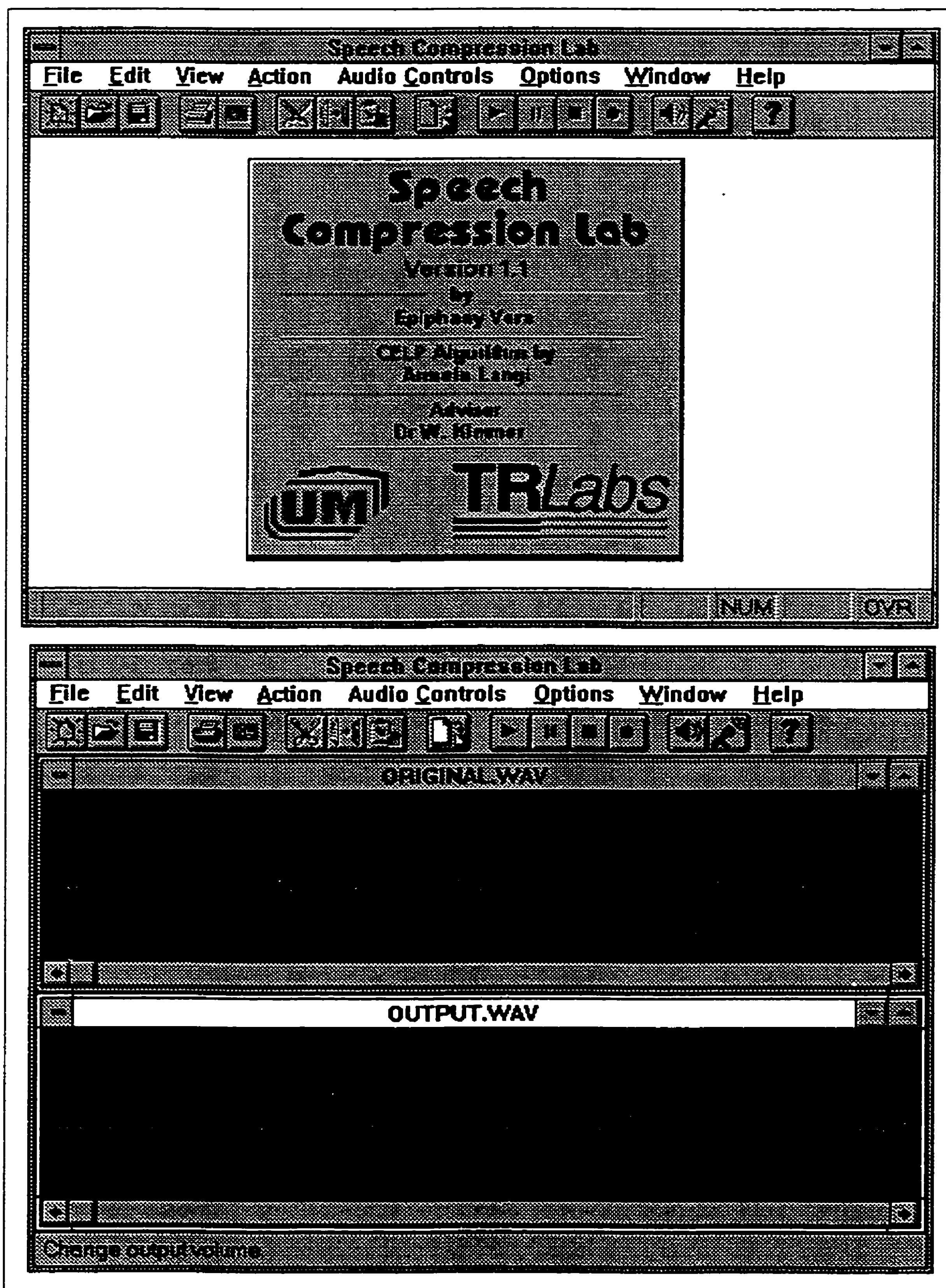


Fig. 5.7. Screen shots of the Speech Compression Lab.

5.6 Summary of Chapter V

The fractal based codec was implemented in C. This chapter presented the software architecture of the system. In addition to the encoder and decoder whose system architectures were presented in the previous chapter, a test program was introduced and described. The test program was used to prepare speech samples used to evaluate the codec and its implementation. Evaluation of the codec is presented in the next chapter. A brief description of the Speech Compression Lab software was also presented at the end of this chapter.

CHAPTER VI

SYSTEM VERIFICATION AND RESULTS

6.1 Introduction

Chapter four presented a new technique for modelling speech excitations. Chapter five describes the implementation of the system. In this chapter objective and subjective methods of assessing the quality of the synthesized speech are presented. The results of the analysis of the codecs developed in chapter four are presented.

6.2 Methods of Assessing Speech Quality

There are two main variables in assessing speech quality; intelligibility and perceived quality. Speech intelligibility refers to the content of the message. Regardless of how the speech sounds, if users can understand the message with very few errors then the speech has high intelligibility. Perceived quality is related to how natural the speech sounds to the listener. It is thus a measure of how objectionable listeners find distortions in the speech. Intelligibility tests are usually necessary only in very low bit-rate systems whereby the distortions are very severe. As the systems developed in this thesis do not fit into this category, of codecs with severe distortions, only quality measures are considered.

Speech quality assessment methods are either objective or subjective. Objective measures utilize both original and synthesized speech signals and apply some mathematically

based measure of distortion. Subjective assessment is based on the opinion of a human listener or a jury of human listeners. Subjective tests generally provide a better indicator of quality, however they tend to be time-consuming and require a lot of resources. It is also difficult to reproduce results for some subjective tests [DePH93]. Due to all these issues, subjective tests are only utilized after good results have been obtained using objective measures.

There are many different measures available for evaluating speech. The choice of measure depends on, among other things, the kind of algorithm being evaluated, the nature of distortions as well as the application of the system [IEEE69]. For the purpose of evaluating the fractal excitation based codecs, two measures were selected; the signal-to-noise ratio and the mean opinion score (MOS).

6.2.1 Signal -to-Noise Ratio

Signal-to-Noise (SNR) ratio is the most widely used measure for evaluating coding systems. It is however only appropriate for coders that seek to reproduce the original waveform. The SNR in decibels is calculated using Eq. 6.1.

$$SNR = 10 \cdot \log_{10} \cdot \frac{\sum_{n=-\infty}^{\infty} s^2(n)}{\sum_{n=-\infty}^{\infty} [s(n) - s'(n)]^2} \tag{6.1}$$

where $s(n)$ is the original signal and $s'(n)$ is the synthesized signal from the coder.

Although SNR is widely used, in some cases it is a poor estimator of speech quality [McSG78] [TrNM78] [Vora97]. This is because it is not particularly well correlated to any subjective attribute of speech quality. A slight change in the phase of a speech signal could give very low SNR whereas perceptually there might be no difference at all between the new signal and the original. The SNR however provides a good measure of how well a waveform has been reproduced. As reproducing the waveform as accurately as possible is the target of the fractal based codec, SNR is therefore a useful measure.

6.2.2 Mean Opinion Score

The mean opinion score (MOS) is the most widely used subjective quality measure for evaluation of speech coding algorithms [DePH93][Daum82] [KiHi84]. With this measure a jury of listeners rates the quality of the speech. Sentences are played and the listeners rank each one on a scale of 1 to 5. Table 6.1 describes the ranking system. A sentence is played to the listener and he/she has to rank it on a scale of 1 to 5. The scores from all the utterances from all the listeners are averaged to give the MOS score.

By using a simple scale of 1 to 5, the test allows the listeners themselves to decide what attributes to consider when assessing the quality of the speech. This makes MOS applicable to a wide range of systems. Due to this flexibility of the MOS test it is used here to evaluate the fractal excited speech coder.

Table 6.1. Five point scale used for MOS tests.

Rating	Speech Quality	Level of distortion
5	Excellent	Imperceptible
4	Good	Just perceptible but not annoying
3	Fair	Perceptible and slightly annoying
2	Poor	Annoying but not objectionable
1	Unsatisfactory	Very annoying and objectionable

The greatest advantage of the MOS test is also its weakness. The simple scale provides much versatility, however its meaning might vary significantly from listener to listener. To reduce the effect of this variation a system known as anchoring is used. In anchoring, the quality of one or two reference signals is defined and presented to each listener to establish a point of reference [IEEE69].

Other issues that affect the results of MOS test include the selection of listeners, the instructions given to the listeners and the consistency of the test condition framework. The test condition framework includes the order of presentation, the type of speech samples, presentation method, listening equipment and the listening environmental conditions. Even when MOS tests are carried out in standardized laboratories where variables are strictly controlled, the results may vary significantly depending on the jury of listeners. Goodman and Nash showed the effect of this variation by evaluating a codec in seven different countries [GoNa82]. Three of the countries are English speaking, Britain, Canada, and the United States. The other four were France, Italy, Japan and Norway. Listener opinion was

evaluated using MOS under equivalent and well controlled conditions. The results showed large variations, by at least a unit of the five point scale for most tests.

Having described the MOS test, its strength and weakness, we now describe the informal MOS test conducted to test the fractal based codec.

6.2.2.1 Informal MOS Test

The IEEE and the ITU have set up very formal requirements for MOS tests. These tests are very expensive to run. In most cases samples are sent to standardized laboratories that conduct the tests to ensure that results can be compared with other systems. In 1997 the cost of such a test at an ITU approved lab was more than US $12,000.

As the evaluation of the fractal excited codec is preliminary, an informal MOS test is used. The test was conducted with the following setup:

Playback Hardware: 486 computer running Windows 3.1, Sound Blaster 16 sound card.

Playback Software: Soundo'LE. A program that is part of the Windows™ 3.1 operating system.

Headphones: Koss TD/60

Speech Material: 10 sentences from list 1 and 10 sentences from list 2 of the Harvard sentences [IEEE69]. These were saved in Microsoft wave format.

Reference Sentence: The utterance "She left your suit in greasy wash water all year" from the TIMIT database.

Listeners: 31 undergraduate university students. 12 female and 19 male. They were all volunteers from St. John's College residence.

Procedure: Each listener was brought into the room with the computer. They were told that they were participating in a test which was part of an M.Sc. thesis. The standard introduction given to each listener was: "There are a total of 20 sentences you will listen to. You play each sentence by double-clicking its icon here on the file manager. After listening to it give a score from 1 to 5 based on this table." Table 6.1 was given to them. "Enter your score into this form." The results were collected on WordPerfect 5.2 database form. The listener was asked if he/she had any questions. After answering any questions, the reference sentence was played. The listener was informed that the reference sentence had a quality considered to be a 4 and that they should use that as their reference. Listeners were told that this was not a test of them but of the system that produced the sound and they should feel free to use their own judgement of quality. The listener was then left in a room to do the test.

6.3 System Performance Results

As mentioned in chapter four the GSM 06.10 was used as a reference codec for quality. The fractal excited codecs, for both the self-affine model, and the piecewise self-affine model, were designed to be identical to this codec with the exception of the modelling of the residual. This therefore means that a lot of the codec's parameters are determined by the GSM standard. In particular, the analysis frame size for the short term predictor is 20 milliseconds or 160 samples. Pitch prediction is done on sub-frames of 5 milliseconds. This requires that the IFS analysis be done every 5 milliseconds in order to provide the pitch predictor with the reconstructed residual that it needs for its analysis. Preset parameters for the IFS used are therefore multiples of 5 milliseconds, 40 samples, to accommodate this requirement.

The design objective of the systems developed was to obtain speech reproduction quality comparable to the GSM 06.10 with comparable or better bit-rate. GSM has a SNR of about 12 dB. Its subjective quality was also measured using the informal MOS test. A MOS score of 3.91 was obtained. In designing the fractal based codecs, the bits required to represent the excitation were reduced until the SNR was comparable to that of GSM. Sections 6.3.1 and 6.3.2 give the results.

6.3.1 Performance of Self-Affine Model

In this system, the model parameters were varied so as to obtain different values for the average map coverage. The quality of the speech was then evaluated for each value.

Table 6.2 shows a summary of the results. The first column gives the average map coverage. This value is used to calculate the number of maps that are required to represent each sub-frame.

Table 6.2. Speech quality as a function of map coverage in the self-affine based speech codec.

Average map coverage	Maps per sub-frame	SNR (dB)
2.7	14.8	8.6
5.6	7.1	3.9
10.8	3.7	1.3

Three parameters are required to describe each map. These are the contradiction factor, the x-value and y-value of the interpolation points. Thus for a system with an average of 14.8 maps, 44.4 parameters would be required for 40 samples in the sub-frame. This value is too high for any significant compression to be achieved. The second and third entries in the table require significantly fewer parameters but the speech quality is too low compared to GSM 06.10. Experiments were also done with larger frame sizes. The results were not much better than those presented in table 6.2.

6.3.2 Performance of Piecewise Self-Affine Model

In order to make the inverse problem tractable, the map coverage was fixed by the designer unlike in the self-affine model. The map coverage size was set to be a multiple of the codec's sub-frame size. This was done to align the IFS parameters with those of the LP

and LTP analysis modules. The first experiment varied the map coverage to investigate its effect on quality. The results are shown in Table 6.3.

Table 6.3. Speech quality as a function of map coverage in the piecewise self-affine based codec.

Map Coverage	Maps per Codec Frame	SNR (dB)
160	1	6.4
120	1.3	11.1
80	2	13
40	4	13.9
20	8	14.4
10	16	15.3

The results show that map coverage of 80 samples or below, would provide quality comparable to GSM. Since a large map coverage is desirable, 80 would be a good choice, however 40 was selected to allow for degradation of quality from quantization.

Having selected a map coverage size, the next issue was the size of the analysis frame. The larger the analysis frame, the more address intervals in the IFS. The number of address intervals determines the number of bits required to code the address interval. As mentioned in Chapter 4, the size of the address intervals is chosen to be twice that of the map coverage. The map coverage was selected to be 40, therefore the address interval has to be 80. The number of address intervals in an analysis frame is calculated by dividing the size of the analysis frame by 80, the address interval size. Table 6.4 shows the trade-off between bits used to code the address interval and SNR. A significant drop in SNR is observed when

the analysis frame size drops from 640 samples to 320. An analysis frame size of 640 was therefore chosen. Since there are eight possible address intervals in an analysis frame of 640 samples, only three bits are required to code the address interval.

Table 6.4. Speech quality as a function of analysis frame size for a map coverage of 40 samples.

Analysis Frame Size	**Number of address intervals**	**Address Interval Bits**	**SNR (dB)**
160	2	1	5.7
320	4	2	10.2
640	8	3	13.5
1280	16	4	13.8
2560	32	5	13.9
2120	64	6	13.8

Two more parameters that have to be retained and quantized for each IFS map are the contraction factor and the y-value at the interpolation point. The next experiment studied the effect of linear quantization on the contraction factor. Table 6.5 shows the trade-off between the number of bits assigned to coding of the contraction factor and SNR. Four bits were chosen for quantizing the contraction factor. Table 6.1 shows the results of linear scalar quantization of the interpolation values.

In order to obtain a 6 kbps system, quantization bits for the interpolation values was set to five. Table 6.6 gives a summary of the bit allocation for the 6 kbps system.

Table 6.5. Speech quality as a function of bits used to quantize the contraction factor.

Contraction Factor Bits	SNR (dB)
2	5.1
3	8.9
4	13.1
5	13.3
6	13.4
7	13.3
8	13.5
9	13.5

Table 6.6. Speech quality as a function of bits used to quantize the interpolation values.

Interpolation value bits	SNR (dB)
2	3.4
3	7.8
4	9.5
5	10.9
6	11.6
7	11.8
8	12
9	12.3
10	12.5
11	12.8
12	13.1

Table 6.7. Bit-allocation of 6 kbps piecewise self-affine based codec.

Parameter		Bits per Frame	Bit Rate
Log area ratios		36 bits/160 sample frame	1,800 bps
Long term prediction		9 bits/40 sample sub-frame	1,800 bps
Fractal:	Address interval	3 bits/40 sample sub-frame	600 bps
	Contraction factor	4 bits/40 sample sub-frame	800 bps
	Interpolation value	5 bits/40 sample sub-frame	1,000 bps
Total		120 bits/160 sample frame	6,000 bps

This codec configuration was evaluated using the informal MOS test described in this chapter. The MOS score was 3.84.

6.4 Discussion of Results

Among other things, the experiments in the previous section demonstrated the ease with which the parameters of the fractal interpolation system can be modified to control bit-rate and codec quality. A 6 kbps codec was developed. The SNR of the codec was measured to be 10.9 dB. This seems to imply that the quality is much lower than that of GSM 06.10 which was measured to be 12dB. The MOS results indicated that the quality was much closer with the 6 kbps codec scoring 3.84 against GSM's 3.91. Informal comparison of the two during development showed them to be indistinguishable. The lower SNR might have been due to the weakness of SNR discussed earlier.

6.5 Summary of Chapter VI

This chapter described two methods for evaluating speech codecs; MOS test and signal-to-noise ratio (SNR). SNR was used to tune the parameters of the fractal based codec developed in chapter four. The result of the parameter tuning was a 6 kbps codec with an SNR of 10.9 dB and an informal MOS score of 3.84. The process of tuning the parameters demonstrated the flexibility of the fractal modelling of excitation. The system can be easily changed to different bit-rates.

CHAPTER VII

CONCLUSIONS AND RECOMMENDATIONS

The aim of this thesis was to find an alternative approach for modelling the excitation in linear prediction based codecs. The main requirement for the alternative approach was having lower bit-rate while maintaining speech quality. A method of modelling the speech excitations using fractal interpolation was developed. The fractal interpolation was performed using two different techniques; the self-affine fractal model and the piecewise self-affine model. To provide a reference point, the architecture of the codec was based on the ETSI codec standard GSM 06.10. The only difference between the fractal codecs and the GSM codec is in modelling of the excitation.

The self-affine fractal model was found not to provide a good model for excitations. When applied to the excitation, the model requires a very large number of parameters. This results in very high bit-rates making it unsuitable for speech compression. Besides having high bit-rates, the quality of reconstructed speech is also very poor. From these observations, we conclude that excitation is neither self-affine nor statistically self-affine.

The piecewise self-affine model was found to provide a good alternative for modelling speech excitations. It is very flexible and can be easily tuned to provide coders at different bit-rates and quality levels. The 6 kbps coder implemented based on this model had an SNR of 10.9dB which is comparable to the 12dB SNR for 13kbps GSM coder. It also scored an informal MOS score of 3.84 which is comparable to the 3.91 MOS score for the GSM. This 6 kbps implementation demonstrated that fractal interpolation is a viable alternative for developing high quality linear prediction based codecs.

There are three main contributions made in this thesis. The first is the addition of a new tool, fractal interpolation, for use in modelling excitations. The second contribution is an improved algorithm for solving the inverse problem for the self-affine fractal model. The improved algorithm gives the system designer a method of influencing the solution of the

inverse algorithm. The thesis has also contributed implementations of fractal interpolation algorithms; the self-affine inverse algorithm and the piecewise self-affine algorithm.

The architecture of the codec developed here is based on the ETSI GSM codec standard. This dictated many of the design parameters including; input and output speech frame size, the order for the short term linear prediction filter, the long term prediction order and the quantization tables for the codec. Due to these restrictions the proposed system setup might be sub-optimal. Development of a codec which does not use these restrictions might yield much better results.

An issue that was not addressed in this thesis is that of computational complexity. Both the IFS analysis and synthesis algorithms used for the fractal excited based codec have very high computational complexity. Before IFS based speech codecs gain wide usage, the complexity of IFS algorithms have to be significantly reduced. Some of the work being done by Gregory Wornell at MIT and the fractal research group at INRIA in France linking IFS with wavelets might pave the way to finding fast IFS algorithms.

REFERENCES

[Bake90] R. J. Baken, "Irregularity of vocal period and amplitude: A first approach to the fractal analysis of voice," *Journal of Voice*, Vol. 4, No. 3, pp. 185-197. 1990.

[Barn88] M. Barnsley, *Fractals everywhere.* Cambridge, MA: Academic Press, 1993.

[Cupe91] V. Cuperman, "Speech Coding," in *Advances in Electronics and Electron Physics*, Vol. 82, pp. 97 - 196. 1991.

[DaLM95] K. Daoudi, J. L. Véhel, and Y. Meyer, "Construction of continuous functions with prescribed local regularity," preprint to appear in Journal of Constructive Approximation

[Daum82] W. R. Daumer, "Subjective comparison of several efficient speech coders," *IEEE Trans. on Communications*, Vol. 30, pp. 655-662, April 1982.

[DaVé95] K. Daoudi and J. L. Véhel, "Speech signal modelling based on local regularity analysis," *IASTED/IEEE International Conference on Signal and Image Processing,* November 1995.

[DePH87] J. Deller Jr, J. G. Proakis, and J. H. L. Hansen, *Discrete-Time Processing of Speech Signals.* Upper Saddle River, NJ: Prentice Hall, 1987.

[ETSI91] *European Digital Cellular Telecommunications System (phase 1); Full-rate Speech Transcoding (GSM 06.10)*, Valbonne Cedex, France: European Telecommunications Standards Institute (ETSI), 1991.

[FrDu91] G. C. Freeland and T. S. Durrani, "On the use of general iterated function systems in signal modelling," *Proc. of IEEE ICASSP*, vol. 4. pp. 2345 - 2348. 1990.

[GeGr92] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression.* Boston, MA: Kluwer Academic Publishers, 1992.

[GoNa82] D. Goodman and R. D. Nash, "Subjective quality of the same speech transmission conditions in seven different countries," *IEEE Trans. on Acoustics, Speech and Signal Processing*, Vol. 30, pp. 642-654, April 1982.

[IEEE69] "IEEE standard publication No. 297: IEEE recommended practice for speech quality measurements," *IEEE Trans. Audio and Electro-acoustics*, pp. 225-246, September 1969.

[KiHI84] N. Kitawaki, M. Honda, and K. Itoh, "Speech quality assessment methods for speech coding systems," *IEEE Communications Magazine*, Vol. 22, pp. 26-33, October 1984.

[Kins94] W. Kinsner, "Noise, power laws, and the spectral fractal dimension," *Technical Report*, DEL94-1; University of Manitoba; 1994. pp. 38.

[Krey78] E. Kreyszig, *Introduction to Functional Analysis with Applications*. New York, NY: John Wiley and Sons. 1978.

[Kubi97] G. Kubin, "Poincaré section techniques for speech," *IEEE Workshop on Speech Coding for Telecommunications Proceedings*, 0-7803-4073-6/97, pp. 7-8. 1997.

[LaKi95] A. Langi and W. Kinsner, "Consonant characterization using correlation fractal dimension for speech recognition," Proc. IEEE WESCANEX, 95CH3581-6/0-7803-2741-1/95, pp. 208-213. 1995.

[Lang92] A. Langi, *Code-Excited Linear Predictive Coding for High-quality and Low Bit-rate Speech*. M.Sc. Thesis, University of Manitoba, 1992.

[LuCu92] P. Lupini and V. Cuperman, "Excitation modelling based on speech residual information," *Proc. of IEEE ICASSP*, 0-7803-0532-9/92, pp. I-333 - I-336. 1992.

[MaHa92] D.S. Mazel and M.H. Hayes, "Using iterated function systems to model discrete sequences," *IEEE Trans on Signal Processing*, vol. 40, No. 7. pp. 1724 - 1734, July 1992.

[MaPo97] P. Maragos and A. Potamianos, "On using fractal features of speech sounds in automatic speech recognition," *Proc. of Eurospeech*. 1997

[Mara91] P. Maragos, "Fractal aspects of speech signals: Dimension and interpolation," *Proc. of IEEE ICASSP*, CH2977-7/91/0000-0417, pp. 417-420. 1991.

[MaYo90] P. Maragos and K.L Young, "Fractal excitation signals for CELP speech coders," *Proc. of IEEE ICASSP*, CH2847-2/90, pp. 669-672. 1990.

[MaZh92] S. Mallat and S. Zhong, "Characterization of signals from multiscale edges," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 14. No. 7. pp. 710-732. July 1992.

[McSG78] B. J. McDermott, C. Scagliola, and D. J. Goodman, "Perceptual and objective

evaluation of speech processed by adaptive differential PCM," *Proc. of IEEE ICASSP*, pp. 581-585, April 1978.

[Pars86] T.W. Parsons, *Voice and Speech Processing*. New York, NY: McGraw-Hill. 1986.

[RaSc78] L. R. Rabiner and R. W. Schafer, *Digital Processing of Speech Signals*, Upper Saddle River, NJ:Prentice Hall. 1978.

[Stry88] L. Stryer, *Biochemistry*. New York, NY: W.H. Freeman and Company, 1988, pp. 187-191.

[Swan87] C. Swanson, *A Study and Implementation of Real-Time Linear Predictive Coding of Speech*. M.Sc. Thesis, University of Manitoba, 1987.

[TrNM78] J. M. Tribolet, P. Noll, and B. J. McDermott, "A study of complexity and quality of speech waveform coders," in *Proc. Of IEEE ICASSP*, pp. 586-590, April 1978.

[VéDL94] J. L. Véhel, K. Daoudi, and E. Lutton, "Fractal modelling of speech signals," *Fractals*. Vol. 2. No. 3. pp. 379-382, June 1994.

[ViHa93] G. Vines and M. H. Hayes, "Nonlinear address maps in a one-dimensional fractal model," *IEEE Transactions on Signal Processing*, Vol. 41, No. 4. pp. 1721 - 1724. April 1993.

[Vine93] G. Vines, *Signal modelling with iterated function systems*. PhD Thesis, Georgia Institute of Technology; 1993.

[Vora97] S. Voran, "Estimation of perceived speech quality using measuring normalizing blocks," *Proc. of IEEE Workshop on Speech Coding for Telecommunications*, pp. 83-84, September 1997.

[Worn96] G. W. Wornell, *Signal Processing with Fractals: A Wavelet Based Approach*. Upper Saddle River, NJ: Prentice Hall, pp. 30-57, 1996.

[ZhCT94] X. Zhu, B. Cheng, and D. M. Titterington, "Fractal model of one dimensional discrete signal and its implementation," *IEE Proc. Image and Signal Processing*, Vol. 141, No. 5, pp. 318-324, October 1994.

APPENDIX A: SOURCE CODE FOR PIECEWISE SELF-AFFINE FRACTAL EXCITED LINEAR PREDICTIVE CODEC

```
/* ******************************************************************* */
/*                                                                     */
/* AUTHOR: Epiphany Vera                                               */
/*         The University of Manitoba and                              */
/*         TRLabs (Telecommunications Research Labs)                   */
/*         Winnipeg, MB                                                */
/*         Canada.                                                     */
/*                                                                     */
/*         Copyright (c) 1996                                          */
/*                                                                     */
/* ******************************************************************* */


/* ******************************************************************* */
/*                                                                     */
/* FILENAME:            testRIFS.c                                     */
/*                                                                     */
/* DESCRIPTION:         Program for testing the fractal excited codec. */
/*                                                                     */
/*                                                                     */
/* PUBLIC FUNCTIONS:   main(..)                                        */
/*                                                                     */
/* PRIVATE FUNCTIONS:  none                                            */
/*                                                                     */
/* ******************************************************************* */


#include <stdio.h>
#include <conio.h>

#include "encDecR.h"


int main(void)
{
   unsigned char bitStream[FELP_BITSTREAM];
   short PCMStream[FELP_FRAME_SIZE];
   char ch, quit;
   unsigned long Frame;
   FILE *InFile, *OutFile;

   quit = 0;
   do
   {
      printf
      ("Would you like to Analyze, Synthesize, Both, Quit? (A/S/B/Q): ");
      ch = getche();
      printf ("\n");
      if (ch=='A'||ch=='a')
      {
         InFile = fopen("..\\data\\voiced.raw","rb");
         OutFile = fopen("test.bit","wb");
```

```
    if((InFile != 0)&&(OutFile != 0))
    {
       Frame=0;
       while(fread(PCMStream,sizeof(short),FELP_FRAME_SIZE,InFile)
             == FELP_FRAME_SIZE)
       {
          printf("Frame: %lu\n",Frame);
          encodeRIFS(PCMStream, bitStream);
          fwrite(bitStream,sizeof(unsigned char),
                 FELP_BITSTREAM,OutFile);
          Frame=Frame+1;
       }
       fclose(InFile);
       fclose(OutFile);
    }
    else
    {
       printf("An error occured while opening files or in memory \
              allocation.\n");
       fclose(InFile);
       fclose(OutFile);
       return -1;
    }
 }
 else if (ch=='S'||ch=='s')
 {
    OutFile = fopen("test_syn.raw","wb");
    InFile = fopen("test.bit","rb");
    if((OutFile != 0)&&(InFile != 0))
    {
       Frame=0;
       do
       {
          fread(bitStream,sizeof(unsigned char),
                FELP_BITSTREAM,InFile);
          printf("Frame: %lu\n",Frame);
          decodeRIFS(bitStream, PCMStream);
          fwrite(PCMStream,sizeof(short),FELP_FRAME_SIZE,OutFile);
          Frame=Frame+1;
       }while (!feof(InFile));
    }
    fclose(OutFile);
    fclose(InFile);
 }
 else if (ch=='B'||ch=='b')
 {
    InFile = fopen("..\\data\\voiced.raw","rb");
    OutFile = fopen("test_syn.raw","wb");
    if((InFile != 0)&&(OutFile != 0))
    {
       Frame=0;
       while(fread(PCMStream,sizeof(short),FELP_FRAME_SIZE,InFile)
          == FELP_FRAME_SIZE)
       {
          printf("Frame: %lu\n",Frame);
```

```
                encodeRIFS(PCMStream, bitStream);
                decodeRIFS(bitStream, PCMStream);
                fwrite(PCMStream,sizeof(short),FELP_FRAME_SIZE,OutFile);
                Frame=Frame+1;
             }
          }
          fclose(InFile);
          fclose(OutFile);
       }
       else if (ch=='Q'||ch=='q')
       {
          quit = 1;
       }
   } while (!quit);
   printf ("\n\nOperation Complete, program terminating.\n");
   return 1;
}
```

```
/* ******************************************************************** */
/*                                                                      */
/* AUTHOR: Epiphany Vera                                                */
/*         The University of Manitoba and                               */
/*         TRLabs (Telecommunications Research Labs)                    */
/*         Winnipeg, MB                                                 */
/*         Canada.                                                      */
/*                                                                      */
/*         Copyright (c) 1996                                           */
/*                                                                      */
/* ******************************************************************** */

/* ******************************************************************** */
/*                                                                      */
/* FILENAME: encDecR.h                                                  */
/*                                                                      */
/* ******************************************************************** */

#define     FELP_FRAME_SIZE 160
#define  FELP_BITSTREAM 16

void encodeRIFS(short *source, unsigned char *c);
int  decodeRIFS(unsigned char *c, short *output);
```

```
/* *************************************************************** */
/*                                                                 */
/* AUTHOR: Epiphany Vera                                           */
/*         The University of Manitoba and                          */
/*         TRLabs (Telecommunications Research Labs)               */
/*         Winnipeg, MB                                            */
/*         Canada.                                                 */
/*                                                                 */
/*         Copyright (c) 1996                                      */
/*                                                                 */
/* *************************************************************** */


/* *************************************************************** */
/*                                                                 */
/* FILENAME:     encDecR.c   Recurrent IFS encoder and decoder     */
/*                                                                 */
/* DESCRIPTION: Has the encoder and decoder functions.             */
/*                                                                 */
/*                                                                 */
/* PUBLIC FUNCTIONS:  encode(..)                                   */
/*                    decode(..)                                   */
/*                                                                 */
/* PRIVATE FUNCTIONS: floatToShortBuff(..)                         */
/*                                                                 */
/* *************************************************************** */


#include <stdio.h>
#include <limits.h>
#include <string.h>

#include "defs.h"
#include "RIFS.h"
#include "LTP.h"
#include "pre_post.h"
#include "packRIFS.h"
#include "STP.h"
#include "encDecR.h"


static void floatToShortBuff(float *floatBuff, int length,
                             short *shortBuff);

static float encBuffer1[LOOK_BACK_SAMPLES+FRAME_SAMPLES];
static float encBuffer2[IFS_ANALYSIS_SIZE];

void encodeRIFS(short *inSpeech, unsigned char *bitStream)
{

   short qInterpPoint[SUBFRAMES], addrInt[SUBFRAMES],
        qContFactor[SUBFRAMES];
                         /* Quantized Log Area Ratios.              */
   short    qLAR[LPC_ORDER];
                         /* Quantized pitch lag.                    */
```

```
    short qLag[SUBFRAMES];
                            /* Quantized long term prediction gain.      */
    short qGain[SUBFRAMES];
                            /* LPC residual synthesized by LTP and IFS.  */
    float *synthLpcResidue;
                            /* LTP estimate of LPC residual.             */
    float *ltpEstLpcResidual;
                            /* Input speech after preprocessing.         */
    float filteredSpeech[FRAME_SAMPLES];
                            /* LPC residual.                             */
    float *lpcResidual;
                            /* LTP residual.                             */
    float ltpResidual[SUBFRAME_SAMPLES];
                            /* LTP residual synthesized by IFS module.   */
    float *synthLtpResidue;
    int       k,i;

    preprocess(inSpeech, FRAME_SAMPLES, filteredSpeech);
    lpcResidual = filteredSpeech;
    lpcAnalysis(lpcResidual, qLAR);

    /* The two buffers, synthLpcResidue and ltpEstLpcResidue together   */
    /* form the buffer used for long term prediction analysis.          */
                                 /* Indexed from -LOOK_BACK_SAMPLES to 0 */
    synthLpcResidue= encBuffer1+LOOK_BACK_SAMPLES;
                                 /* Indexed from 0 to SUBFRAME_SAMPLES-1 */
    ltpEstLpcResidual = synthLpcResidue;
    synthLtpResidue = encBuffer1;

    for (k=0; k<SUBFRAMES; k++)
    {
       ltpAnalysis(lpcResidual, synthLpcResidue, ltpResidual,
                ltpEstLpcResidual, &qLag[k], &qGain[k]);
       memcpy(synthLtpResidue+IFS_ANALYSIS_SIZE-SUBFRAME_SAMPLES,
            ltpResidual, SUBFRAME_SAMPLES*sizeof(float));
       ifsAnalysisPWise(synthLtpResidue, &addrInt[k], &qInterpPoint[k],
                   &qContFactor[k]) ;

       for (i = 0; i < SUBFRAME_SAMPLES; i++)
       synthLpcResidue[i] = synthLtpResidue[IFS_ANALYSIS_SIZE-
                            SUBFRAME_SAMPLES+i] + ltpEstLpcResidual[i];
       synthLpcResidue    += SUBFRAME_SAMPLES;
       ltpEstLpcResidual  += SUBFRAME_SAMPLES;
       lpcResidual        += SUBFRAME_SAMPLES;
       memcpy(synthLtpResidue, synthLtpResidue+SUBFRAME_SAMPLES,
           (IFS_ANALYSIS_SIZE-SUBFRAME_SAMPLES)*sizeof(float));
    }
    memcpy(encBuffer1, encBuffer1+FRAME_SAMPLES,
          LOOK_BACK_SAMPLES*sizeof(float));

    packRIFS(qLAR, qLag, qGain, qInterpPoint, addrInt, qContFactor,
            bitStream);
}
```

```
static float encBuffer1[LOOK_BACK_SAMPLES+FRAME_SAMPLES];
static float encBuffer2[IFS_ANALYSIS_SIZE];

int decodeRIFS(unsigned char *bitStream, short *output)
{
   static float buffer[LOOK_BACK_SAMPLES+SUBFRAME_SAMPLES];
                              /* Quantized Log Area Ratios.                */
   short    qLAR[LPC_ORDER];
                              /* Quantized pitch lag.                      */
   short qLag[SUBFRAMES];
                              /* Quantized long term prediction gain.      */
   short qGain[SUBFRAMES];
                              /* LPC residual synthesized by LTP and IFS. */
   float *synthLpcResidue;
                              /* LTP residual synthesized by IFS module.  */
   float *synthLtpResidue;
   short qInterpPoint[SUBFRAMES], addrInt[SUBFRAMES],
         qContFactor[SUBFRAMES];
   float speech[FRAME_SAMPLES];
   int   j;


   unPackRIFS(bitStream, qLAR, qLag, qGain, qInterpPoint, qContFactor,
              addrInt);

   /* synthLpcResidue is indexed from -LOOK_BACK_SAMPLES to          */
   /* SUBFRAME_SAMPLES                                                */
   synthLpcResidue  = encBuffer1 + LOOK_BACK_SAMPLES;
   synthLtpResidue  = encBuffer2;

   for (j=0; j < SUBFRAMES; j++)
   {
      ifsSynthesisPWise(qInterpPoint[j], addrInt[j], qContFactor[j],
                        synthLtpResidue);
      ltpSynthesis(qLag[j], qGain[j], synthLtpResidue+(IFS_ANALYSIS_SIZE
                  -SUBFRAME_SAMPLES), synthLpcResidue );
      memcpy(synthLtpResidue, synthLtpResidue+SUBFRAME_SAMPLES,
             (IFS_ANALYSIS_SIZE-SUBFRAME_SAMPLES)*sizeof(float));
   }

   /* synthLpcResidue is now all samples from the present frame.     */
   synthLpcResidue  = buffer;

   lpcSynthesis(qLAR, synthLpcResidue, speech);
   Postprocess(speech, FRAME_SAMPLES, speech);

   floatToShortBuff(speech, FRAME_SAMPLES, output);

   return 0;
}


static void floatToShortBuff(float *floatBuff, int length,
                             short *shortBuff)
```

```
{
   int   i;
   float temp;

   for(i=0; i<length; i++)
   {
      temp = floatBuff[i];
      if(temp>SHRT_MAX)
         shortBuff[i] = SHRT_MAX;
      else if(temp<SHRT_MIN)
         shortBuff[i] = SHRT_MIN;
      else
         shortBuff[i] = (short)temp;
   }
}
```

```
/* ******************************************************************* */
/*                                                                     */
/* AUTHOR: Epiphany Vera                                               */
/*         The University of Manitoba and                              */
/*         TRLabs (Telecommunications Research Labs)                   */
/*         Winnipeg, MB                                                */
/*         Canada.                                                     */
/*                                                                     */
/*         Copyright (c) 1996                                          */
/*                                                                     */
/* ******************************************************************* */

/* ******************************************************************* */
/*                                                                     */
/* FILENAME: pre_post.h                                                */
/*                                                                     */
/* ******************************************************************* */

void preprocess(short *sIn, short length, float *sOut);
void Postprocess(float *sIn, short length, float *sOut);
```

```
/* ******************************************************************* */
/*                                                                     */
/* AUTHOR: Epiphany Vera                                               */
/*         The University of Manitoba and                              */
/*         TRLabs (Telecommunications Research Labs)                   */
/*         Winnipeg, MB                                                */
/*         Canada.                                                     */
/*                                                                     */
/*         Copyright (c) 1996                                          */
/*                                                                     */
/* ******************************************************************* */


/* ******************************************************************* */
/*                                                                     */
/* FILENAME:     pre_post.c                                            */
/*                                                                     */
/* DESCRIPTION:  Preprocessing and postprocessing routines.            */
/*                                                                     */
/*                                                                     */
/* PUBLIC FUNCTIONS:  preprocess(..)                                   */
/*                    postprocess(..)                                  */
/*                                                                     */
/* PRIVATE FUNCTIONS:                                                  */
/*                                                                     */
/* ******************************************************************* */


#include    <stdio.h>

#include "pre_post.h"

#define ALPHA 0.9989929199219F /* Offset compensation filter constant */
#define BETA  0.8599853515625F /* Preemphasis filter constant.        */

static float sInPrev   = 0.0F;
static float sCompPrev = 0.0F;

void preprocess(short *sIn, short length, float *sOut)
{
   int k;
   float input, sComp, sPre;

   for(k=0; k<length; k++)
   {
      /* Downscale input signal. */
      input = (float)(((*sIn++)>>3)<<2);

      /* Offset Compensation */
      sComp = input - sInPrev + ALPHA*sCompPrev;

      /* Preemphasis */
      sPre = sComp - BETA*sCompPrev;

      sInPrev   = input;
```

```
      sCompPrev = sComp;
      *sOut++ = sPre;
   }
}


static float postProcessMem = 0.0F;

void Postprocess(float *sIn, short length, float *sOut)
{
   int   k;
   float prevSample;

   prevSample = postProcessMem;
   for (k=length; k!=0; k--)
   {
      *sOut++ = ((*sIn)*2) + 0.85999F*prevSample;
      prevSample = *sIn++;
   }
   postProcessMem = prevSample;
}
```

```
/* ******************************************************************* */
/*                                                                     */
/* AUTHOR: Epiphany Vera                                               */
/*         The University of Manitoba and                              */
/*         TRLabs (Telecommunications Research Labs)                   */
/*         Winnipeg, MB                                                */
/*         Canada.                                                     */
/*                                                                     */
/*         Copyright (c) 1996                                          */
/*                                                                     */
/* ******************************************************************* */

/* ******************************************************************* */
/*                                                                     */
/* FILENAME: STP.h                                                     */
/*                                                                     */
/* ******************************************************************* */

void lpcSynthesis(short *LARcr, float    *excitation, float     *s);
void lpcAnalysis(float *lpcResidual, short *qLAR);
```

```
/* ******************************************************************* */
/*                                                                     */
/* AUTHOR: Epiphany Vera                                               */
/*         The University of Manitoba and                              */
/*         TRLabs (Telecommunications Research Labs)                   */
/*         Winnipeg, MB                                                */
/*         Canada.                                                     */
/*                                                                     */
/*         Copyright (c) 1996                                          */
/*                                                                     */
/* ******************************************************************* */


/* ******************************************************************* */
/*                                                                     */
/* FILENAME:    STP.c                                                  */
/*                                                                     */
/* DESCRIPTION: Short term prediction analysis and synthesis routines */
/*              Also contains routines for conversion of reflection   */
/*              coefficients to log area ratios and quantization of   */
/*              log area ratios.                                       */
/*                                                                     */
/*                                                                     */
/* PUBLIC FUNCTIONS:  lpcAnalysis(..)                                  */
/*                    lpcSynthesis(..)                                 */
/*                                                                     */
/* PRIVATE FUNCTIONS: quantizeLARs(..)                                 */
/*                    unquantizeLARs(..)                               */
/*                    interpolateLARs(..)                              */
/*                                                                     */
/* ******************************************************************* */


#include <stdio.h>
#include <assert.h>
#include <math.h>
#include <string.h>

#include "schur.h"
#include "STP.h"


static const float A[] = {20.0F, 20.0F, 20.0F, 20.0F, 13.637F, 15.0F,
8.334F, 8.824F};
static const float B[] = {0.0F, 0.0F, 4.0F, -5.0F, 0.184F, -3.5F,
-0.666F, -2.235F};
static const short minLAR[] = {-32, -32, -16, -16, -8, -8, -4, -4};
static const short maxLAR[] = {31, 31, 15, 15, 7, 7, 3, 3};

static void QuantizeLARs(float *LAR, short *LARc)
{
   float temp;
   int   i;

   for (i=0; i<8; i++)
```

```
    {
       temp = LAR[i]*A[i]+B[i];
       temp = (float)floor(temp+0.5);

       if(temp>maxLAR[i])
          LARc[i] = maxLAR[i] - minLAR[i];
       else if(temp<minLAR[i])
          LARc[i] = 0;
       else
          LARc[i] = (short)(temp) - minLAR[i];
    }
}


static void unquantizeLARs(short *LARc, float *LAR)
{
   int i;

   for(i=0; i<8; i++)
      LAR[i] = (LARc[i]+minLAR[i]-B[i])/A[i];
}


static void interpolateLARs(float *prevLAR, float *LAR, float factorA,
                            float factorB, float *interpLAR)
{
   int i;

   for(i=0; i<8; i++)
      interpLAR[i] = factorA*prevLAR[i] + factorB*LAR[i];
}


static void larToReflectionCoef(float *LAR, float *R)
{
   int i;
   float temp, sign;

   for(i=0; i<8; i++)
   {
      sign = 1.0F;
      if(LAR[i]<0)
         sign = -1.0F;
      temp = (float)fabs(LAR[i]);

      assert(temp<=1.625F);
      if(temp>=1.225)
         R[i] = sign*(0.125F*temp + 0.796875F);
      else if(temp>=0.675F)
         R[i] = sign*(0.5F*temp+0.3375F);
      else
         R[i] = LAR[i];
   }
```

```
}


static void reflectionCoefToLAR(float      *r)
{
   float sign, temp;
   int   i;

   /* Computation of the LAR[0..7] from the r[0..7] */
   for (i=0; i<8; i++)
   {
      sign = 1.0F;
      if(*r<0)
         sign = -1.0F;
      temp = (float)fabs(*r);
      if((temp>=0.675)&&(temp<0.95F))
         *r = sign*(2*temp-0.625F);
      else if(temp>=0.95F)
         *r = sign*(8*temp-6.375F);
      *r++;
   }
}


static void lpcInverseFilter(float *R, int start, int end, float *s)
{
   static float U[8] = {0.0F};
   int k,i;
   float d, prevD, prevU, temp;

   for(k=start; k<=end; k++)
   {
      prevD = s[k];
      temp = s[k];
      for (i = 0; i < 8; i++)
      {
         prevU = U[i];
         U[i]  = temp;
         d     = prevD + R[i]*prevU;
         temp  = prevU + R[i]*prevD;
         prevD = d;
      }
      s[k] = d;
   }
}


static void lpcFilter(float *R, int start, int end, float *excitation,
                      float *speech)
{
   static float V[8] = {0.0F};
```

```
   int            i, k;
   float currentSample;

   for(k=start; k<=end; k++)
   {
      currentSample = excitation[k] - (R[7]*V[7]);
      for (i=6; i>=0; i--)
      {
         currentSample = currentSample - (R[i]*V[i]);
         V[i+1] = V[i] + (R[i]*currentSample);
      }
      V[0] = currentSample;

      speech[k] = currentSample;
   }
}


void calcLpcResidual(short *LARc, float *s)
{
   static float prevLAR[8]={0.0F};
   float LAR[8], subFrameLAR[8];

   unquantizeLARs(LARc, LAR);

   interpolateLARs(prevLAR, LAR, 0.75F, 0.25F, subFrameLAR);
   larToReflectionCoef(subFrameLAR, subFrameLAR);
   lpcInverseFilter(subFrameLAR, 0, 13, s);

   interpolateLARs(prevLAR, LAR, 0.5F, 0.5F, subFrameLAR);
   larToReflectionCoef(subFrameLAR, subFrameLAR);
   lpcInverseFilter(subFrameLAR, 14, 26, s);

   interpolateLARs(prevLAR, LAR, 0.25F, 0.75F, subFrameLAR);
   larToReflectionCoef(subFrameLAR, subFrameLAR);
   lpcInverseFilter(subFrameLAR, 27, 39, s);

   interpolateLARs(prevLAR, LAR, 0.0F, 1.0F, subFrameLAR);
   larToReflectionCoef(subFrameLAR, subFrameLAR);
   lpcInverseFilter(subFrameLAR, 40, 159, s);

   memcpy(prevLAR, LAR, 8*sizeof(float));
}


void lpcSynthesis(short *LARcr, float      *excitation, float      *s)
{
   static float prevLAR[8]={0.0F};
   float LAR[8], subFrameLAR[8];

   unquantizeLARs(LARcr, LAR);

   interpolateLARs(prevLAR, LAR, 0.75F, 0.25F, subFrameLAR);
   larToReflectionCoef(subFrameLAR, subFrameLAR);
   lpcFilter(subFrameLAR, 0, 13, excitation, s);
```

```
   interpolateLARs(prevLAR, LAR, 0.5F, 0.5F, subFrameLAR);
   larToReflectionCoef(subFrameLAR, subFrameLAR);
   lpcFilter(subFrameLAR, 14, 26, excitation, s);

   interpolateLARs(prevLAR, LAR, 0.25F, 0.75F, subFrameLAR);
   larToReflectionCoef(subFrameLAR, subFrameLAR);
   lpcFilter(subFrameLAR, 27, 39, excitation, s);

   interpolateLARs(prevLAR, LAR, 0.0F, 1.0F, subFrameLAR);
   larToReflectionCoef(subFrameLAR, subFrameLAR);
   lpcFilter(subFrameLAR, 40, 159, excitation, s);

   memcpy(prevLAR, LAR, 8*sizeof(float));
}


void lpcAnalysis(float *lpcResidual, short *qLAR)
{
   float *LAR, refCoef[8];

   calcRefCoef(lpcResidual, refCoef);
   LAR = refCoef;
   reflectionCoefToLAR(LAR);
   QuantizeLARs(LAR, qLAR);
   calcLpcResidual(qLAR, lpcResidual);
}
```

```
/* ****************************************************************** */
/*                                                                    */
/* AUTHOR: Epiphany Vera                                              */
/*         The University of Manitoba and                             */
/*         TRLabs (Telecommunications Research Labs)                  */
/*         Winnipeg, MB                                               */
/*         Canada.                                                    */
/*                                                                    */
/*         Copyright (c) 1996                                         */
/*                                                                    */
/* ****************************************************************** */

/* ****************************************************************** */
/*                                                                    */
/* FILENAME: schur.h                                                  */
/*                                                                    */
/* ****************************************************************** */

void calcRefCoef(float *s, float *fLAR);
```

```
/* ****************************************************************** */
/*                                                                    */
/* AUTHOR: Epiphany Vera                                              */
/*         The University of Manitoba and                             */
/*         TRLabs (Telecommunications Research Labs)                  */
/*         Winnipeg, MB                                               */
/*         Canada.                                                    */
/*                                                                    */
/*         Copyright (c) 1996                                         */
/*                                                                    */
/* ****************************************************************** */


/* ****************************************************************** */
/*                                                                    */
/* FILENAME:    schur.c                                               */
/*                                                                    */
/* DESCRIPTION: Schur recursion for calculation of reflection         */
/*              coeffecients.                                         */
/*                                                                    */
/*                                                                    */
/* PUBLIC FUNCTIONS:  calcRefCoef(..)                                 */
/*                                                                    */
/* PRIVATE FUNCTIONS: autocorrelation(..)                             */
/*                    calcRefCoef(..)                                 */
/*                                                                    */
/* ****************************************************************** */


#include <stdio.h>
#include <assert.h>
#include <math.h>

#include "schur.h"


static void autocorrelation (float *s, float *autocor)
{
   int   k, i;
   float *delay;
   float temp;

   for (k=0; k<=8; k++)
   {
      delay = s-k;
      temp = 0.0F;
      for (i = k; i < 160; ++i)
         temp += s[i]*delay[i];
      autocor[k] = temp;
   }
}
```

```
static void schurRecursion(float *autocor, float *refCoef)
{
   int      i, m, n;
   float fTemp;
   float    fP[9];      /* 0..8 */
   float    fK[9];  /* 2..8 */

   if (autocor[0] < 0.0000001)
   {
      for (i = 8; i--; *refCoef++ = 0.0F) ;
      return;
   }

   /*   Initialize array P[..] and K[..] for the recursion. */
   for (i = 1; i <= 7; i++)
      fK[i] = autocor[i];
   for (i = 0; i <= 8; i++)
      fP[i] = autocor[i];

   /*   Compute reflection coefficients  */
   for(n=1; n<=8; n++, refCoef++)
   {
      fTemp = (float)fabs(fP[1]);
      if (fP[0] < fTemp)
      {
         for (i = n; i <= 8; i++)
            *refCoef++ = 0.0F;
         return;
      }

      *refCoef = fTemp/fP[0];
      if (fP[1] > 0)
         *refCoef = -(*refCoef);

      if(n==8)
         return;

      /*  Schur recursion  */
      fP[0] = fP[0] + fP[1]*(*refCoef);
      for (m=1; m<=(8-n); m++)
      {
         fP[m] = fP[m+1] + fK[m]*(*refCoef);
         fK[m] = fK[m] + fP[m+1]*(*refCoef);
      }
   }
}



void calcRefCoef(float *s, float *R)
{
   float autoCorrCoef[9];

   autocorrelation(s, autoCorrCoef);
```

```
   schurRecursion(autoCorrCoef, R);
}
```

```
/* ****************************************************************** */
/*                                                                    */
/* AUTHOR: Epiphany Vera                                              */
/*         The University of Manitoba and                             */
/*         TRLabs (Telecommunications Research Labs)                  */
/*         Winnipeg, MB                                               */
/*         Canada.                                                    */
/*                                                                    */
/*         Copyright (c) 1996                                         */
/*                                                                    */
/* ****************************************************************** */

/* ****************************************************************** */
/*                                                                    */
/* FILENAME: LTP.h                                                    */
/*                                                                    */
/* ****************************************************************** */

void ltpAnalysis(float *subFrame, float *prevSubFrames, float *residual,
                    float    *subFrameEst,short   *qLag,short   *qGain);
void ltpSynthesis(short lag, short qGain, float *erp, float *speech);
```

```
/* ****************************************************************** */
/*                                                                    */
/* AUTHOR: Epiphany Vera                                              */
/*         The University of Manitoba and                             */
/*         TRLabs (Telecommunications Research Labs)                  */
/*         Winnipeg, MB                                               */
/*         Canada.                                                    */
/*                                                                    */
/*         Copyright (c) 1996                                         */
/*                                                                    */
/* ****************************************************************** */

/* ****************************************************************** */
/*                                                                    */
/* FILENAME:     LTP.c                                                */
/*                                                                    */
/* DESCRIPTION:  Long term prediction functions.                      */
/*                                                                    */
/*                                                                    */
```

```
/* PUBLIC FUNCTIONS:  ltpAnalysis(..)                                    */
/*                    ltpSynthesis(..)                                   */
/*                                                                       */
/* PRIVATE FUNCTIONS: calcLtpParams(..)                                  */
/*                    calcLtpResidual(..)                                */
/*                    quantizeGain(..)                                   */
/*                    unquantizeGain(..)                                 */
/*                    quantizeLag(..)                                    */
/*                    unquantizeLag(..)                                  */
/*                                                                       */
/* ******************************************************************* */


#include <stdio.h>
#include <assert.h>

#include "LTP.h"


static void calcLtpParams(float *prevSubFrames, float *subFrame,
                          int *lag, float *gain)
{
  int i,candidate, maxCorrIndex;
  float maxCorr, power, temp;

  /* Calculate cross correlation function and find max correlation */
  maxCorr = 0.0F;
  maxCorrIndex = 40;
  for(candidate=40; candidate<=120; candidate++)
  {
    temp = 0.0F;
    for(i=0; i<40; i++)
    {
      temp += subFrame[i]*prevSubFrames[i-candidate];
    }
    if(temp>maxCorr)
    {
      maxCorr = temp;
      maxCorrIndex = candidate;
    }
  }
  *lag = maxCorrIndex;

  /* Calculate gain of reconstructed residual. */
  power = 0.0000001F; /* Make non-zero to avoid divide by 0 error. */
  for(i=0; i<40; i++)
  {
    temp  = prevSubFrames[i-maxCorrIndex];
    power += temp*temp;
  }

  *gain = maxCorr/power;
}
```

```
static short quantizeGain(float gain)
{
  short qGain;

  if(gain<=0.2)
    qGain = 0;
  else if(gain<=0.5)
    qGain = 1;
  else if(gain<=0.8)
    qGain = 2;
  else
    qGain = 3;

  return qGain;
}



static float unquantizeGain(int qGain)
{
  float gain;

  assert(qGain>=0);
  assert(qGain<=3);

  switch(qGain)
  {
  case 0:
    gain = 0.10F;
    break;
  case 1:
    gain = 0.35F;
    break;
  case 2:
    gain = 0.65F;
    break;
  case 3:
    gain = 1.00F;
    break;
  }
  return gain;
}


static short quantizeLag(int lag)
{
  short qLag;

  qLag = (short)(lag-40);

  return qLag;
}
```

```
static int unquantizeLag(short qLag)
{
  int lag;

  lag = (int)(qLag+40);

  return lag;
}


static void calcLtpResidual(short qLag, short qGain,
                            float *prevSubFrames, float *subFrame,
                            float *subFrameEst, float *residual)
{
  float gain;
  int   lag, k;

  lag  = unquantizeLag(qLag);
  gain = unquantizeGain(qGain);
  for(k=0; k<40; k++)
  {
    subFrameEst[k] = gain*prevSubFrames[k-lag];
    residual[k] = subFrame[k] - subFrameEst[k];
  }
}

void ltpAnalysis(
                      float *subFrame,          /* [0..39]
Input    */
                      float *prevSubFrames,  /* [-120..-1]        Input
*/
                      float *residual,          /* [0..39]
Output      */
                      float *subFrameEst,    /* [0..39]
Output   */
                      short *qLag,           /* correlation lag   Output
*/
                      short *qGain           /* gain factor       Output
*/
                )
{
  float gain;
  int lag;

  calcLtpParams(prevSubFrames, subFrame, &lag, &gain);
  *qGain = quantizeGain(gain);
  *qLag = quantizeLag(lag);
  calcLtpResidual(*qLag, *qGain, prevSubFrames, subFrame, subFrameEst,
                  residual);
}


void ltpSynthesis(
   short qLag, short qGain,
   register float *erp,       /* [0..39]IN           */
```

```
   register float *speech     /* [-120..-1]IN, [0..40]OUT */
)
{
  int            i, lag;
  float gain;


   lag = unquantizeLag(qLag);
   assert((lag>=40)&&(lag<=120));
   gain = unquantizeGain(qGain);

   /* Update previous subframes */
   for (i = 0; i <= 119; i++)
      speech[-120+i] = speech[-80+i];

   for(i=0; i<40; i++)
      speech[i] = erp[i] + gain*(speech[i-lag]);
}
```

```
/* ******************************************************************* */
/*                                                                     */
/* AUTHOR: Epiphany Vera                                               */
/*         The University of Manitoba and                              */
/*         TRLabs (Telecommunications Research Labs)                   */
/*         Winnipeg, MB                                                */
/*         Canada.                                                     */
/*                                                                     */
/*         Copyright (c) 1996                                          */
/*                                                                     */
/* ******************************************************************* */

/* ******************************************************************* */
/*                                                                     */
/* FILENAME: Rifs.h                                                    */
/*                                                                     */
/* ******************************************************************* */

void ifsSynthesisPWise(short qY, short addrInt, short qD,
                       float *residual);
void ifsAnalysisPWise(float *residual, short *addrInt, short *qY,
                      short *qD);
```

```
/* ***************************************************************** */
/*                                                                   */
/* AUTHOR: Epiphany Vera                                             */
/*         The University of Manitoba and                            */
/*         TRLabs (Telecommunications Research Labs)                 */
/*         Winnipeg, MB                                              */
/*         Canada.                                                   */
/*                                                                   */
/*         Copyright (c) 1996                                        */
/*                                                                   */
/* ***************************************************************** */


/* ***************************************************************** */
/*                                                                   */
/* FILENAME:     Rifs.c                                              */
/*                                                                   */
/* DESCRIPTION:  Piecewise self-affine fractal modelling functions.  */
/*                                                                   */
/*                                                                   */
/* PUBLIC FUNCTIONS:  ifsSynthesisPWise(..)                          */
/*                    ifsAnalysisPWise(..)                           */
/*                                                                   */
/* PRIVATE FUNCTIONS: round(..)                                      */
/*                    calcMaps(..)                                   */
/*                    synthAttractor(..)                             */
/*                    calcContFactor(..)                             */
/*                    calcError(..)                                  */
/*                    findMinError(..)                               */
/*                    calcEstimation(..)                             */
/*                    calcParam(..)                                  */
/*                    quanParams(..)                                 */
/*                    unquanParams(..)                               */
/*                                                                   */
/* ***************************************************************** */



/***** HEADER FILES *****/
#include <conio.h>
#include <stdio.h>
#include <limits.h>
#include <float.h>
#include <math.h>
#include <stdlib.h>

#include "defs.h"


/***** GLOBAL CONSTANTS *****/
#define frameSize IFS_ANALYSIS_SIZE
#define maxMaps 20    //Maximum number of maps to be used +1
#define minContFactor 1E-12
#define DCBias INT_MAX
```

```
#define maxRepeats 15


/*************************************************************************/
/*                          SHARED ROUTINES                              */
/*************************************************************************/


int round(float x)
{
   if ((0.5-(ceil(x)-x)>0))
      return (int)ceil(x);
   else
      return (int)floor(x);
}

/*************************************************************************/
/*                       END OF SHARED ROUTINES                          */
/*************************************************************************/


/*************************************************************************/
/*                         SYNTHESIS ROUTINES                            */
/*************************************************************************/

void calcMaps(float *residual, float y, short addrInt, float a, float e,
              float c,float d, float f,int delta, int psi)
{
   int p,q,I;
   float Yp,Yq,YI,YF;
   float F;

                              // Using endpoint constraints, parameters of
the affine
                              // maps are calculated below.
   I=addrInt*psi;
   YI=residual[(int)(floor(I/delta))];
   F=(float)(I+psi);
   YF=residual[(int)(ceil(F/delta))];
   p=IFS_ANALYSIS_SIZE-1-delta;
   //           p=p-(floor(p/(float)psi))*psi; //This is different from
the
      //[MaHa92] description, here the first interpolation point
      //in an address interval is set to an x value of zero
   q=p+delta;
   Yp=residual[p];
   Yq=y;

   a = (q-p)/(F-I);
   e = (float)((long)F*p-(long)I*q)/(F-I);
   c = ((long)Yq-Yp)/(F-I)-d*((long)YF-YI)/(F-I);
   f = (float)(((double)F*Yp-(long)I*Yq)/(F-I)-d*((double)F*YI-
      (long)I*YF)/(F-I));
}
```

```
void synthAttractor(short addrInt,float a,float e,float c,float d,
                    float f, int delta,int psi,float*G)
{
   const int maxSqaureError=0; //Maximum error in attractor calculation
   float  A1[frameSize+1], A2[frameSize+1];
   int    A1x[frameSize+1], A2x[frameSize+1];
   int numOfMaps, k, n, repeats=0;
   int F, I;
   float error, lastError;

   error=(float)INT_MAX;
   numOfMaps = frameSize/delta;
   for (n=IFS_ANALYSIS_SIZE-1-delta; n<IFS_ANALYSIS_SIZE; n++)
   {//Initialise data array
      A1[n]=10.0F;
      A1x[n]=n;
      G[n]=0.0F;
   }
   while((error>maxSqaureError)&&(repeats<=maxRepeats))
   {
      lastError=error;
      error = 0.0F;

      I=addrInt*psi;
      F=I+psi;
      for (k=I; k<=F; k++)
      {
         A2x[k]=round(a*A1x[k] + e);
         if (A2x[k]>frameSize) //Required because rounding might give
         {                     //result greater than array bound
               A2x[k]=frameSize;
         }
         A2[k] = c*A1x[k] + d*A1[k] + f;
      }
      for (k=I; k<=F; k++)
      {
         G[A2x[k]]=A2[k];
      }

      for (k=IFS_ANALYSIS_SIZE-1-delta; k<=IFS_ANALYSIS_SIZE; k++)
         error += (float)(fabs((double)A1[k]-(double)G[k]));
         //Get new A for more iterations
      for (k=IFS_ANALYSIS_SIZE-1-delta; k<=IFS_ANALYSIS_SIZE; k++)
         A1[k]=G[k];
      if (error==lastError)
         repeats++;
      else
         repeats = 0;
   }
}

/***************************************************************************/
/*                       END OF SYNTHESIS ROUTINES                         */
/***************************************************************************/
```

```
/****************************************************************************/
/*                          ANALYSIS ROUTINES                               */
/****************************************************************************/


float calcContFactor(float *PCMData,int p,float Yp,int q,float Yq,
                     float I, float F)
{
   double num, den; //numerator & denominator of d
   double An, Bn;
   float Xi, a, e;
   int n, m;

   num = 0.0;
   den = 0.0;
   a=(q-p)/(F-I);
   e = (float)((double)(F*p)-(double)(I*q))/(F-I);
   for (n=(int)I; n<=F; n++)
   {
      m=round(a*n+e);
      Xi=(F-n)/(F-I);
      An = PCMData[n]-((double)(Xi*PCMData[(int)I])+
           (double)((1-Xi)*PCMData[(int)F]));
      Bn=PCMData[m]-(long)((long)(Xi*Yp)+(long)((1-Xi)*Yq));
      num += Bn*An;
      den += An*An;
   }
   if ((den*den)<0.00000001)
      return FLT_MAX;
   else
      return (float)(num/den);
}

double calcError(float *PCMData, float *Hi, int delta, int p, int q)
{
   int k;
   float testError, newError;

   testError = 0.0F;
   for (k=p; k<=q; k++)
   {
      newError = (((long)Hi[k]-PCMData[k])*((long)Hi[k]-
                 PCMData[k]));
      testError=testError+newError;
   }
   testError /= delta;
   return testError;
}

float findMinError(double *error, int numOfTryMaps, int *best)
{
   float minError, errorCheck;
```

```
    int k;

    if (numOfTryMaps>=1)
    {
       minError=FLT_MAX;
       for (k=1; k<=numOfTryMaps; k++)
       {
          errorCheck=(float)(minError-error[k]);
          if (errorCheck>0)
          {
             *best=k;
             minError=(float)error[k];
          }
       }
    }
    return minError;
}

void calcEstimation(float *PCMData, float testD, float I, float F, int p,
                     int q, float *Hi)
{
    int Hx[frameSize+1];
    float H[frameSize+1];
    float testA, testE, testC, testF;
    float YI, YF;
    float Yp, Yq;
    int k;

    YI=(float)PCMData[(int)I];
    YF=(float)PCMData[(int)F];
    Yp=PCMData[p];
    Yq=PCMData[q];

    testA = (q-p)/(F-I);
    testE = (float)(((double)(F*p)-(double)(I*q))/(F-I));
    testC = ((long)Yq-Yp)/(F-I)-testD*((long)YF-YI)/(F-I);
    testF = (float)(((double)(F*Yp)-(double)(I*Yq))/(F-I)-testD*
            ((double)(F*YI)-(double)(I*YF))/(F-I));
    for (k=(int)I; k<=(int)F; k++)    //Map every point in the range I to F
    {
       Hx[k] = round(testA*k + testE);
       H[k]  = testC*k + testD*PCMData[k] + testF;
    }

    for (k=(int)I; k<=(int)F; k++)  //Create the function Hi (or H hat)
    {
       Hi[Hx[k]]=H[k];    //This function is a map of the original into the
    }                     //the interpolation interval of the current ma~
}

void calcParam(float *PCMData, float *y, short *addrInt,float *d,
                int delta, int psi)
{
    int   p, q;
    float Yp, Yq;
```

```
    int j, numOfTryMaps, best;
    float testD, minError;
    int tempAddr[frameSize-1];
    float tempD[frameSize-1];
    double error[100];
    float Hi[frameSize+1];
    float I,F;


    q  = (NUM_MAPS-1)*delta;
    p  = q-delta;
    Yp = PCMData[p];
    Yq = PCMData[q];
    j  = -1;
    numOfTryMaps=0;
    while (j<(NUM_ADDRESS_INT-1))
    {
       testD=2.0F;
       while ((((testD*testD)>=1)||((testD*testD)<minContFactor))
              &&(j<(NUM_ADDRESS_INT-1)))
       {
          j++;
          I=(float)(j*psi);
          F=I+psi;
          testD=calcContFactor(PCMData,p,Yp,q,Yq,I,F);
       }

       if ((j<(NUM_ADDRESS_INT+1))&&(((testD*testD)<=1)&&((testD*testD)
          >minContFactor)))
       {            //Increase num of trial maps coz one was just found
          numOfTryMaps++;
          tempD[numOfTryMaps] = testD;
          tempAddr[numOfTryMaps]=j;

          calcEstimation(PCMData, testD, I, F, p, q, Hi);

          error[numOfTryMaps] = calcError(PCMData, Hi, delta, p, q);
       }          /*End if*/
    }             /*Next j*/

    minError = findMinError(error, numOfTryMaps, &best);
    *d        = tempD[best];
    *addrInt  = tempAddr[best];
    *y        = Yq;
}           /*Function calcParam*/

/**************************************************************************/
/*                         END OF ANALYSIS ROUTINES                       */
/**************************************************************************/
```

```
void quanParams(float y, float d, short *qY, short *qD)
{
   int tempInt;
   float tempFloat;

   tempInt = (int)(y + (1<<((16-INTERP_VALUE_BITS)-1)));
   *qY = (short)(tempInt>>(16-INTERP_VALUE_BITS));

   tempFloat = d*((1<<INTERP_FACTOR_BITS)-1);
   *qD = round(tempFloat);
}


void unquanParams(short qY, short qD, float *y, float *d)
{

   *y = (short)((qY)<<(16-INTERP_VALUE_BITS));

   *d = qD/(float)((1<<INTERP_FACTOR_BITS)-1);
}


void ifsSynthesisPWise(short qY, short addrInt, short qD,
                       float *residual)
{
   float y;
   float a,e,c,d,f;

   unquanParams(qY, qD, &y, &d);
   calcMaps(residual,y,addrInt,a,e,c,d,f,INTERP_INTERVAL,
            ADDRESS_INTERVAL);
   synthAttractor(addrInt,a,e,c,d,f,INTERP_INTERVAL,ADDRESS_INTERVAL,
                  residual);
}


void ifsAnalysisPWise(float *residual, short *addrInt, short *qY,
                      short *qD)
{
   float y, d;

   calcParam(residual, &y, addrInt, &d, INTERP_INTERVAL,
             ADDRESS_INTERVAL);
   quanParams(y, d, qY, qD);
   ifsSynthesisPWise(*qY, *addrInt, *qD, residual);
}
```

APPENDIX B:
SOURCE CODE FOR THE GSM 06.10 CODEC

```
/* ****************************************************************** */
/*                                                                    */
/* AUTHOR: Epiphany Vera                                              */
/*         The University of Manitoba and                            */
/*         TRLabs (Telecommunications Research Labs)                  */
/*         Winnipeg, MB                                               */
/*         Canada.                                                    */
/*                                                                    */
/*         Copyright (c) 1996                                         */
/*                                                                    */
/* ****************************************************************** */


/* ****************************************************************** */
/*                                                                    */
/* FILENAME:             testGSM.c                                    */
/*                                                                    */
/* DESCRIPTION:          Program for testing the GSM codec.           */
/*                                                                    */
/*                                                                    */
/* PUBLIC FUNCTIONS:     main(..)                                     */
/*                                                                    */
/* PRIVATE FUNCTIONS:    none                                         */
/*                                                                    */
/* ****************************************************************** */


#include <stdio.h>
#include <conio.h>

#include "encDecG.h"


int main(void)
{
   unsigned char bitStream[GSM_BITSTREAM];
   short PCMStream[GSM_FRAME_SIZE];
   char ch, quit;
   unsigned long Frame;
   FILE *InFile, *OutFile;

   quit = 0;
   do
   {
      printf
      ("Would you like to Analyze, Synthesize, Both, Quit? (A/S/B/Q): ");
      ch = getche();
      printf ("\n");
      if (ch=='A'||ch=='a')
      {
         InFile = fopen("..\\data\\voiced.raw","rb");
         OutFile = fopen("test.bit","wb");
```

```
    if((InFile != 0)&&(OutFile != 0))
    {
       Frame=0;
       while(fread(PCMStream,sizeof(short),
             GSM_FRAME_SIZE,InFile) == GSM_FRAME_SIZE)
       {
          printf("Frame: %lu\n",Frame);
          encodeGSM(PCMStream, bitStream);
          fwrite(bitStream,sizeof(unsigned char),
                 GSM_BITSTREAM,OutFile);
          Frame=Frame+1;
       }
       fclose(InFile);
       fclose(OutFile);
    }
    else
    {
       printf("An error occured while opening files or in memory \
              allocation.\n");
       fclose(InFile);
       fclose(OutFile);
       return -1;
    }
 }
 else if (ch=='S'||ch=='s')
 {
    OutFile = fopen("test_syn.raw","wb");
    InFile = fopen("test.bit","rb");
    if((OutFile != 0)&&(InFile != 0))
    {
       Frame=0;
       do
       {
          fread(bitStream,sizeof(unsigned char),
                GSM_BITSTREAM,InFile);
          printf("Frame: %lu\n",Frame);
          decodeGSM(bitStream, PCMStream);
          fwrite(PCMStream,sizeof(short),GSM_FRAME_SIZE,OutFile);
          Frame=Frame+1;
       }while (!feof(InFile));
    }
    fclose(OutFile);
    fclose(InFile);
 }
 else if (ch=='B'||ch=='b')
 {
    InFile = fopen("..\\data\\voiced.raw","rb");
    OutFile = fopen("test_syn.raw","wb");
    if((InFile != 0)&&(OutFile != 0))
    {
       Frame=0;
       while(fread(PCMStream,sizeof(short),GSM_FRAME_SIZE,InFile)
          == GSM_FRAME_SIZE)
       {
          printf("Frame: %lu\n",Frame);
```

```
                encodeGSM(PCMStream, bitStream);
                decodeGSM(bitStream, PCMStream);
                fwrite(PCMStream,sizeof(short),GSM_FRAME_SIZE,OutFile);
                Frame=Frame+1;
            }
        }
        fclose(InFile);
        fclose(OutFile);
    }
    else if (ch=='Q'||ch=='q')
    {
        quit = 1;
    }
  }while (!quit);
  printf ("\n\nOperation Complete, program terminating.\n");
  return 1;
}
```

```
/* ****************************************************************** */
/*                                                                    */
/* AUTHOR: Epiphany Vera                                              */
/*         The University of Manitoba and                             */
/*         TRLabs (Telecommunications Research Labs)                  */
/*         Winnipeg, MB                                               */
/*         Canada.                                                    */
/*                                                                    */
/*         Copyright (c) 1996                                         */
/*                                                                    */
/* ****************************************************************** */

/* ****************************************************************** */
/*                                                                    */
/* FILENAME: encDecG.h                                                */
/*                                                                    */
/* ****************************************************************** */


#define     GSM_FRAME_SIZE 160
#define  GSM_BITSTREAM 33


void encodeGSM(short *source, unsigned char *c);
int  decodeGSM(unsigned char *c, short *output);
```

```
/* ******************************************************************** */
/*                                                                      */
/* AUTHOR: Epiphany Vera                                                */
/*         The University of Manitoba and                               */
/*         TRLabs (Telecommunications Research Labs)                    */
/*         Winnipeg, MB                                                 */
/*         Canada.                                                      */
/*                                                                      */
/*         Copyright (c) 1996                                           */
/*                                                                      */
/* ******************************************************************** */


/* ******************************************************************** */
/*                                                                      */
/* FILENAME:    encDecG.c   GSM Floating-point encoder and decoder      */
/*                                                                      */
/* DESCRIPTION: Has the encoder and decoder functions.                  */
/*                                                                      */
/*                                                                      */
/* PUBLIC FUNCTIONS:  encode(..)                                        */
/*                    decode(..)                                        */
/*                                                                      */
/* PRIVATE FUNCTIONS: floatToShortBuff(..)                              */
/*                                                                      */
/* ******************************************************************** */


#include <stdio.h>
#include <limits.h>
#include <string.h>

#include "defs.h"
#include "RPE.h"
#include "LTP.h"
#include "pre_post.h"
#include "packGSM.h"
#include "STP.h"
#include "encDecG.h"


static void floatToShortBuff(float *floatBuff, int length,
                             short *shortBuff);

void encodeGSM(short *inSpeech, unsigned char *bitStream)
{
   static float buffer[LOOK_BACK_SAMPLES+FRAME_SAMPLES];

   short Mc[SUBFRAMES], xmaxc[SUBFRAMES], xMc[13*SUBFRAMES];
                    /* Quantized Log Area Ratios.                       */
   short    qLAR[LPC_ORDER];
                    /* Quantized pitch lag.                             */
   short qLag[SUBFRAMES];
                    /* Quantized long term prediction gain.             */
   short qGain[SUBFRAMES];
```

```
                    /* LPC residual synthesized by LTP and IFS.     */
   float *synthLpcResidue;
                    /* LTP estimate of LPC residual.                */
   float *ltpEstLpcResidual;
                    /* Input speech after preprocessing.            */
   float filteredSpeech[FRAME_SAMPLES];
                    /* LPC residual.                                */
   float *lpcResidual;
                    /* LTP residual.                                */
   float ltpResidual[SUBFRAME_SAMPLES];
                    /* LTP residual synthesized by IFS module.      */
   float *synthLtpResidue;
   int        k,i;

   preprocess(inSpeech, FRAME_SAMPLES, filteredSpeech);
   lpcResidual = filteredSpeech;
   lpcAnalysis(lpcResidual, qLAR);

   /* The two buffers, synthLpcResidue and ltpEstLpcResidue together   */
   /* form the buffer used for long term prediction analysis.          */
                              /* Indexed from -LOOK_BACK_SAMPLES to 0 */
   synthLpcResidue  = buffer + LOOK_BACK_SAMPLES;
                              /* Indexed from 0 to SUBFRAME_SAMPLES-1 */
   ltpEstLpcResidual = synthLpcResidue;

   for (k=0; k<SUBFRAMES; k++)
   {
      ltpAnalysis(lpcResidual, synthLpcResidue, ltpResidual,
                ltpEstLpcResidual, &qLag[k], &qGain[k]);
      synthLtpResidue = ltpResidual;
      rpeResidualQuan(synthLtpResidue, &xmaxc[k], &Mc[k], &xMc[k*13]);

      for (i = 0; i < SUBFRAME_SAMPLES; i++)
      synthLpcResidue[i] = synthLtpResidue[i] + ltpEstLpcResidual[i];
      synthLpcResidue   += SUBFRAME_SAMPLES;
      ltpEstLpcResidual += SUBFRAME_SAMPLES;
      lpcResidual       += SUBFRAME_SAMPLES;
   }
   memcpy(buffer, buffer+FRAME_SAMPLES, LOOK_BACK_SAMPLES*sizeof(float));

   packGSM(qLAR, qLag, qGain, Mc, xmaxc, xMc, bitStream);
}


int decodeGSM(unsigned char *bitStream, short *output)
{
   static float buffer[LOOK_BACK_SAMPLES+SUBFRAME_SAMPLES];
                          /* Quantized Log Area Ratios.              */
   short    qLAR[LPC_ORDER];
                          /* Quantized pitch lag.                    */
   short qLag[SUBFRAMES];
                          /* Quantized long term prediction gain.    */
   short qGain[SUBFRAMES];
                          /* LPC residual synthesized by LTP and IFS.*/
```

```
   float *synthLpcResidue;
                             /* LTP residual synthesized by IFS module. */
   float synthLtpResidue[SUBFRAME_SAMPLES];
   short Mc[SUBFRAMES], xmaxc[SUBFRAMES], xmc[13*SUBFRAMES];
   float speech[FRAME_SAMPLES];
   int          j;


   unPackGSM(bitStream, qLAR, qLag, qGain, Mc, xmc, xmaxc);

   /* synthLpcResidue is indexed from -LOOK_BACK_SAMPLES to            */
   /* SUBFRAME_SAMPLES                                                 */
   synthLpcResidue  = buffer + LOOK_BACK_SAMPLES;

   for (j=0; j < SUBFRAMES; j++)
   {
      rpeResidualUnquan(xmaxc[j], Mc[j], &xmc[j*13], synthLtpResidue );
      ltpSynthesis(qLag[j], qGain[j], synthLtpResidue, synthLpcResidue);
   }

   /* synthLpcResidue is now all samples from the present frame.      */
   synthLpcResidue  = buffer;

   lpcSynthesis(qLAR, synthLpcResidue, speech);
   Postprocess(speech, FRAME_SAMPLES, speech);

   floatToShortBuff(speech, FRAME_SAMPLES, output);

   return 0;
}


static void floatToShortBuff(float *floatBuff, int length,
                             short *shortBuff)
{
  int   i;
  float temp;

  for(i=0; i<length; i++)
  {
    temp = floatBuff[i];
    if(temp>SHRT_MAX)
      shortBuff[i] = SHRT_MAX;
    else if(temp<SHRT_MIN)
      shortBuff[i] = SHRT_MIN;
    else
      shortBuff[i] = (short)temp;
  }
}
```

```
/* ******************************************************************* */
/*                                                                     */
/* AUTHOR: Epiphany Vera                                               */
/*         The University of Manitoba and                              */
/*         TRLabs (Telecommunications Research Labs)                   */
/*         Winnipeg, MB                                                */
/*         Canada.                                                     */
/*                                                                     */
/*         Copyright (c) 1996                                          */
/*                                                                     */
/* ******************************************************************* */

/* ******************************************************************* */
/*                                                                     */
/* FILENAME: RPE.h                                                     */
/*                                                                     */
/* ******************************************************************* */

void rpeResidualQuan(float    *residual, short *xmaxc, short *Mc,
                     short *xMc);
void rpeResidualUnquan(short xmaxcr, short Mcr, short *xMcr,
                       float *erp);
```

```
/* ******************************************************************* */
/*                                                                     */
/* AUTHOR: Epiphany Vera                                               */
/*         The University of Manitoba and                              */
/*         TRLabs (Telecommunications Research Labs)                   */
/*         Winnipeg, MB                                                */
/*         Canada.                                                     */
/*                                                                     */
/*         Copyright (c) 1996                                          */
/*                                                                     */
/* ******************************************************************* */

/* ******************************************************************* */
/*                                                                     */
/* FILENAME:     RPE.c                                                 */
/*                                                                     */
/* DESCRIPTION:                                                        */
/*                                                                     */
/*                                                                     */
/* PUBLIC FUNCTIONS:                                                   */
/*
/* PRIVATE FUNCTIONS:                                                  */
/*                                                                     */
/* ******************************************************************* */
```

```
#include <assert.h>
#include <math.h>
#include <limits.h>

#include "rpe.h"


#define     SATURATE(x)   ((x) < -32768 ? -32768 : (x) > 32767 ? 32767:
(short)(x))
#define ADD(a,b)      (SATURATE(((long)a+(long)b)))
#define     FLOAT_TO_SHORT(x)       \
       ((x) < SHRT_MIN ? SHRT_MIN : (x) > SHRT_MAX ? SHRT_MAX: (short)(x))


static const float H[11] = {-0.016F, -0.046F, 0.0F, 0.251F, 0.701F, 1.0F,
                             0.701F, 0.251F, 0.0F, -0.046F, -0.016F};

static void weightingFilter(float *residual, float *wResidual)
{
   int   i, j, index;
   float temp, sample;

   for(j=0; j<40; j++)
   {
      temp = 0.0F;
      for(i=0; i<=10; i++)
      {
         index = j-5+i;
         if((index>=0)&&(index<40))
            sample = residual[index];
         else
            sample = 0.0F;
         temp += sample*H[i];
      }
      wResidual[j] = temp;
   }
}


static void selectRpeGrid(float *residual, float *xM, short *Mc)
{
   int   j, i;
   float temp, sample, lastPower;


   lastPower = 0.0F;
   *Mc = 0;
   for (j=0; j<=3; j++)
   {
      temp = 0.0F;
      for (i = 0; i <= 12; i++)
      {
```

```
            sample = residual[j+3*i];
            temp += sample*sample;
         }
         if (temp>lastPower)
         {
            *Mc = j;
            lastPower = temp;
         }
   }

   /* Down sample signal. */
   for (i = 0; i <= 12; i ++)
      xM[i] = residual[(*Mc)+3*i];
}


static void unquantizeMax(short xmaxc, short *exp, short *mant)
{
   *exp = 0;
   if(xmaxc>15)
      *exp = (xmaxc>>3) - 1;
   *mant = xmaxc - (*exp << 3);

   if(*mant==0)
   {
      *exp  = -4;
      *mant = 7;
   }
   else
   {
      while (*mant <= 7)
      {
         *mant = *mant << 1 | 1;
         (*exp)--;
      }
         *mant -= 8;
   }
}


/* Inverse mantisa table */
static short NRFAC[8] = {29128, 26215, 23832, 21846, 20165, 18725,
                         17476, 16384 };


static void ApcmQuantization(float *xM, short *xMc, short *mant,
                             short *exp, short *xmaxc)
{
   int      i, itest;
   float tempF;
   short xmax, invMant, temp;

   xmax = 0;
   for(i=0; i<13; i++)
   {
```

```
        tempF = (float)fabs(xM[i]);
        if(tempF>xmax)
           xmax = (short)tempF;
     }

     *exp   = 0;
     temp   = xmax>>9;
     itest = 0;

     for (i=0; i<=5; i++)
     {
        if(temp <= 0)
           itest=1;
        temp = temp>>1;

        if(itest==0)
           (*exp)++;
     }

     temp = *exp + 5;

     *xmaxc = ADD((xmax>>temp),(*exp<<3));

     unquantizeMax(*xmaxc, exp, mant);

     invMant = NRFAC[*mant];

     for (i=0; i<=12; i++)
     {
        temp = FLOAT_TO_SHORT(xM[i]) << (6-*exp);
        temp = (short)((temp*(long)invMant)>>(15+12));
        xMc[i] = temp + 4;
     }
}

/* Normalized direct mantisa table */
const static short FAC[8]=
             {18431, 20479, 22527, 24575, 26623, 28671, 30719, 32767 };


static void ApcmUnquantization(short *xMc, short mant, short exp,
                               short *xMp)
{
        int   i;
        short temp, temp1, temp2, temp3;

        temp1 = FAC[mant];
        temp2 = 6-exp;
        temp3 = 1<<(temp2-1);

        for (i = 13; i--;)
     {
        temp = (*xMc++ << 1) - 7;
        temp <<= 12;
        temp = (short)((temp1*(long)temp + 16384)>>15);
```

```
        temp = ADD(temp, temp3);
        *xMp++ = temp>>temp2;
    }
}


static void RpeUpsample(short Mc, short *xMp, float *ep)
{
        int   i;

        for (i=0; i<40; i++)
        ep[i] = 0.0F;
        for (i=0; i<=12; i++)
        ep[Mc+(3*i)] = (float)(xMp[i]);
}


void rpeResidualQuan(float    *residual, short *xmaxc, short *Mc,
                     short *xMc)
{
        float x[40], xM[13];
        short xMp[13], mant, exp;

        weightingFilter(residual, x);
        selectRpeGrid(x, xM, Mc);

        ApcmQuantization(xM, xMc, &mant, &exp, xmaxc);
        ApcmUnquantization(xMc, mant, exp, xMp);

        RpeUpsample(*Mc, xMp, residual);

}

void rpeResidualUnquan(short xmaxcr, short Mcr, short *xMcr, float *erp)
{
        short exp, mant;
        short xMp[13];

        unquantizeMax( xmaxcr, &exp, &mant );
        ApcmUnquantization( xMcr, mant, exp, xMp );
        RpeUpsample( Mcr, xMp, erp );

}
```

```
/* ******************************************************************* */
/*                                                                     */
/* AUTHOR: Epiphany Vera                                               */
/*         The University of Manitoba and                              */
/*         TRLabs (Telecommunications Research Labs)                   */
/*         Winnipeg, MB                                                */
/*         Canada.                                                     */
/*                                                                     */
/*         Copyright (c) 1996                                          */
/*                                                                     */
/* ******************************************************************* */

/* ******************************************************************* */
/*                                                                     */
/* FILENAME: packGSM.h                                                 */
/*                                                                     */
/* ******************************************************************* */

void packGSM(short *LARc, short *Nc, short *bc, short *Mc, short *xmaxc,
         short *xmc, unsigned char *c);
int unPackGSM(unsigned char *c, short *LARc, short *Nc, short *bc,
              short *Mc, short *xmc, short *xmaxc);
```

```
/* ******************************************************************* */
/*                                                                     */
/* AUTHOR: Epiphany Vera                                               */
/*         The University of Manitoba and                              */
/*         TRLabs (Telecommunications Research Labs)                   */
/*         Winnipeg, MB                                                */
/*         Canada.                                                     */
/*                                                                     */
/*         Copyright (c) 1996                                          */
/*                                                                     */
/* ******************************************************************* */

/* ******************************************************************* */
/*                                                                     */
/* FILENAME:     packGSM.c                                             */
/*                                                                     */
/* DESCRIPTION:  Routines for packing quantized parameters into a      */
/*               bit-stream.                                           */
/*                                                                     */
/*                                                                     */
/* PUBLIC FUNCTIONS:  packGSM(..)                                      */
/*                    unpackGSM(..)                                    */
/*                                                                     */
/* PRIVATE FUNCTIONS:                                                  */
/*                                                                     */
/* ******************************************************************* */
```

```
#include "packGSM.h"

#define     GSM_ID      0xD

void packGSM(short *LARc, short *Nc, short *bc, short *Mc, short *xmaxc,
             short *xmc, unsigned char *c)
{
    *c++ =   ((GSM_ID & 0xF) << 4)
             | ((LARc[0] >> 2) & 0xF);
    *c++ =   ((LARc[0] & 0x3) << 6)
             | (LARc[1] & 0x3F);
    *c++ =   ((LARc[2] & 0x1F) << 3)
             | ((LARc[3] >> 2) & 0x7);
    *c++ =   ((LARc[3] & 0x3) << 6)
             | ((LARc[4] & 0xF) << 2)
             | ((LARc[5] >> 2) & 0x3);
    *c++ =   ((LARc[5] & 0x3) << 6)
             | ((LARc[6] & 0x7) << 3)
             | (LARc[7] & 0x7);
    *c++ =   ((Nc[0] & 0x7F) << 1)
             | ((bc[0] >> 1) & 0x1);
    *c++ =   ((bc[0] & 0x1) << 7)
             | ((Mc[0] & 0x3) << 5)
             | ((xmaxc[0] >> 1) & 0x1F);
    *c++ =   ((xmaxc[0] & 0x1) << 7)
             | ((xmc[0] & 0x7) << 4)
             | ((xmc[1] & 0x7) << 1)
             | ((xmc[2] >> 2) & 0x1);
    *c++ =   ((xmc[2] & 0x3) << 6)
             | ((xmc[3] & 0x7) << 3)
             | (xmc[4] & 0x7);
    *c++ =   ((xmc[5] & 0x7) << 5)
             | ((xmc[6] & 0x7) << 2)
             | ((xmc[7] >> 1) & 0x3);
    *c++ =   ((xmc[7] & 0x1) << 7)
             | ((xmc[8] & 0x7) << 4)
             | ((xmc[9] & 0x7) << 1)
             | ((xmc[10] >> 2) & 0x1);
    *c++ =   ((xmc[10] & 0x3) << 6)
             | ((xmc[11] & 0x7) << 3)
             | (xmc[12] & 0x7);
    *c++ =   ((Nc[1] & 0x7F) << 1)
             | ((bc[1] >> 1) & 0x1);
    *c++ =   ((bc[1] & 0x1) << 7)
             | ((Mc[1] & 0x3) << 5)
             | ((xmaxc[1] >> 1) & 0x1F);
    *c++ =   ((xmaxc[1] & 0x1) << 7)
             | ((xmc[13] & 0x7) << 4)
             | ((xmc[14] & 0x7) << 1)
             | ((xmc[15] >> 2) & 0x1);
    *c++ =   ((xmc[15] & 0x3) << 6)
             | ((xmc[16] & 0x7) << 3)
             | (xmc[17] & 0x7);
    *c++ =   ((xmc[18] & 0x7) << 5)
```

```
                | ((xmc[19] & 0x7) << 2)
                | ((xmc[20] >> 1) & 0x3);
    *c++ =      ((xmc[20] & 0x1) << 7)
                | ((xmc[21] & 0x7) << 4)
                | ((xmc[22] & 0x7) << 1)
                | ((xmc[23] >> 2) & 0x1);
    *c++ =      ((xmc[23] & 0x3) << 6)
                | ((xmc[24] & 0x7) << 3)
                | (xmc[25] & 0x7);
    *c++ =      ((Nc[2] & 0x7F) << 1)
                | ((bc[2] >> 1) & 0x1);
    *c++ =      ((bc[2] & 0x1) << 7)
                | ((Mc[2] & 0x3) << 5)
                | ((xmaxc[2] >> 1) & 0x1F);
    *c++ =      ((xmaxc[2] & 0x1) << 7)
                | ((xmc[26] & 0x7) << 4)
                | ((xmc[27] & 0x7) << 1)
                | ((xmc[28] >> 2) & 0x1);
    *c++ =      ((xmc[28] & 0x3) << 6)
                | ((xmc[29] & 0x7) << 3)
                | (xmc[30] & 0x7);
    *c++ =      ((xmc[31] & 0x7) << 5)
                | ((xmc[32] & 0x7) << 2)
                | ((xmc[33] >> 1) & 0x3);
    *c++ =      ((xmc[33] & 0x1) << 7)
                | ((xmc[34] & 0x7) << 4)
                | ((xmc[35] & 0x7) << 1)
                | ((xmc[36] >> 2) & 0x1);
    *c++ =      ((xmc[36] & 0x3) << 6)
                | ((xmc[37] & 0x7) << 3)
                | (xmc[38] & 0x7);
    *c++ =      ((Nc[3] & 0x7F) << 1)
                | ((bc[3] >> 1) & 0x1);
    *c++ =      ((bc[3] & 0x1) << 7)
                | ((Mc[3] & 0x3) << 5)
                | ((xmaxc[3] >> 1) & 0x1F);
    *c++ =      ((xmaxc[3] & 0x1) << 7)
                | ((xmc[39] & 0x7) << 4)
                | ((xmc[40] & 0x7) << 1)
                | ((xmc[41] >> 2) & 0x1);
    *c++ =      ((xmc[41] & 0x3) << 6)
                | ((xmc[42] & 0x7) << 3)
                | (xmc[43] & 0x7);
    *c++ =      ((xmc[44] & 0x7) << 5)
                | ((xmc[45] & 0x7) << 2)
                | ((xmc[46] >> 1) & 0x3);
    *c++ =      ((xmc[46] & 0x1) << 7)
                | ((xmc[47] & 0x7) << 4)
                | ((xmc[48] & 0x7) << 1)
                | ((xmc[49] >> 2) & 0x1);
    *c++ =      ((xmc[49] & 0x3) << 6)
                | ((xmc[50] & 0x7) << 3)
                | (xmc[51] & 0x7);
}
```

```
int unPackGSM(unsigned char *c, short *LARc, short *Nc, short *bc,
              short *Mc, short *xmc, short *xmaxc)
{

   if (((*c >> 4) & 0x0F) != GSM_ID)
      return -1;

   LARc[0]  = (*c++ & 0xF) << 2;
   LARc[0] |= (*c >> 6) & 0x3;
   LARc[1]  = *c++ & 0x3F;
   LARc[2]  = (*c >> 3) & 0x1F;
   LARc[3]  = (*c++ & 0x7) << 2;
   LARc[3] |= (*c >> 6) & 0x3;
   LARc[4]  = (*c >> 2) & 0xF;
   LARc[5]  = (*c++ & 0x3) << 2;
   LARc[5] |= (*c >> 6) & 0x3;
   LARc[6]  = (*c >> 3) & 0x7;
   LARc[7]  = *c++ & 0x7;
   Nc[0]  = (*c >> 1) & 0x7F;
   bc[0]  = (*c++ & 0x1) << 1;
   bc[0] |= (*c >> 7) & 0x1;
   Mc[0]  = (*c >> 5) & 0x3;
   xmaxc[0]  = (*c++ & 0x1F) << 1;
   xmaxc[0] |= (*c >> 7) & 0x1;
   xmc[0]  = (*c >> 4) & 0x7;
   xmc[1]  = (*c >> 1) & 0x7;
   xmc[2]  = (*c++ & 0x1) << 2;
   xmc[2] |= (*c >> 6) & 0x3;
   xmc[3]  = (*c >> 3) & 0x7;
   xmc[4]  = *c++ & 0x7;
   xmc[5]  = (*c >> 5) & 0x7;
   xmc[6]  = (*c >> 2) & 0x7;
   xmc[7]  = (*c++ & 0x3) << 1;
   xmc[7] |= (*c >> 7) & 0x1;
   xmc[8]  = (*c >> 4) & 0x7;
   xmc[9]  = (*c >> 1) & 0x7;
   xmc[10]  = (*c++ & 0x1) << 2;
   xmc[10] |= (*c >> 6) & 0x3;
   xmc[11]  = (*c >> 3) & 0x7;
   xmc[12]  = *c++ & 0x7;
   Nc[1]  = (*c >> 1) & 0x7F;
   bc[1]  = (*c++ & 0x1) << 1;
   bc[1] |= (*c >> 7) & 0x1;
   Mc[1]  = (*c >> 5) & 0x3;
   xmaxc[1]  = (*c++ & 0x1F) << 1;
   xmaxc[1] |= (*c >> 7) & 0x1;
   xmc[13]  = (*c >> 4) & 0x7;
   xmc[14]  = (*c >> 1) & 0x7;
   xmc[15]  = (*c++ & 0x1) << 2;
   xmc[15] |= (*c >> 6) & 0x3;
   xmc[16]  = (*c >> 3) & 0x7;
   xmc[17]  = *c++ & 0x7;
   xmc[18]  = (*c >> 5) & 0x7;
   xmc[19]  = (*c >> 2) & 0x7;
   xmc[20]  = (*c++ & 0x3) << 1;
```

```
    xmc[20] |= (*c >> 7) & 0x1;
    xmc[21]  = (*c >> 4) & 0x7;
    xmc[22]  = (*c >> 1) & 0x7;
    xmc[23]  = (*c++ & 0x1) << 2;
    xmc[23] |= (*c >> 6) & 0x3;
    xmc[24]  = (*c >> 3) & 0x7;
    xmc[25]  = *c++ & 0x7;
    Nc[2]   = (*c >> 1) & 0x7F;
    bc[2]   = (*c++ & 0x1) << 1;
    bc[2]  |= (*c >> 7) & 0x1;
    Mc[2]   = (*c >> 5) & 0x3;
    xmaxc[2]  = (*c++ & 0x1F) << 1;
    xmaxc[2] |= (*c >> 7) & 0x1;
    xmc[26]  = (*c >> 4) & 0x7;
    xmc[27]  = (*c >> 1) & 0x7;
    xmc[28]  = (*c++ & 0x1) << 2;
    xmc[28] |= (*c >> 6) & 0x3;
    xmc[29]  = (*c >> 3) & 0x7;
    xmc[30]  = *c++ & 0x7;
    xmc[31]  = (*c >> 5) & 0x7;
    xmc[32]  = (*c >> 2) & 0x7;
    xmc[33]  = (*c++ & 0x3) << 1;
    xmc[33] |= (*c >> 7) & 0x1;
    xmc[34]  = (*c >> 4) & 0x7;
    xmc[35]  = (*c >> 1) & 0x7;
    xmc[36]  = (*c++ & 0x1) << 2;
    xmc[36] |= (*c >> 6) & 0x3;
    xmc[37]  = (*c >> 3) & 0x7;
    xmc[38]  = *c++ & 0x7;
    Nc[3]   = (*c >> 1) & 0x7F;
    bc[3]   = (*c++ & 0x1) << 1;
    bc[3]  |= (*c >> 7) & 0x1;
    Mc[3]   = (*c >> 5) & 0x3;
    xmaxc[3]  = (*c++ & 0x1F) << 1;
    xmaxc[3] |= (*c >> 7) & 0x1;
    xmc[39]  = (*c >> 4) & 0x7;
    xmc[40]  = (*c >> 1) & 0x7;
    xmc[41]  = (*c++ & 0x1) << 2;
    xmc[41] |= (*c >> 6) & 0x3;
    xmc[42]  = (*c >> 3) & 0x7;
    xmc[43]  = *c++ & 0x7;
    xmc[44]  = (*c >> 5) & 0x7;
    xmc[45]  = (*c >> 2) & 0x7;
    xmc[46]  = (*c++ & 0x3) << 1;
    xmc[46] |= (*c >> 7) & 0x1;
    xmc[47]  = (*c >> 4) & 0x7;
    xmc[48]  = (*c >> 1) & 0x7;
    xmc[49]  = (*c++ & 0x1) << 2;
    xmc[49] |= (*c >> 6) & 0x3;
    xmc[50]  = (*c >> 3) & 0x7;
    xmc[51]  = *c & 0x7;
    return 0;
}
```

APPENDIX C:
SOURCE CODE FOR SELF-AFFINE FRACTAL EXCITED LINEAR PREDICTIVE CODEC

```
/* ****************************************************************** */
/*                                                                    */
/* AUTHOR: Epiphany Vera                                              */
/*         The University of Manitoba and                             */
/*         TRLabs (Telecommunications Research Labs)                  */
/*         Winnipeg, MB                                               */
/*         Canada.                                                    */
/*                                                                    */
/*         Copyright (c) 1996                                         */
/*                                                                    */
/* ****************************************************************** */


/* ****************************************************************** */
/*                                                                    */
/* FILENAME:            testGSM.c                                     */
/*                                                                    */
/* DESCRIPTION:         Program for testing the GSM codec.            */
/*                                                                    */
/*                                                                    */
/* PUBLIC FUNCTIONS:    main(..)                                      */
/*                                                                    */
/* PRIVATE FUNCTIONS:   none                                          */
/*                                                                    */
/* ****************************************************************** */


#include <stdio.h>
#include <conio.h>

#include "encDecI.h"


int main(void)
{
   unsigned char bitStream[GSM_BITSTREAM];
   short PCMStream[GSM_FRAME_SIZE];
   char ch, quit;
   unsigned long Frame;
   FILE *InFile, *OutFile;

   quit = 0;
   do
   {
      printf
      ("Would you like to Analyze, Synthesize, Both, Quit? (A/S/B/Q): ");
      ch = getche();
      printf ("\n");
      if (ch=='A'||ch=='a')
      {
         InFile = fopen("..\\data\\voiced.raw","rb");
         OutFile = fopen("test.bit","wb");
```

```
        if((InFile != 0)&&(OutFile != 0))
        {
           Frame=0;
           while(fread(PCMStream,sizeof(short),GSM_FRAME_SIZE,InFile)
                == GSM_FRAME_SIZE)
           {
              printf("Frame: %lu\n",Frame);
              encodeIFS(PCMStream, bitStream);
              fwrite(bitStream,sizeof(unsigned char),
                     GSM_BITSTREAM,OutFile);
              Frame=Frame+1;
           }
           fclose(InFile);
           fclose(OutFile);
        }
        else
        {
           printf("An error occured while opening files or in memory \
                 allocation.\n");
           fclose(InFile);
           fclose(OutFile);
           return -1;
        }
     }
     else if (ch=='S'||ch=='s')
     {
        OutFile = fopen("test_syn.raw","wb");
        InFile = fopen("test.bit","rb");
        if((OutFile != 0)&&(InFile != 0))
        {
           Frame=0;
           do
           {
              fread(bitStream,sizeof(unsigned char),
                    GSM_BITSTREAM,InFile);
              printf("Frame: %lu\n",Frame);
              decodeIFS(bitStream, PCMStream);
              fwrite(PCMStream,sizeof(short),GSM_FRAME_SIZE,OutFile);
              Frame=Frame+1;
           }while (!feof(InFile));
        }
        fclose(OutFile);
        fclose(InFile);
     }
     else if (ch=='B'||ch=='b')
     {
        InFile = fopen("..\\data\\voiced.raw","rb");
        OutFile = fopen("test_syn.raw","wb");
        if((InFile != 0)&&(OutFile != 0))
        {
           Frame=0;
           while(fread(PCMStream,sizeof(short),GSM_FRAME_SIZE,InFile)
                == GSM_FRAME_SIZE)
           {
              printf("Frame: %lu\n",Frame);
```

```
                encodeIFS(PCMStream, bitStream);
                decodeIFS(bitStream, PCMStream);
                fwrite(PCMStream,sizeof(short),GSM_FRAME_SIZE,OutFile);
                Frame=Frame+1;
             }
          }
          fclose(InFile);
          fclose(OutFile);
       }
       else if (ch=='Q'||ch=='q')
       {
          quit = 1;
       }
    } while (!quit);
    printf ("\n\nOperation Complete, program terminating.\n");
    return 1;
}
```

```
/* ****************************************************************** */
/*                                                                    */
/* AUTHOR: Epiphany Vera                                              */
/*         The University of Manitoba and                             */
/*         TRLabs (Telecommunications Research Labs)                  */
/*         Winnipeg, MB                                               */
/*         Canada.                                                    */
/*                                                                    */
/*         Copyright (c) 1996                                         */
/*                                                                    */
/* ****************************************************************** */


/* ****************************************************************** */
/*                                                                    */
/* FILENAME:     ifs.c                                                */
/*                                                                    */
/* DESCRIPTION:  Self-affine fractal modelling functions.             */
/*                                                                    */
/*                                                                    */
/* PUBLIC FUNCTIONS:  ifsSynthesis(..)                                */
/*                    ifsAnalysis(..)                                 */
/*                                                                    */
/* PRIVATE FUNCTIONS: round(..)                                       */
/*                    calcMaps(..)                                    */
/*                    synthAttractor(..)                              */
/*                    calcContFactor(..)                              */
/*                    calcEstimation(..)                              */
/*                    calcParam(..)                                   */
/*                    quanParams(..)                                  */
/*                    unquanParams(..)                                */
/*                                                                    */
/* ****************************************************************** */
```

```
/***** HEADER FILES *****/
#include <conio.h>
#include <stdio.h>
#include <limits.h>
#include <float.h>
#include <math.h>
#include <stdlib.h>

#define maxMaps 51    //Maximum number of maps to be used +1
#define frameSize 160
#define maxIterations 1000 /* Number of iterations done before          */
                           /* non-convergence is assumed                */

typedef struct {float x,y,a,e,c,d,f;} mapType;
typedef mapType mapsType[maxMaps]; //Array of map parameters
                                   //Row 0 contains: x_0 y_0 n
                                      //where n is the number of maps
                                   //The remaining n rows contain:
                                      // x_i y_i a_i e_i c_i d_i f_i
typedef int frameType[frameSize];
typedef float mapParamType[maxMaps];

static void getParam(FILE *paramFile, float*x, float*y, float*d);
static void storeAttractor(FILE *outFile, float *G);
static void getPCMData(FILE *PCMFile, float *PCMData);
static void saveParam(FILE*paramFile,float*x,float*y,float*d);
FILE *errorFile;


/************************************************************************/
/*                           SHARED ROUTINES                            */
/************************************************************************/


static int round(float x)
{
   if ((0.5-(ceil(x)-x)>0))
      return (int)ceil(x);
   else
      return (int)floor(x);
}

/************************************************************************/
/*                        END OF SHARED ROUTINES                        */
/************************************************************************/
```

```
/***************************************************************************/
/*                          SYNTHESIS ROUTINES                             */
/***************************************************************************/

static void calcMaps(float*x, float*y, float*a, float*e, float*c,
                     float*d, float*f)
{
   int numOfMaps, i;

   numOfMaps = d[0];

   for (i=1; i<=numOfMaps; i++)
   {
      //Using endpoint constraints, parameters of the affine maps
      // are calculated below.
      a[i] = (x[i] - x[i-1])/(x[numOfMaps] - x[0]);
      e[i] = (x[numOfMaps]*x[i-1]-x[0]*x[i])/(x[numOfMaps]-x[0]);
      c[i] = (y[i]-y[i-1])/(x[numOfMaps]-x[0])-d[i]*(y[numOfMaps]-y[0])/
                      (x[numOfMaps]-x[0]);
      f[i] = (x[numOfMaps]*y[i-1]-x[0]*y[i])/(x[numOfMaps]-x[0])-
                  d[i]*(x[numOfMaps]*y[0]-x[0]*y[numOfMaps])/
              (x[numOfMaps]-x[0]);
   }
}

static void synthAttractor(float*a,float*e,float*c,float*d,float*f,
                           float*G)
{
   const int maxSqaureError=0; //Maximum error in attractor calculation
   float  A1[frameSize+1], A2[frameSize+1];
   int    A1x[frameSize+1], A2x[frameSize+1];
   int numOfMaps, i, k, errorSame, iterations;
   float error, lastError;


   error=(float)INT_MAX;
   numOfMaps = d[0];
   for (i=0; i<frameSize; i++) //Initialise data array
   {
      A1[i]=100;
      A1x[i]=i;
      G[i]=0;
   }
   iterations=0;
   errorSame=0;
   lastError=0;
   while((error>maxSqaureError)&&(errorSame<5)&&
         (iterations<maxIterations))
   {
      error=0;
      for (i=1; i<=numOfMaps; i++)
      {
         for (k=0; k<frameSize; k++)
         {
               A2x[k]=round(a[i]*A1x[k] + e[i]);
```

```
                A2[k] =round(c[i]*A1x[k] + d[i]*A1[k] + f[i]);
            }
            for (k=0; k<frameSize; k++)
            {
                G[A2x[k]]=A2[k];
            }
        }
        for (k=0; k<frameSize; k++)
            error += (fabs(A1[k]-G[k]));
        for (k=0; k<frameSize; k++)  //Get new A for more iterations
            A1[k]=G[k];

        if (lastError==error)
            errorSame++;
        lastError=error;
        iterations++;
    }
}

static void synth(void)
{
    float G[frameSize];
    mapParamType x,y,a,e,c,d,f;
    FILE *synthFile, *paramFile;
    int frame=0;

    paramFile = fopen("residual.ifs","rt");
    synthFile = fopen("syn_res.rst","wt");
    if ((paramFile != 0)&&(synthFile != 0))
    do
    {
        printf("Processing frame %d.\n",frame);
        getParam(paramFile,x,y,d);
        calcMaps(x,y,a,e,c,d,f);
        synthAttractor(a,e,c,d,f,G);
        storeAttractor(synthFile,G);
        frame++;
    }while (!feof(paramFile));
    else
    printf("ERROR: Could not open input file and\\or create output file");
    fclose(synthFile);
    fclose(paramFile);
}

/*************************************************************************/
/*                        END OF SYNTHESIS ROUTINES                       */
/*************************************************************************/
```

```
/*************************************************************************/
/*                          ANALYSIS ROUTINES                            */
/*************************************************************************/

static float calcD(float *PCMData, int p, int Yp,int q, int Yq)
{
   float num, den; //numerator & denominator of d
   float Xi, An, Bn, a, e, F, I;
   int n, m;

   num = 0.0F;
   den = 0.0F;
   I=0;
   F=(frameSize-1);
   a=(q-p)/(F-I);
   e=(F*p-I*q)/(F-I);
   for (n=0; n<frameSize; n++)
   {
      m=round(a*n+e);
      Xi=(F-n)/(F-I);
      An=PCMData[n]-(Xi*PCMData[(int)I]+(1-Xi)*PCMData[(int)F]);
      Bn=PCMData[m]-(Xi*Yp+(1-Xi)*Yq);
      num += Bn*An;
      den += An*An;
   }
   if ((den*den)<0.00000001)
      return FLT_MAX;
   else
      return (float)(num/den);
}


static void calcParam(float *PCMData, float*x,float*y,float*d )
{
   int p, Yp, q, Yq, i, k, numOfMaps, numOfTryMaps, best;
   float testD, minError, errorCheck;
   float testA, testE, testC, testF;
   int tempX[frameSize-1], tempY[frameSize-1];
   float tempD[frameSize-1];
   double error[frameSize-1];
   frameType Hx, H, Hi;

   float I=0;          //The initial and final points of the PCM data
   float YI=PCMData[(int)I];
   float F=frameSize-1;
   float YF=PCMData[(int)F];

   x[0]=I;          //Set the first point in the PCM as the initial point
   y[0]=PCMData[0];
   numOfMaps=0;     //Total number of maps found

   p=0;
   Yp=y[0];
   i=0;
   q=0;
```

```
while (p<(frameSize-1))
{
   numOfTryMaps=0; //Number of candidate parameters for the map i
   while (i<(frameSize-1))
   {
      testD=2.0;   //initialise the temporary d to allow loop to start
      numOfTryMaps++;
      while((((testD*testD)>=1)||((testD*testD)<0.000000001))&&
                (testD!=0))&&(i<frameSize))
      {
         i++;
         q=i;
         Yq=PCMData[i];
         testD=calcD(PCMData,p,Yp,q,Yq);
      }
      if (i<frameSize)
      {
         tempX[numOfTryMaps] = q;
         tempY[numOfTryMaps] = Yq;
         tempD[numOfTryMaps] = testD;
         testA = (q-p)/(F-I);
         testE = (F*p-I*q)/(F-I);
         testC = (Yq-Yp)/(F-I)-testD*(YF-YI)/(F-I);
         testF = (F*Yp-I*Yq)/(F-I)-testD*(F*YI-I*YF)/(F-I);
         for (k=0; k<frameSize; k++)
         {
               Hx[k]=round(testA*k + testE);
               H[k] =round(testC*k + testD*PCMData[k] + testF);
         }

         for (k=0; k<frameSize; k++)      //Create the function Hi
         {
               Hi[Hx[k]]=H[k];//This function is a map of the original
         }     //into the interpolation interval of the current map
         error[numOfTryMaps]=0;
         for (k=p; k<=q; k++)
               error[numOfTryMaps] = error[numOfTryMaps]+ fabs(Hi[k]-
                                       PCMData[k]);
                                       //Normalize the error
         error[numOfTryMaps] = (double)error[numOfTryMaps]/(q-p+1);
         error[numOfTryMaps] = error[numOfTryMaps]-
                                 (((float)tempX[numOfTryMaps]-p)*0.2);
      }
      else
         numOfTryMaps--;
   }
   if (numOfTryMaps>0)
   {
      minError=FLT_MAX;
      for (k=numOfTryMaps; k>0; k--)
      {
         errorCheck=minError-error[k]-0.0;   /*Best value so far is */
                                 /* 0.3(no origin shift) 0.2 with shift */
         if ((errorCheck>0)&&( (tempX[k]-p)>1)||
            (tempX[k]>(frameSize-5)) ) /* Should experiment with    */
```

```
                /* putting a min value with which it has to be greater */
            {//Imposed that map has to have width > 1
                best=k;
                minError=error[k];
            }
        }
        numOfMaps++;
        if (numOfMaps>maxMaps)
           printf("**** ERROR: Maximum number of maps exceeded. ****");
        x[numOfMaps]=tempX[best];
        y[numOfMaps]=tempY[best];
        d[numOfMaps]=tempD[best];
        p=tempX[best];
        Yp=tempY[best];
        i=tempX[best];
     }
     else
     {
        x[numOfMaps]=frameSize-1;
        p=frameSize-1;
     }
   }
   d[0]=numOfMaps;
}


static void analyse(void)
{
   float PCMData[frameSize];
   mapParamType x,y,d;
   FILE *PCMFile, *paramFile;
   int frame=0;
   int doAll=1;  //0 to process single frame, 1 otherwise

   PCMFile = fopen("residual.rst","rt");
   paramFile = fopen("residual.ifs","wt");
   if ((paramFile != 0)&&(PCMFile != 0))
   do
   {
      printf("Processing frame %d.\n",frame);
      getPCMData(PCMFile,PCMData);
      calcParam(PCMData, x,y,d);
      saveParam(paramFile,x,y,d);
      frame++;
   } while ((!feof(PCMFile))&&(doAll));
   else
      printf(
      "ERROR: Could not open input file and\\or create output file.");
   fclose(PCMFile);
   fclose(paramFile);
}

/**************************************************************************/
/*                          END OF ANALYSIS ROUTINES                      */
/**************************************************************************/
```

```
/***********************************************************************/
/*                         FILE I/O ROUTINES                          */
/***********************************************************************/

static void getParam(FILE *paramFile, float*x, float*y, float*d)
{
   int i;

   fscanf(paramFile,"%f\n",&(d[0]));
   fscanf(paramFile,"%f %f\n",&(x[0]),&(y[0]));
   for (i=1; i<=(int)d[0]; i++)
      fscanf(paramFile, "%f %f %f\n",&(x[i]),&(y[i]),&(d[i]));
}


static void storeAttractor(FILE *outFile, float *G)
{
   int i;

   for (i=0; i<frameSize; i++)
      fprintf(outFile,"%d\n",(int)((G[i]-16384)*2));
      //fwrite(&(G[i]),sizeof(int),1,outFile);
}

static void getPCMData(FILE *PCMFile,float *PCMData)
{
   int i, dataRead;

   for (i=0; i<frameSize; i++)
   {
      fscanf(PCMFile,"%d\n",&dataRead);
      PCMData[i] = dataRead/2.0+16384;
   }
}

static void saveParam(FILE*paramFile,float*x,float*y,float*d)
{
   int i;

   fprintf(paramFile,"%d\n",(int)d[0]);
   fprintf(paramFile,"%d %d\n",(int)x[0],(int)y[0]);
   for (i=1; i<=(int)d[0]; i++)
      fprintf(paramFile, "%d %d %f\n",(int)x[i],(int)y[i],d[i]);
}

/***********************************************************************/
/*                      END OF FILE I/O ROUTINES                      */
/***********************************************************************/
```

```
/***************************************************************************/
/*                              MAIN ROUTINE                               */
/***************************************************************************/

int main(void)
{
   char ch;

   errorFile=fopen("error.txt","wt");
   printf("Would you like to (A)nalyse or (S)ynthesize? Enter A or S: ");
   ch=getch();
   printf("\n");
   if  ((ch==82)||(ch==115))
      synth();
   else if ((ch==65)||(ch==97))
      analyse();
   else printf("Invalid key entered.");

   fclose(errorFile);
   printf("Operation Complete.\n");
   return 0;
}

/***************************************************************************/
/*                          END OF MAIN ROUTINE                            */
/***************************************************************************/
```

APPENDIX D:
SOURCE CODE FOR THE SPEECH COMPRESSION LAB

```
/* ****************************************************************** */
/*                                                                    */
/* AUTHOR: Epiphany Vera                                              */
/*         The University of Manitoba and                             */
/*         TRLabs (Telecommunications Research Labs)                  */
/*         Winnipeg, MB                                               */
/*         Canada.                                                    */
/*                                                                    */
/*         Copyright (c) 1996                                         */
/*                                                                    */
/* ****************************************************************** */


/* ****************************************************************** */
/*                                                                    */
/* FILENAME:    spechlb2.cpp                                          */
/*                                                                    */
/* DESCRIPTION: SpeechLab Application class.                          */
/*                                                                    */
/*                                                                    */
/* PUBLIC FUNCTIONS:  OwlMain(..)                                     */
/*                                                                    */
/* CLASSES:           TMyApp                                          */
/*                    TDrawView                                       */
/*                    TLine                                           */
/*                    TSoundBar                                       */
/*                    TDrawDocument                                   */
/*                                                                    */
/* ****************************************************************** */



/* ****************************************************************** */
/*                  CLASS METHOD AND PARENT DETAILS                   */
/* ****************************************************************** */

/* ****************************************************************** */
/*                                                                    */
/* CLASS:            TMyApp                                           */
/*                                                                    */
/* PARENT CLASS:     TApplication                                     */
/*                                                                    */
/* OVERIDE METHODS:                                                   */
/*                   InitInstance(..)                                 */
/*                   InitMainWindow(..)                               */
/*                        // Event handlers                           */
/*                   EvNewView(..)                                    */
/*                   EvCloseView(..)                                  */
/*                   EvDropFiles(..)                                  */
/*                   CmAbout(..)                                      */
/*                                                                    */
/* ****************************************************************** */
```

```
/* ***************************************************************** */
/*                                                                   */
/* CLASS:        TSoundBar                                           */
/*                                                                   */
/* PARENT CLASS: THSlider                                            */
/*                                                                   */
/* METHODS:                                                          */
/*               SetInfo(..)                                         */
/*               SetName(..)                                         */
/*               SBLineUp(..)                                        */
/*               SBLineDown(..)                                      */
/*               SBPageUp(..)                                        */
/*               SBPageDown(..)                                      */
/*               SBThumbPosition(..)                                 */
/*               SBTop(..)                                           */
/*               SBBottom(..)                                        */
/*               ReposAndPlay(..)                                    */
/*                                                                   */
/* ***************************************************************** */

/* ***************************************************************** */
/*                                                                   */
/* CLASS:        TLine                                               */
/*                                                                   */
/* PARENT CLASS: TPoints                                             */
/*                                                                   */
/* METHODS:                                                          */
/*               QueryPenSize(..)                                    */
/*               QueryColor(..)                                      */
/*               SetPen(..)                                          */
/*               SetPen(..)                                          */
/*               GetPenSize(..)                                      */
/*               GetPenColor(..)                                     */
/*               Draw(..)                                            */
/*               ==                                                  */
/*                                                                   */
/* ***************************************************************** */

/* ***************************************************************** */
/*                                                                   */
/* CLASS:        TDrawDocument                                       */
/*                                                                   */
/* PARENT CLASS: TFileDocument                                       */
/*                                                                   */
/* METHODS:                                                          */
/*               Open(int mode, const char far* path=0);             */
/*               Close();                                            */
/*               IsOpen()                                            */
/*               Commit(BOOL force = FALSE);                         */
/*               Revert(BOOL clear = FALSE);                         */
/*               FindProperty(const char far* name);                 */
/*               PropertyFlags(int index);                           */
/*               PropertyName(int index);                            */
/*               PropertyCount()                                     */
/*               GetProperty(int index, void far* dest, int textlen) */
```

```
/*                GetLine(unsigned int index);                      */
/*                AddLine(TLine& line);                             */
/*                DeleteLine(unsigned int index);                   */
/*                ModifyLine(TLine& line, unsigned int index);      */
/*                Undo();                                           */
/*                                                                  */
/* **************************************************************** */

/* **************************************************************** */
/*                                                                  */
/* CLASS:         TDrawView                                         */
/*                                                                  */
/* PARENT CLASS: TWindowView                                        */
/*                                                                  */
/* METHODS:                                                         */
/*                EvLButtonDown(UINT, TPoint&);                     */
/*                EvRButtonDown(UINT, TPoint&);                     */
/*                EvMouseMove(UINT, TPoint&);                       */
/*                EvLButtonUp(UINT, TPoint&);                       */
/*                Paint(TDC&, BOOL, TRect&);                        */
/*                CmUndo();                                         */
/*                VnCommit(BOOL force);                             */
/*                VnRevert(BOOL clear);                             */
/*                VnAppend(unsigned int index);                     */
/*                VnDelete(unsigned int index);                     */
/*                VnModify(unsigned int index);                     */
/*                UpdateGauges(UINT Percentage);                    */
/*                SetupWindow();                                    */
/*                EvTimer(UINT id);                                 */
/*                CmControlsstop ();                                */
/*                CmControlsplay ();                                */
/*                CmControlspause ();                               */
/*                CmControlsrecord ();                              */
/*                CmActioncompress ();                              */
/*                TemporaryEnable (TCommandEnabler &tce);           */
/*                MciNotify(WPARAM, LPARAM);                        */
/*                                                                  */
/* **************************************************************** */

#include <owl\owlpch.h>
#include <owl\dc.h>
#include <owl\inputdia.h>
#include <owl\chooseco.h>
#include <owl\gdiobjec.h>
#include <owl\docmanag.h>
#include <owl\filedoc.h>
#include <owl\listbox.h>
#include <classlib\arrays.h>
#include "spechlb2.rc"
#include <owl\statusba.h>
#include <owl\decframe.h>
#include <owl\scroller.h>
#include <owl\gauge.h>
#include <owl\dialog.h>
#pragma hdrstop
```

```
#include <owl\applicat.h>
#include <owl\opensave.h>
#include <owl\framewin.h>
#include <owl\menu.h>
#include <owl\button.h>
#include <owl\slider.h>
#include <owl\slider.rh>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <mmsystem.h>
#include "typedef.h"
#include "celp1.h"
#include <stdio.h>

const WORD ID_PROGRESSWIN = 400;


extern "C" { void far EraseBuffers(void); }
extern "C" { void far InitializeAllParameters(void);}
extern "C" { void far Analyze(i_16 *pcm, u_16 *stream);}
extern "C" { void far Synthesize(u_16 *stream, i_16 *pcm);}


typedef TArray<TPoint> TPoints;
typedef TArrayIterator<TPoint> TPointsIterator;
typedef  struct TWaveFileHdr {
          unsigned char   RIFF;
          unsigned long   RIFFLength;
          unsigned char   WAVE;
          unsigned char   fmt;
          unsigned long   fmtLength;
          unsigned int    formatTag;          //Format category
          unsigned int    nChannels;          //Number of channels
          unsigned long   sampleRate;         //samples per second
          unsigned long   bytesPerSec;    //average bytes per second
          unsigned int align;                 //nBAlign
          unsigned int bitsPerSample;
          unsigned long   data;
          long   numOfSamples;
            };

#define ID_SCROLL     150         // Scroll bar
#define TIMER_ID      264         // Unique timer ID.
#define MCI_PARM2(p)  ((long)(void far*)&p)

UINT  DeviceId = 0;    // The global waveform device opened handle
BOOL  FlushNotify = FALSE;


/* ******************************************************************* */
/*                                TMyApp                               */
/* ******************************************************************* */
#include <owl\decmdifr.h>
#include <owl\controlb.h>
```

```
#include <owl\buttonga.h>
#include <string.h>
#include "spechlbl.rc"

/* ******************************************************************* */
/*                                                                     */
/* CLASS:            TMyApp                                            */
/*                                                                     */
/* PARENT CLASS:     TApplication                                      */
/*                                                                     */
/* OVERIDE METHODS:                                                    */
/*                   InitInstance(..)                                  */
/*                   InitMainWindow(..)                                */
/*                         // Event handlers                           */
/*                   EvNewView(..)                                     */
/*                   EvCloseView(..)                                   */
/*                   EvDropFiles(..)                                   */
/*                   CmAbout(..)                                       */
/*                                                                     */
/* ******************************************************************* */

 class TMyApp : public TApplication {
  public:
    TMyApp() : TApplication() {}

  protected:
    // Override methods of TApplication
    void InitInstance();
    void InitMainWindow();

    // Event handlers
    void EvNewView  (TView& view);
   void EvCloseView(TView& view);
   void EvDropFiles(TDropInfo dropInfo);
    void CmAbout();

    TMDIClient* Client;
  DECLARE_RESPONSE_TABLE(TMyApp);
};

DEFINE_RESPONSE_TABLE1(TMyApp, TApplication)
  EV_OWLVIEW(dnCreate, EvNewView),
  EV_OWLVIEW(dnClose,  EvCloseView),
  EV_WM_DROPFILES,
  EV_COMMAND(CM_ABOUT, CmAbout),
END_RESPONSE_TABLE;

void TMyApp::InitMainWindow()
{
  // Construct the decorated frame window
  TDecoratedMDIFrame* frame = new TDecoratedMDIFrame(
                                   "Speech Compression Lab", 0,
                                   *(Client = new TMDIClient), TRUE);
  // Construct a status bar
  TStatusBar* sb = new TStatusBar(frame, TGadget::Recessed);
```

```
  // Construct a control bar
  TControlBar *cb = new TControlBar(frame);
  cb->Insert(*new TButtonGadget(CM_FILENEW, CM_FILENEW,
                              TButtonGadget::Command));
  cb->Insert(*new TButtonGadget(CM_FILEOPEN, CM_FILEOPEN,
                              TButtonGadget::Command));
  cb->Insert(*new TButtonGadget(CM_FILESAVE, CM_FILESAVE,
                              TButtonGadget::Command));
  cb->Insert(*new TButtonGadget(CM_FILESAVEAS, CM_FILESAVEAS,
                              TButtonGadget::Command));
  cb->Insert(*new TSeparatorGadget);
  cb->Insert(*new TSeparatorGadget);
  cb->Insert(*new TButtonGadget(CM_ACTIONCOMPRESS, CM_ACTIONCOMPRESS,
                              TButtonGadget::Command));
  cb->Insert(*new TSeparatorGadget);
  cb->Insert(*new TSeparatorGadget);
  cb->Insert(*new TButtonGadget(CM_CONTROLSPLAY, CM_CONTROLSPLAY,
                              TButtonGadget::Command));
  cb->Insert(*new TButtonGadget(CM_CONTROLSPAUSE, CM_CONTROLSPAUSE,
                              TButtonGadget::Command));
  cb->Insert(*new TButtonGadget(CM_CONTROLSSTOP, CM_CONTROLSSTOP,
                              TButtonGadget::Command));
  cb->Insert(*new TButtonGadget(CM_CONTROLSRECORD, CM_CONTROLSRECORD,
                              TButtonGadget::Command));
  cb->Insert(*new TSeparatorGadget);
  cb->Insert(*new TSeparatorGadget);
  cb->Insert(*new TButtonGadget(CM_ABOUT, CM_ABOUT,
                              TButtonGadget::Command));
  cb->SetHintMode(TGadgetWindow::EnterHints);

  // Insert the status bar and control bar into the frame
  frame->Insert(*sb, TDecoratedFrame::Bottom);
  frame->Insert(*cb, TDecoratedFrame::Top);

  // Set the main window and its menu
  SetMainWindow(frame);
  GetMainWindow()->SetMenuDescr(TMenuDescr("COMMANDS",1,1,0,0,1,1));

  // Install the document manager
  SetDocManager(new TDocManager(dmMDI | dmMenu));
}

void
TMyApp::InitInstance()
{
  TApplication::InitInstance();
  GetMainWindow()->DragAcceptFiles(TRUE);
}

void
TMyApp::EvDropFiles(TDropInfo dropInfo)
{
  int fileCount = dropInfo.DragQueryFileCount();
  for (int index = 0; index < fileCount; index++) {
    int fileLength = dropInfo.DragQueryFileNameLen(index)+1;
```

```
    char* filePath = new char [fileLength];
    dropInfo.DragQueryFile(index, filePath, fileLength);
    TDocTemplate* tpl = GetDocManager()->MatchTemplate(filePath);
    if (tpl)
     tpl->CreateDoc(filePath);
    delete filePath;
  }
  dropInfo.DragFinish();
}

void
TMyApp::EvNewView(TView& view)
{
  TMDIChild* child = new TMDIChild(*Client, 0, view.GetWindow());
  if (view.GetViewMenu())
    child->SetMenuDescr(*view.GetViewMenu());
  child->Create();
}

void
TMyApp::EvCloseView(TView& /*view*/)
{  // nothing needs to be done here for MDI
}

void TMyApp::CmAbout()
{
  TDialog(GetMainWindow(), IDD_ABOUT).Execute();
}

int OwlMain(int /*argc*/, char* /*argv*/ [])
{
  return TMyApp().Run();
}
```

```
/* ****************************************************************** */
/*                              TSoundBar                             */
/* ****************************************************************** */



/* ****************************************************************** */
/*                                                                    */
/* CLASS:        TSoundBar                                            */
/*                                                                    */
/* PARENT CLASS: THSlider                                             */
/*                                                                    */
/* METHODS:                                                           */
/*               SetInfo(..)                                          */
/*               SetName(..)                                          */
/*               SBLineUp(..)                                         */
/*               SBLineDown(..)                                       */
/*               SBPageUp(..)                                         */
/*               SBPageDown(..)                                       */
/*               SBThumbPosition(..)                                  */
/*               SBTop(..)                                            */
/*               SBBottom(..)                                         */
/*               ReposAndPlay(..)                                     */
/*                                                                    */
/* ****************************************************************** */

class TSoundBar : public THSlider {
  public:
    TSoundBar(TWindow* parent, int id, int x, int y, int w, int h,
           TModule* module = 0)
          : THSlider(parent, id, x, y, w, h, IDB_HSLIDERTHUMB, module) {}

    void    SetInfo(int ratio, long length);
    void    SetName(char* name);

    //
    // Override TScrollBars virtual functions
    //
    void    SBLineUp();
   void    SBLineDown();
    void    SBPageUp();
    void    SBPageDown();
    void    SBThumbPosition(int thumbPos);
    void    SBTop();
   void    SBBottom();

  private:
    int     WaveRatio;
    long    WaveLength;
   char    ElementName[255];

    void    ReposAndPlay(long newPos);
};

void
```

```
TSoundBar::ReposAndPlay(long newPos)
{
  MCI_PLAY_PARMS    MciPlayParm;
  MCI_SEEK_PARMS    MciSeekParm;
  MCI_SET_PARMS     MciSetParm;
  MCI_OPEN_PARMS    MciOpenParm;
  MCI_GENERIC_PARMS MciGenParm;

  // Only allow SEEK if playing.
  //
  if (!DeviceId)
     return;

  // Close the currently playing wave.
  //
  FlushNotify = TRUE;
  MciGenParm.dwCallback = 0;
  mciSendCommand(DeviceId, MCI_STOP, MCI_WAIT, MCI_PARM2(MciGenParm));
  mciSendCommand(DeviceId, MCI_CLOSE, MCI_WAIT, MCI_PARM2(MciGenParm));

  // Open the wave again and seek to new position.
  //
  MciOpenParm.dwCallback = 0;
  MciOpenParm.wDeviceID = DeviceId;
  #if !defined(__WIN32__)
    MciOpenParm.wReserved0 = 0;
  #endif
  MciOpenParm.lpstrDeviceType = 0;
  MciOpenParm.lpstrElementName = ElementName;
  MciOpenParm.lpstrAlias = 0;

  if (mciSendCommand(DeviceId, MCI_OPEN, MCI_WAIT| MCI_OPEN_ELEMENT,
      MCI_PARM2(MciOpenParm)))
     {
       MessageBox("Open Error", "Sound Play", MB_OK);
      return;
     }
  DeviceId = MciOpenParm.wDeviceID;

  // Our time scale is in SAMPLES.
  //
  MciSetParm.dwTimeFormat = MCI_FORMAT_SAMPLES;
  if (mciSendCommand(DeviceId, MCI_SET, MCI_SET_TIME_FORMAT,
      MCI_PARM2(MciSetParm)))
  {
    MessageBox("Set Time Error", "Sound Play", MB_OK);
     return;
  }

  // Compute new position, remember the scrollbar range has been
  // scaled based on WaveRatio.
  //
  MciSeekParm.dwCallback = 0;
  MciSeekParm.dwTo = (newPos*WaveRatio > WaveLength) ? WaveLength :
                                                        newPos*WaveRatio;
```

```
  if (mciSendCommand(DeviceId, MCI_SEEK, MCI_TO, MCI_PARM2(MciSeekParm)))
  {
    MessageBox("Seek Error", "Sound Play", MB_OK);
   return;
  }

  MciPlayParm.dwCallback = (long)HWindow;
  MciPlayParm.dwFrom = 0;
  MciPlayParm.dwTo = 0;
  if (mciSendCommand(DeviceId, MCI_PLAY, MCI_NOTIFY,
      MCI_PARM2(MciPlayParm)))
  {
    MessageBox("Play Error", "Sound Play", MB_OK);
    return;
  }
}

void
TSoundBar::SetInfo(int ratio, long length)
{
  WaveRatio = ratio;
  WaveLength = length;
}

void
TSoundBar::SetName(char* name)
{
  strcpy(ElementName, name);
}

void
TSoundBar::SBLineUp()
{
  THSlider::SBLineUp();
  ReposAndPlay(GetPosition());
}

void
TSoundBar::SBLineDown()
{
  THSlider::SBLineDown();
  ReposAndPlay(GetPosition());
}

void
TSoundBar::SBPageUp()
{
  THSlider::SBPageUp();
  ReposAndPlay(GetPosition());
}

void
TSoundBar::SBPageDown()
{
  THSlider::SBPageDown();
```

```
  ReposAndPlay(GetPosition());
}

void
TSoundBar::SBThumbPosition(int thumbPos)
{
  THSlider::SBThumbPosition(thumbPos);
  ReposAndPlay(GetPosition());
}

void
TSoundBar::SBTop()
{
  THSlider::SBTop();
  ReposAndPlay(GetPosition());
}

void
TSoundBar::SBBottom()
{
  THSlider::SBBottom();
  ReposAndPlay(GetPosition());
}
```

```
/* ******************************************************************** */
/*                              TLine                                   */
/* ******************************************************************** */


/* ******************************************************************** */
/*                                                                      */
/* CLASS:         TLine                                                 */
/*                                                                      */
/* PARENT CLASS: TPoints                                                */
/*                                                                      */
/* METHODS:                                                             */
/*                QueryPenSize(..)                                      */
/*                QueryColor(..)                                        */
/*                SetPen(..)                                            */
/*                SetPen(..)                                            */
/*                GetPenSize(..)                                        */
/*                GetPenColor(..)                                       */
/*                Draw(..)                                              */
/*                ==                                                    */
/*                                                                      */
/* ******************************************************************** */
```

```
class TLine : public TPoints {
  public:
    // Constructor to allow construction from a color and a pen size.
    // Also serves as default constructor.
   TLine(const TColor &color = 2 /*TColor(0)*/, int penSize = 1)
    : TPoints(10, 0, 10), PenSize(penSize), Color(color) {}

    typedef struct tagPOINT {
           long x;
           long y;
       } POINT;
    // Functions to modify and query pen attributes.
    int QueryPenSize() const { return PenSize; }
    const TColor& QueryColor() const { return Color; }
    void SetPen(TColor &newColor, int penSize = 0);
    void SetPen(int penSize);
    BOOL GetPenSize();
    BOOL GetPenColor();

    // TLine draws itself.  Returns TRUE if everything went OK.
    virtual BOOL Draw(TDC &) const;

    // The == operator must be defined for the container class,
    // even if unused
    BOOL operator ==(const TLine& other) const { return &other == this;}
   TWaveFileHdr waveHdr;


  protected:
    int PenSize;
   TColor Color;
};

typedef TArray<TLine> TLines;
typedef TArrayIterator<TLine> TLinesIterator;

const int vnDrawAppend = vnCustomBase+0;
const int vnDrawDelete = vnCustomBase+1;
const int vnDrawModify = vnCustomBase+2;

NOTIFY_SIG(vnDrawAppend, unsigned int)
NOTIFY_SIG(vnDrawDelete, unsigned int)
NOTIFY_SIG(vnDrawModify, unsigned int)

#define EV_VN_DRAWAPPEND  VN_DEFINE(vnDrawAppend,  VnAppend,  int)
#define EV_VN_DRAWDELETE  VN_DEFINE(vnDrawDelete,  VnDelete,  int)
#define EV_VN_DRAWMODIFY  VN_DEFINE(vnDrawModify,  VnModify,  int)

DEFINE_DOC_TEMPLATE_CLASS(TDrawDocument, TDrawView,   DrawTemplate);
DrawTemplate drawTpl("Microsoft Wave (*.wav)","*.wav",0,"wav",
                     dtAutoDelete|dtUpdateDir);

void TLine::SetPen(int penSize)
{
  if (penSize < 1)
```

```
    PenSize = 1;
  else
    PenSize = penSize;
}

void TLine::SetPen(TColor &newColor, int penSize)
{
  // If penSize isn't the default (0), set PenSize to the new size.
  if (penSize)
   PenSize = penSize;

  Color = newColor;
}

BOOL TLine::Draw(TDC &dc) const
{
  // Set pen for the dc to the values for this line
  TPen pen(Color, PenSize);
  dc.SelectObject(pen);

  // Iterates through the points in the line i.
  TPointsIterator j(*this);
  BOOL first = TRUE;

  while (j) {
   TPoint p = j++;

    if (!first)
     dc.LineTo(p);
     //dc.SetPixel(p,Color);
   else {
     dc.MoveTo(p);
       first = FALSE;
     }
  }
  dc.RestorePen();
  return TRUE;
}
```

```
/* ******************************************************************** */
/*                          TDrawDocument                               */
/* ******************************************************************** */



/* ******************************************************************** */
/*                                                                      */
/* CLASS:        TDrawDocument                                          */
/*                                                                      */
/* PARENT CLASS: TFileDocument                                          */
/*                                                                      */
/* METHODS:                                                             */
/*               Open(int mode, const char far* path=0);                */
/*               Close();                                               */
/*               IsOpen()                                               */
/*               Commit(BOOL force = FALSE);                            */
/*               Revert(BOOL clear = FALSE);                            */
/*               FindProperty(const char far* name);                    */
/*               PropertyFlags(int index);                              */
/*               PropertyName(int index);                               */
/*               PropertyCount()                                        */
/*               GetProperty(int index, void far* dest, int textlen)    */
/*               GetLine(unsigned int index);                           */
/*               AddLine(TLine& line);                                  */
/*               DeleteLine(unsigned int index);                        */
/*               ModifyLine(TLine& line, unsigned int index);           */
/*               Undo();                                                */
/*                                                                      */
/* ******************************************************************** */

class _DOCVIEWCLASS TDrawDocument : public TFileDocument
{
  public:
   enum {
    PrevProperty = TFileDocument::NextProperty-1,
    LineCount,
    Description,
    NextProperty,
   };

   enum {
    UndoNone,
    UndoDelete,
    UndoAppend,
    UndoModify
   };

   TDrawDocument(TDocument* parent = 0)
       : TFileDocument(parent), Lines(0), UndoLine(0),
                     UndoState(UndoNone) {}
  ~TDrawDocument() { delete Lines; delete UndoLine; }

   // implement virtual methods of TDocument
   BOOL   Open(int mode, const char far* path=0);
```

```
    BOOL   Close();
    BOOL   IsOpen() { return Lines != 0; }
    BOOL   Commit(BOOL force = FALSE);
    BOOL   Revert(BOOL clear = FALSE);

    int         FindProperty(const char far* name);  // return index
    int         PropertyFlags(int index);
    const char* PropertyName(int index);
     int         PropertyCount() {return NextProperty - 1;}
     int         GetProperty(int index, void far* dest, int textlen=0);

     // data access functions
     const TLine* GetLine(unsigned int index);
     int    AddLine(TLine& line);
     void   DeleteLine(unsigned int index);
     void   ModifyLine(TLine& line, unsigned int index);
     void   Undo();

   protected:
     TLines* Lines;
     TLine* UndoLine;
     int    UndoState;
     int    UndoIndex;
     string FileInfo;
};

static char* PropNames[] = {
  "Line Count",         // LineCount
  "Description",        // Description
};

static int PropFlags[] = {
  pfGetBinary|pfGetText, // LineCount
  pfGetText,             // Description
};

const char*
TDrawDocument::PropertyName(int index)
{
  if (index <= PrevProperty)
   return TFileDocument::PropertyName(index);
  else if (index < NextProperty)
    return PropNames[index-PrevProperty-1];
  else
    return 0;
}

int
TDrawDocument::PropertyFlags(int index)
{
  if (index <= PrevProperty)
    return TFileDocument::PropertyFlags(index);
  else if (index < NextProperty)
    return PropFlags[index-PrevProperty-1];
  else
```

```
    return 0;
}

int
TDrawDocument::FindProperty(const char far* name)
{
  for (int i=0; i < NextProperty-PrevProperty-1; i++)
    if (strcmp(PropNames[i], name) == 0)
      return i+PrevProperty+1;
  return 0;
}

int
TDrawDocument::GetProperty(int prop, void far* dest, int textlen)
{
  switch(prop)
  {
    case LineCount:
    {
      int count = Lines->GetItemsInContainer();
     if (!textlen) {
        *(int far*)dest = count;
        return sizeof(int);
      }
      return wsprintf((char far*)dest, "%d", count);
    }
    case Description:
      char* temp = new char[textlen]; //need local copy for medium model
      int len = FileInfo.copy(temp, textlen);
      strcpy((char far*)dest, temp);
      return len;
  }
  return TFileDocument::GetProperty(prop, dest, textlen);
}


BOOL TDrawDocument::Commit(BOOL force)
{
  if (!IsDirty() && !force)
    return TRUE;

//******* First check if file open was successful
  TOutStream* os = OutStream(ofWrite);
  if (!os)
   return FALSE;

//****** Put code to save here
  SetDirty(FALSE);
  return TRUE;
}

BOOL TDrawDocument::Revert(BOOL clear)
{
  if (!TFileDocument::Revert(clear))
```

```
    return FALSE;
  if (!clear)
    Open(0);
  return TRUE;
}

BOOL TDrawDocument::Open(int /*mode*/, const char far* path)
{
  char fileinfo[100];
  Lines = new TLines(5, 0, 5);
  if (path)
   SetDocPath(path);
  if (GetDocPath()) {
   ifstream is(GetTitle());
   if (!is)
    return FALSE;

   TLine line;
   unsigned char sampleByte;
   BOOL fileFormatError = 0;

   char dataRead[4];
   is.read(dataRead,4);
   if (strncmp("RIFF", dataRead, 4))
    fileFormatError = 1;
   is.read(dataRead,4);       //Dummy read length of chunk
   is.read(dataRead,4);
   if (strncmp("WAVE", dataRead, 4))
    fileFormatError = 1;
   is.read(dataRead,4);
   if (strncmp("fmt ", dataRead, 4))
    fileFormatError = 1;
   is.read(dataRead,4);       //Dummy read length of fmt chunk

              //initialise wave
   line.waveHdr.formatTag=line.waveHdr.nChannels=
       line.waveHdr.bitsPerSample=0;
   line.waveHdr.numOfSamples=line.waveHdr.sampleRate=0;
   int t;
   for (t=1; t>=0; t--)
    {
    is.get(sampleByte);
    line.waveHdr.formatTag += (sampleByte << t);
    }
   for (t=1; t>=0; t--)
    {
    is.get(sampleByte);
    line.waveHdr.nChannels += (sampleByte << t);
    }
   for (t=3; t>=0; t--)
    {
    is.get(sampleByte);
    line.waveHdr.sampleRate += (sampleByte << t);
    }
  is.read(dataRead,4);     //Dummy read bytes per second
```

```
    is.get(sampleByte); //Dummy read align
    is.get(sampleByte);
    for (t=1; t>=0; t--)
      {
      is.get(sampleByte);
      line.waveHdr.bitsPerSample += (sampleByte << t);
      }
    is.read(dataRead,4);
     if (strncmp("data", dataRead, 4))
      fileFormatError = 1;
     for (t=3; t>=0; t--)
      {
      is.get(sampleByte);
      line.waveHdr.numOfSamples += (sampleByte << t);
      }

    long i=(-1);
    while (line.waveHdr.numOfSamples > i++)
      {
      TPoint point;
      is.get(sampleByte);
      point.x = i;
      point.y = sampleByte;
      line.Add(point);
      }
     Lines->Add(line);

     FileInfo = fileinfo;
    }
  SetDirty(FALSE);
  UndoState = UndoNone;
  return TRUE;
}

BOOL TDrawDocument::Close()
{
  delete Lines;
  Lines = 0;
  return TRUE;
}

const TLine* TDrawDocument::GetLine(unsigned int index)
{
  if (!IsOpen() && !Open(ofRead | ofWrite))
   return 0;
  return index < Lines->GetItemsInContainer() ? &(*Lines)[index] : 0;
}

int TDrawDocument::AddLine(TLine& line)
{
  int index = Lines->GetItemsInContainer();
  Lines->Add(line);
  SetDirty(TRUE);
  NotifyViews(vnDrawAppend, index);
  UndoState = UndoAppend;
```

```
  return index;
}

void TDrawDocument::DeleteLine(unsigned int index)
{
  const TLine* oldLine = GetLine(index);
  if (!oldLine)
    return;
  delete UndoLine;
  UndoLine = new TLine(*oldLine);
  Lines->Detach(index);
  SetDirty(TRUE);
  NotifyViews(vnDrawDelete, index);
  UndoState = UndoDelete;
}

void TDrawDocument::ModifyLine(TLine& line, unsigned int index)
{
  delete UndoLine;
  UndoLine = new TLine((*Lines)[index]);
  SetDirty(TRUE);
  (*Lines)[index] = line;
  NotifyViews(vnDrawModify, index);
  UndoState = UndoModify;
  UndoIndex = index;
}


void TDrawDocument::Undo()
{
  switch (UndoState) {
   case UndoAppend:
    DeleteLine(Lines->GetItemsInContainer()-1);
    return;
   case UndoDelete:
    AddLine(*UndoLine);
      delete UndoLine;
    UndoLine = 0;
      return;
   case UndoModify:
    TLine* temp = UndoLine;
    UndoLine = 0;
      ModifyLine(*temp, UndoIndex);
    delete temp;
  }
}
```

```
/* ****************************************************************** */
/*                          TDrawView                                 */
/* ****************************************************************** */


/* ****************************************************************** */
/*                                                                    */
/* CLASS:        TDrawView                                            */
/*                                                                    */
/* PARENT CLASS: TWindowView                                          */
/*                                                                    */
/* METHODS:                                                           */
/*               EvLButtonDown(UINT, TPoint&);                        */
/*               EvRButtonDown(UINT, TPoint&);                        */
/*               EvMouseMove(UINT, TPoint&);                          */
/*               EvLButtonUp(UINT, TPoint&);                          */
/*               Paint(TDC&, BOOL, TRect&);                           */
/*               CmUndo();                                            */
/*               VnCommit(BOOL force);                                */
/*               VnRevert(BOOL clear);                                */
/*               VnAppend(unsigned int index);                        */
/*               VnDelete(unsigned int index);                        */
/*               VnModify(unsigned int index);                        */
/*               UpdateGauges(UINT Percentage);                       */
/*               SetupWindow();                                       */
/*               EvTimer(UINT id);                                    */
/*               CmControlsstop ();                                   */
/*               CmControlsplay ();                                   */
/*               CmControlspause ();                                  */
/*               CmControlsrecord ();                                 */
/*               CmActioncompress ();                                 */
/*               TemporaryEnable (TCommandEnabler &tce);              */
/*               MciNotify(WPARAM, LPARAM);                           */
/*                                                                    */
/* ****************************************************************** */
class _DOCVIEWCLASS TDrawView : public TWindowView
{
  public:
    TDrawView(TDrawDocument& doc, TWindow *parent = 0);
  ~TDrawView() {delete DragDC; delete Line; StopMCI();}
   static const char far* StaticName(){return "Speech Compression Lab";}
   const char far* GetViewName(){return StaticName();}


  protected: // same as Doc member, but cast to derived class
   TDrawDocument* DrawDoc;
   TDC *DragDC;
   TPen *Pen;
   TLine *Line; // To hold a single line sent or received from document
   TGauge *progressWin;

   // Message response functions
   void EvLButtonDown(UINT, TPoint&);
```

```
    void EvRButtonDown(UINT, TPoint&);
    void EvMouseMove(UINT, TPoint&);
    void EvLButtonUp(UINT, TPoint&);
    void Paint(TDC&, BOOL, TRect&);
    void CmUndo();

    // Document notifications
    BOOL VnCommit(BOOL force);
    BOOL VnRevert(BOOL clear);
    BOOL VnAppend(unsigned int index);
    BOOL VnDelete(unsigned int index);
    BOOL VnModify(unsigned int index);

    void UpdateGauges(UINT Percentage);
    virtual void SetupWindow();


//Sound support functions and variables
   void EvTimer(UINT id);
   void CmControlsstop ();
   void CmControlsplay ();
   void CmControlspause ();
   void CmControlsrecord ();
   void CmActioncompress ();
   void TemporaryEnable (TCommandEnabler &tce);
   LRESULT MciNotify(WPARAM, LPARAM);



    char         ElementName[255];
    int          Running;
    int          Pause;
    UINT         TimeGoing;
    int          WaveRatio;
    long         WaveLength;
    TSoundBar*   SoundBar;

   MCI_GENERIC_PARMS MciGenParm;
   MCI_OPEN_PARMS    MciOpenParm;
   MCI_PLAY_PARMS    MciPlayParm;
   MCI_STATUS_PARMS  MciStatusParm;
   MCI_SET_PARMS     MciSetParm;

    void  GetDeviceInfo();
    void  StopWave();
    void  StopMCI();

    DECLARE_RESPONSE_TABLE(TDrawView);
};


DEFINE_RESPONSE_TABLE1(TDrawView, TWindowView)
  EV_WM_LBUTTONDOWN,
  EV_WM_RBUTTONDOWN,
  EV_WM_MOUSEMOVE,
```

```
  EV_WM_LBUTTONUP,
  EV_COMMAND(CM_UNDO, CmUndo),
  EV_COMMAND(CM_CONTROLSPAUSE, CmControlspause),
  EV_COMMAND(CM_CONTROLSRECORD, CmControlsrecord),
  EV_COMMAND(CM_ACTIONCOMPRESS, CmActioncompress),
  EV_COMMAND(CM_CONTROLSSTOP, CmControlsstop),
  EV_COMMAND(CM_CONTROLSPLAY, CmControlsplay),
  EV_MESSAGE(MM_MCINOTIFY, MciNotify),
  EV_WM_TIMER,
  EV_VN_COMMIT,
  EV_VN_REVERT,
  EV_VN_DRAWAPPEND,
  EV_VN_DRAWDELETE,
  EV_VN_DRAWMODIFY,
END_RESPONSE_TABLE;

TDrawView::TDrawView(TDrawDocument& doc,TWindow *parent)
       : TWindowView(doc,parent), DrawDoc(&doc)
{
  DragDC = 0;
  Line = new TLine(TColor::Black, 1);
  SetViewMenu(new TMenuDescr(IDM_DRAWVIEW,0,1,1,3,0,0));
  progressWin = new TGauge(this, "%d%%",
                ID_PROGRESSWIN, 0, 0, 240, 30, TRUE, 2);
  SetBkgndColor(0);
  Attr.Style |= WS_HSCROLL;
  Scroller = new TScroller(this, 1, 1, 100, 100);
//*********SetRange(Lastadatapoint, dummy): needed to observe full wave


// Sound stuff
  Running = 0;
  Pause = 0;
  WaveLength = WaveRatio = 0;
  ElementName[0] = 0;

  SoundBar = new TSoundBar(GetWindowPtr(GetParent()),
                           ID_SCROLL, 50, 67, 300, 40);

     WaveLength = 0;
     WaveRatio = 0;
     SoundBar->SetPosition(0);
}

void TDrawView::EvLButtonDown(UINT, TPoint& point)
{
  if (!DragDC) {
   SetCapture();
    DragDC = new TClientDC(*this);
    Pen = new TPen(Line->QueryColor(), Line->QueryPenSize());
    DragDC->SelectObject(*Pen);
    DragDC->MoveTo(point);
    Line->Add(point);
  }
}
```

```
void TDrawView::EvRButtonDown(UINT, TPoint&)
{
}

void TDrawView::EvMouseMove(UINT, TPoint& point)
{
}

void TDrawView::EvLButtonUp(UINT, TPoint&)
{
}


void TDrawView::CmUndo()
{
  DrawDoc->Undo();
}

void TDrawView::Paint(TDC& dc, BOOL, TRect&)
{
  // Iterates through the array of line objects.
  int i = 0;
  const TLine* line;
  while ((line = DrawDoc->GetLine(i++)) != 0)
   line->Draw(dc);
}

BOOL TDrawView::VnCommit(BOOL /*force*/)
{
  // nothing to do here, no data held in view
  return TRUE;
}

BOOL TDrawView::VnRevert(BOOL /*clear*/)
{
  Invalidate();  // force full repaint
  return TRUE;
}

BOOL TDrawView::VnAppend(unsigned int index)
{
  TClientDC dc(*this);
  const TLine* line = DrawDoc->GetLine(index);
  line->Draw(dc);
  return TRUE;
}

BOOL TDrawView::VnModify(unsigned int /*index*/)
{
  Invalidate();  // force full repaint
  return TRUE;
}

BOOL TDrawView::VnDelete(unsigned int /*index*/)
{
```

```
  Invalidate();  // force full repaint
  return TRUE;
}

void TDrawView::CmControlspause()
{
    if (!Pause) {
     // Pause the playing.
     //
     MciGenParm.dwCallback = 0;
     mciSendCommand(DeviceId, MCI_PAUSE, MCI_WAIT,
                     MCI_PARM2(MciGenParm));

     Pause = TRUE;
   }
}

void TDrawView::CmControlsrecord()
{
}

void TDrawView::CmActioncompress()
{
  u_16   i;
  u_32   Frame=0;
  time_t startTime;
  time_t compileTime;
  time_t compressTime;
  float sampleTimeLength;   //****************Needs changing
  time_t timeLength;
  //Needs changing
  LPSTR CLPFileName = "output.cpl";
  FILE *WavFile;
  FILE *CLPFile;


  InitializeAllParameters();
  EraseBuffers();
  WavFile = fopen(DrawDoc->GetTitle(),"rb");
  CLPFile = fopen(CLPFileName, "wb");
   if ((WavFile == 0)&&(CLPFile==0)) {
    MessageBox ("File Error.", "Speech Compression Lab", MB_OK);
    return ;
  } else {

  unsigned char sampleByte[1];
  char dataRead[4];
  BOOL fileFormatError = 0;


   fread(dataRead, sizeof(char), 4, WavFile);
   if (strncmp("RIFF", dataRead, 4))
    fileFormatError = 1;
    fwrite(dataRead, sizeof(char), 4, CLPFile);
                               //Dummy read length of chunk
```

```
 fread(dataRead, sizeof(char), 4, WavFile);
  fwrite(dataRead, sizeof(char), 4, CLPFile);
 fread(dataRead, sizeof(char), 4, WavFile);
 if (strncmp("WAVE", dataRead, 4))
  fileFormatError = 1;
  fwrite(dataRead, sizeof(char), 4, CLPFile);
 fread(dataRead, sizeof(char), 4, WavFile);
 if (strncmp("fmt ", dataRead, 4))
  fileFormatError = 1;
  fwrite(dataRead, sizeof(char), 4, CLPFile);
                         //Dummy read length of fmt chunk
 fread(dataRead, sizeof(char), 4, WavFile);
   fwrite(dataRead, sizeof(char), 4, CLPFile);
                         //Read formatTag and number of channels
 fread(dataRead, sizeof(char), 4, WavFile);
   fwrite(dataRead, sizeof(char), 4, CLPFile);
 fread(dataRead, sizeof(char), 4, WavFile);  //samplerate
   fwrite(dataRead, sizeof(char), 4, CLPFile);
fread(dataRead, sizeof(char), 4, WavFile);   //bytes per second
  fwrite(dataRead, sizeof(char), 4, CLPFile);
 fread(dataRead, sizeof(char), 4, WavFile);  //align and bits/sample
   fwrite(dataRead, sizeof(char), 4, CLPFile);
fread(dataRead, sizeof(char), 4, WavFile);
 if (strncmp("data", dataRead, 4))
  fileFormatError = 1;
  fwrite(dataRead, sizeof(char), 4, CLPFile);

 unsigned long numOfSamples = 0;
 unsigned long temp;
 for (i=0; i<4; i++)
  {
  fread(sampleByte, 1,1,WavFile);
  fwrite(sampleByte, 1,1,CLPFile);
  temp = sampleByte[0];
  temp = temp << (i*8);
  numOfSamples += temp;
  }
 if (fileFormatError) {
  MessageBox("File Format Error.","Speech Compression Lab", MB_OK);
  return ;
  }

  sampleTimeLength = 0.03;
  timeLength = sampleTimeLength*numOfSamples;

  unsigned long numOfFrames;
  numOfFrames = numOfSamples / PCMFRAMESIZE;
  if (numOfSamples % PCMFRAMESIZE)
    numOfFrames++;


  UINT PercentageDone;
  progressWin->ShowWindow(SW_SHOW);
  progressWin->ShowWindow(SW_RESTORE);
  compressTime = 0;
```

```
    while  (fread(PCMData8,sizeof(i_8),PCMFRAMESIZE,WavFile)
             ==PCMFRAMESIZE)
    {
      PCMData[i]=PCMData8[i];
      startTime = time(NULL);
      Analyze(PCMData, DataStream);
      compressTime = compressTime + (time(NULL)-startTime);
      fwrite(DataStream,sizeof(u_16),CELFRAMESIZE,CLPFile);
      PercentageDone = (100*Frame)/numOfFrames;
      UpdateGauges(PercentageDone);
      if (numOfFrames > Frame)
        GetApplication()->PumpWaitingMessages();
      Frame++;
    }
  int *dec;
  int *sign;
  double ratio=(100*compressTime/timeLength);
  char *myMsg = ecvt(ratio,5,dec,sign);
  MessageBox(myMsg,"Percentage of realtime",MB_OK);
  UpdateGauges(0);
  progressWin->Show(SW_HIDE);
  fclose(WavFile);
  fclose(CLPFile);
  }
}

void TDrawView::CmControlsplay()
{
  if (!Running) {
   //
   // MCI APIs to open a device and play a .WAV file, using
   // notification to close
   //
   memset(&MciOpenParm, 0, sizeof MciOpenParm);


   MciOpenParm.lpstrElementName = DrawDoc->GetTitle();

  if (mciSendCommand(0, MCI_OPEN, MCI_WAIT | MCI_OPEN_ELEMENT,
                  MCI_PARM2(MciOpenParm))) {
    MessageBox(
      "Open Error - a waveForm output device is necessary \
         to use this demo.",
      "Sound Play", MB_OK);
      return;
   }
   DeviceId = MciOpenParm.wDeviceID;

   // The time format in this demo is in Samples.
   //
   MciSetParm.dwCallback = 0;
   MciSetParm.dwTimeFormat = MCI_FORMAT_SAMPLES;
   if (mciSendCommand(DeviceId, MCI_SET, MCI_SET_TIME_FORMAT,
        MCI_PARM2(MciSetParm)))
    {
```

```
      MessageBox("SetTime Error", "Sound Play", MB_OK);
      return;
    }

    MciPlayParm.dwCallback = (long)HWindow;
    MciPlayParm.dwFrom = 0;
    MciPlayParm.dwTo = 0;
    mciSendCommand(DeviceId, MCI_PLAY, MCI_NOTIFY,
                   MCI_PARM2(MciPlayParm));

    // Modify the menu to toggle PLAY to STOP, and enable PAUSE.
    //
    TMenu menu(HWindow);
//    menu.ModifyMenu(CM_PLAY, MF_STRING, SM_PLAY, "P&ause\tCtrl+A");
    menu.EnableMenuItem(CM_CONTROLSSTOP, MF_ENABLED);

    // Make sure the Play/Stop toggle menu knows we're Running.
    //
    Running = TRUE;

    // Start a timer to show our progress through the waveform file.
    //
    TimeGoing = SetTimer(TIMER_ID, 200, 0);

    // Give enough information to the scrollbar to monitor the
    // progress and issue a re-MCI_OPEN.
    //
    //******** Remove when element name is known
//    SoundBar->SetName(ElementName);

  } else {
      // Resume the playing.
      //
      MciGenParm.dwCallback = 0;
      mciSendCommand(DeviceId, MCI_RESUME, MCI_WAIT,
                     MCI_PARM2(MciGenParm));

      // Toggle Resume menu to Pause.
      //
//      TMenu menu(HWindow);
//      menu.ModifyMenu(SM_PLAY, MF_STRING, SM_PLAY, "P&ause\tCtrl+A");
      Pause = FALSE;
    }
}

void TDrawView::CmControlsstop()
{
  StopWave();
}


//
// Response function MM_MCINOTIFY message when MCI_PLAY is complete.
//
```

```
LRESULT TDrawView::MciNotify(WPARAM, LPARAM)
{
  if (!FlushNotify) {        // Internal STOP/CLOSE, from thumb re-pos?
   StopWave();

   //Make sure the thumb is at the end. There could be some WM_TIMER
   //messages on the queue when we kill it,thereby flushing WM_TIMER's
   //from the message queue.
   //
   int   loVal, hiVal;
   SoundBar->GetRange(loVal, hiVal);
   SoundBar->SetPosition(hiVal);

  } else
   FlushNotify = FALSE;            // Yes, so ignore the close.
  return 0;
}


void TDrawView::EvTimer(UINT)
{
  if (!FlushNotify) {// Internal STOP/CLOSE, from thumb re-pos?
   MciStatusParm.dwCallback = 0;    // No, normal close.
   MciStatusParm.dwItem = MCI_STATUS_LENGTH;
   mciSendCommand (DeviceId, MCI_STATUS, MCI_STATUS_ITEM,
                   MCI_PARM2(MciStatusParm));

   if (WaveLength != MciStatusParm.dwReturn)
    {  // First time it's different update the scrollbar nums
    Invalidate();
    WaveLength = MciStatusParm.dwReturn;
   }

   // Compute the length and ratio and update SoundBar info.
   //
   WaveRatio = int((WaveLength + 32000/2) / 32000);
   if (!WaveRatio)
    WaveRatio = 1;
   SoundBar->SetInfo(WaveRatio, WaveLength);
   SoundBar->SetRange(0, int(WaveLength / WaveRatio));
   SoundBar->SetRuler(int((WaveLength / WaveRatio) / 10));

   // Update the current position.
   //
   MciStatusParm.dwCallback = 0;
   MciStatusParm.dwItem = MCI_STATUS_POSITION;
   mciSendCommand(DeviceId, MCI_STATUS, MCI_STATUS_ITEM,
                  MCI_PARM2(MciStatusParm));

   SoundBar->SetPosition(int(MciStatusParm.dwReturn / WaveRatio));
  }

  FlushNotify = FALSE;          // Yes, ignore this close.
}
```

```
void TDrawView::GetDeviceInfo()
{
  WAVEOUTCAPS waveOutCaps;

  if (!waveOutGetDevCaps(DeviceId, &waveOutCaps, sizeof(waveOutCaps)))
  {
   MessageBox("GetDevCaps Error", "Sound Play", MB_OK);
   return;
  }
}

void
TDrawView::StopMCI()
{
  if (TimeGoing)        // if Timer is Running, then kill it now.
   TimeGoing = !KillTimer(TIMER_ID);

  // Stop playing the waveform file and close the waveform device.
  //
  MciGenParm.dwCallback = 0;
  mciSendCommand(DeviceId, MCI_STOP, MCI_WAIT, MCI_PARM2(MciGenParm));
  mciSendCommand(DeviceId, MCI_CLOSE, MCI_WAIT,MCI_PARM2(MciGenParm));

  Running = FALSE;
  DeviceId = 0;
}


//
// Reset the menus to Play menu and gray the Pause menu.
//
void
TDrawView::StopWave()
{
  if (DeviceId) {
   StopMCI();
//    TMenu menu(HWindow);
//    menu.ModifyMenu(SM_PLAY, MF_STRING, SM_PLAY, "&Play\tCtrl+P");
  }
}

void TDrawView::SetupWindow()
{
  TWindow::SetupWindow();

  progressWin->Show(SW_HIDE);
  progressWin->Attr.Style |= WS_DLGFRAME;
  progressWin->SetRange(0, 100);
  UpdateGauges(0);
}

void TDrawView::UpdateGauges(UINT Percentage)
{
  progressWin->SetValue(Percentage);
}
```