الخلاصة

لقد استحدثت طرق ضغط عديدة بأستخدام تقنيات مختلفة الغرض منها تحقيق نسب ضغط عالية مع المحافظة على جودة الصوت المضغوط، مع الأخذ بنظر الأعتبار أنجاز الضغط بأقل وقت ممكن.
طريقة الضغط الكسوري هي احدى هذه الطرق التي يرافقها ضياع بالبيانات وتعتبر من طرق الضغط الحديثة وتعتمد على عملية ايجاد مجاميع محددة وصغيرة من المعادلات الرياضية التي تصف الاشارة . بأرسال معاملات هذه المعادلات الى محلل الشفرة يمكن اعادة تشكيل الاشارة الاصلية.
بأيجاز ضغط الصوت الكسوري يعتمد على مبدأ نظام التكرار المجزئ(PIFS).
ضغط الصوت الكسوري يستغل ال(Affine redundancy) والتي تتواجد في الصوت، هذه ال (redundancy) ترتبط بالتشابه الذاتي للصوت.
بكلمة اخرى ضغط الصوت الكسوري يقوم بأيجادالأ نماط المتشابهة و التي تتواجد بقياسات مختلفة(different scales) واماكن مختلفة(different places) في الصوت، وبعد ذللك يقوم بحذف اكثر ما يمكن من ال(redundancy).
هذا البحث يهدف الى محاولة التحقق من امكانية تطبيق الضغط الكسوري في عملية ضغط الصوت.
النظام المعتمد يتكون من مرحلتين: المرحلة الأولى مرحلة التشفير(Encoding Unit) و الثانية مرحلة فك التشفير (Decoding Unit).
في مرحلة التشفير يجزأ الصوت الأصلي الى نوعين من الكتل، كتل تدعى كتل المجال المقابل (Range Pool) وهي كتل غير متداخلة، و كتل تدعى كتل المجال (Domain Pool) و التي من الممكن ان تكون متداخلة. ومن ثم يتم تقطيع الصوت بأستخدام طريقة التقطيع المتساوي للكتل الى كتل متساوية الحجم. بعد ذلك يتم ايجاد افضل كتل في المجال لكل كتلة في المجال المقابل وذلك بتطبيق احد انواع التحويلات وتدعى(Affine Transformation) .تنتهي مرحلة التشفير بخزن تفاصيل (معاملات)هذه التحويلات لكل كتلة من كتل المجال المقابل .ان عملية ايجاد الكتل المتشابهة تتطلب عمليات حسابية معقدة تستغرق وقت طويل وهذا ما يؤخذ على طريقة الضغط الكسوري كنقطة ضعف.
ان البرامجيات التي تم بناؤها لهذا الغرض تم اختبارها باستخدام خمسة نماذج لبيانات صوتية.
لقد تم تنفيذ النظام المعتمد بأستخدام فيجوال بيسك6.0 كلغة برمجة، تم استخدام معدل الخطأ التربيعي(MSE) و نسبة الأشارة الى التشويش(PSNR) كمعاملات حساب دقة النتائج المستخلصة من التقنية الكلية في النظام المعتمد.
و كانت بعض النتائج المستخلصة من البحث هي بالنسبة للعينة الاولى في حالة حجم الكتلة تساوي (٦٠) فأن نسبة الضغط (١:١٥,٥١)، اما العينة الثانية في حالة حجم الكتلة تساوي (٤٠) فأن نسبة الضغط (١:١١,٥١)، وفي حالة حجم كتلة صغيرة كما في العينة الثالثة والتي تساوي (٢٠) فان قيمة ال(PSNR) جيدة و تساوي (dB ٣٥,٧٦)، و كذالك للعينة الأولى في حالة حجم الكتلة تساوي (١٠) فان قيمة ال(PSNR) تساوي (dB ٤٥,٦٣).

Abstract

Fractal audio compression is based on the concept of a partitioned iterated function system (PIFS). Fractal audio compression exploits the affine redundancy that is commonly present in audio; this redundancy is related to the similarity of an audio with itself.

In other words, fractal audio compression finds similar patterns that exist in different scales and different places in audio, and then eliminates as much redundancy as possible.

In this work the possibility of implementing fractal audio compression is investigated.

The implemented system consists of two major units; the first is the Encoding unit and the second is the Decoding unit.

Encoding is done by partitioning the range pool (which is the original audio) into non-overlapping blocks, called ***range blocks***, and partitioning the domain pool (which is the result of the original audio after down sampling) into overlapped blocks with the same size of range blocks called ***domain blocks***. A fixed size-partitioning scheme is used to partition the domain pool and the range pool. After generating the range and domain pools, for every range block, the best-matching domain block in the domain pool is searched for by performing a set of affine transformations on them. Thus the encoding is completed by saving the optimal affine parameters for every range block. The **Decoding** process can be done by repeatedly applying the affine transformation on an initially blank audio and its subsequent reconstructed audio, until it completely reconstruct an approximate wave to the original audio.

The time required to compress an audio file is affected by the size of each block being extracted from the proceed audio file; this means smaller block size implies longer time required to compress the corresponding audio file.

The implemented system was tested using five wave samples of data.

The proposed work was implemented by using Visual Basic (6.0) as a programming language, the fidelity measure MSE and PSNR were used to check the results of the whole implemented technique.

The best results obtained from the implemented system were for the test (sample -1) in case were block size equal to (60) the compression ratio is (15.51:1), also for test (sample-2) were block size equal to (40) the compression ratio is (11.03:1), and for small block size as in test (sample-3) were the block size is (20) the value of the PSNR is good that its equal to (35.76 dB), also for test sample-1 when the block size is (10) the PSNR value (45.63 dB).

Acknowledgment

I would like to express my sincere appreciation to my research supervisor, Dr. Loay A. George, for giving me the major steps to go on to explore the subject, shearing with me the ideas in my research “Fractal Audio Compression” And perform the points that I felt were important.

Also I wish to thank, Dr. Laith A. Al-Ani my supervisor for his available advice and encouragement. Grateful thanks for the Head of Department of Physics Science Dr. Ahmad K. Ahmad.

I wish to thank the staff of Physics Science Department at the AL-Nahrain University for their help.

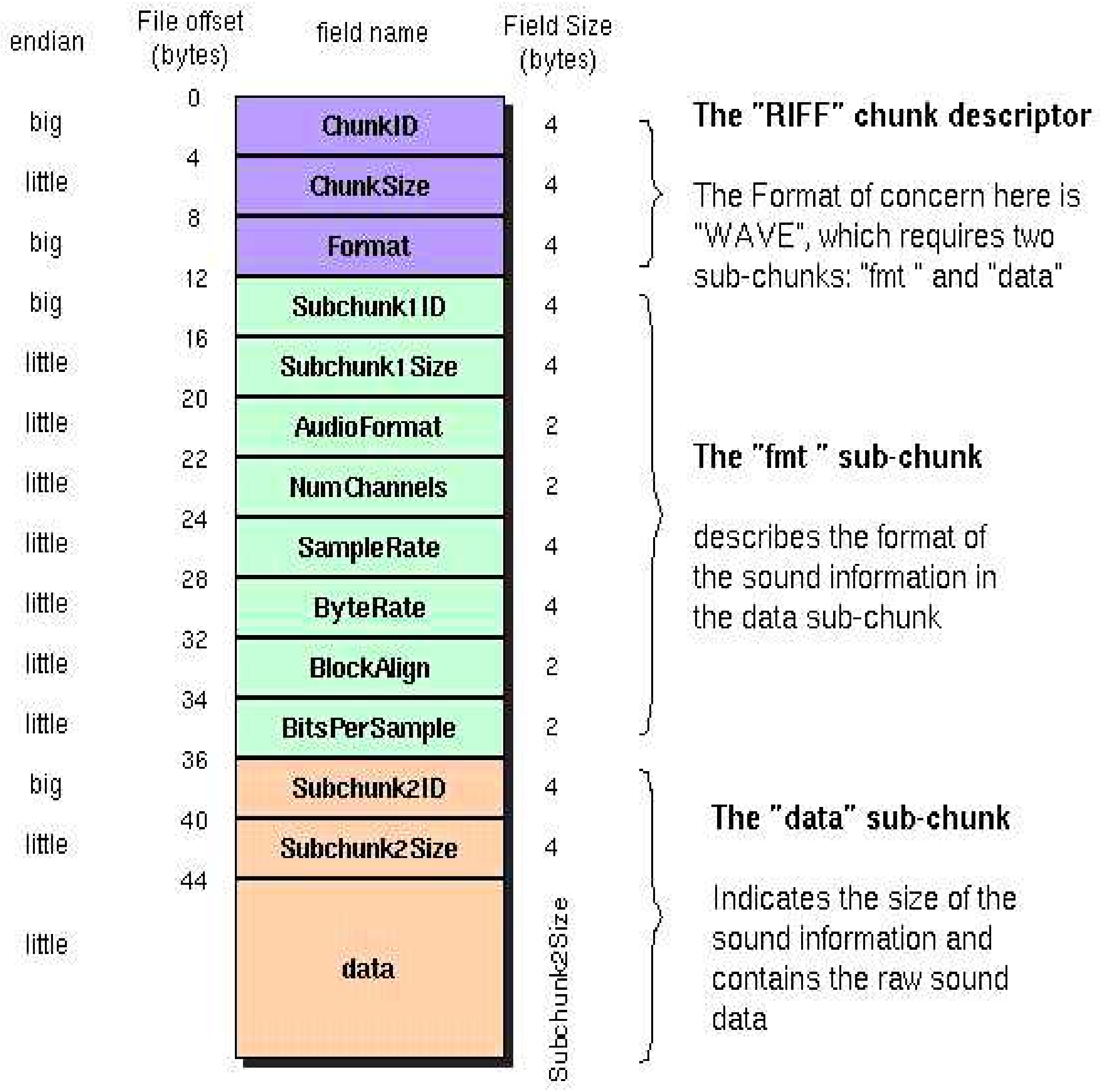
I would like to say "thank you" to my faithful friends for supporting and giving me advises.

Appendices

Appendix A

Wave PCM Sound File Format

The WAVE file format is a subset of Microsoft's RIFF specification for the storage of multimedia files. A RIFF file starts out with a file header followed by a sequence of data chunks. A WAVE file is often just a RIFF file with a single "WAVE" chunk which consists of two sub-chunks -- a "fmt" chunk specifying the data format and a "data" chunk containing the actual sample data. Call this form the "Canonical form".



Offset	Size	Name	Description

The canonical WAVE format starts with the RIFF header:

0	4	**ChunkID**	Contains the letters "RIFF" in ASCII form (0x52494646 big-endian form).
4	4	**Chunk Size**	36 + SubChunk2Size, or more precisely: 4 + (8 + SubChunk1Size) + (8 + SubChunk2Size) This is the size of the rest of the chunk following this number. This is the size of the entire file in bytes minus 8 bytes for the two fields not included in this count: ChunkID and ChunkSize.
8	4	**Format**	Contains the letters "WAVE" (0x57415645 big-endian form).

The "WAVE" format consists of two subchunks: "fmt" and "data":
The "fmt" sub chunk describes the sound data's format:

12	4	**Subchunk1ID**	Contains the letters "fmt" (0x666d7420 big-endian form).
16	4	**Subchunk1Size**	16 for PCM. This is the size of the rest of the Subchunk which follows this number.
20	2	**AudioFormat**	PCM = 1 (i.e. Linear quantization) Values other than 1 indicate some form of compression.
22	2	**NumChannels**	Mono = 1, Stereo = 2, etc.
24	4	**SampleRate**	8000, 44100, etc.
28	4	**ByteRate**	== SampleRate * NumChannels * BitsPerSample/8
32	2	**BlockAlign**	== NumChannels * BitsPerSample/8 The number of bytes for one sample including all channels. I wonder what happens when this number isn't an integer?
34	2	**BitsPerSample**	8 bits = 8, 16 bits = 16, etc.
	2	**ExtraParamSize**	if PCM, then doesn't exist
	X	**ExtraParams**	space for extra parameters

The "data" subchunk contains the size of the data and the actual sound:

36	4	**Subchunk2ID**	Contains the letters "data" (0x64617461 big-endian form).

40	4	**Subchunk2Size**	== NumSamples * NumChannels * BitsPerSample/8 This is the number of bytes in the data. You can also think of this as the size of the read of the subchunk following this number.
44	*	**Data**	The actual sound data.

Notes:

- The default byte ordering assumed for WAVE data files is little-endian. Files written using the big-endian byte ordering scheme have the identifier RIFX instead of RIFF.
- The sample data must end on an even byte boundary. Whatever that means.
- 8-bit samples are stored as unsigned bytes, ranging from 0 to 255. 16-bit samples are stored as 2's-complement signed integers, ranging from -32768 to 32767.
- There may be additional subchunks in a Wave data stream. If so, each will have a char [4] SubChunkID, and unsigned long SubChunkSize, and SubChunkSize amount of data.
- RIFF stands for *Resource Interchange File Format*.

الجمهورية العراقية
وزارة التعليم العالي و البحث العلمي
جامعة النهرين

ضغط الصوت بأستخدام الكسوريات

رسالة
مقدمة الى قسم علوم الفيزياء جامعة النهرين كجزء
من متطلبات نيل درجة الماجستير في علوم الفيزياء

من قبل
وسام فوزي جاسم محمد
(بكالوريوس جامعة النهرين ٢٠٠٢)

أشراف

د. لؤي أدور جورج   د. ليث عبد العزيز العاني

حزيران ٢٠٠٥   جمادي الاولى ١٤٢٥

Supervisor Certification

We certify that this thesis was prepared under our supervision at the Department of Physics/ College of Science/ Al-Nahrain University, by **Wesam Fawzi Jassim Mohammed** as partial fulfillment of the requirements for the degree of Master of Science in Physcis.

Signature:	Signature:
Name: **Dr. Loay A. George**	Name: **Dr. Laith A. Al-Ani**
Title: **Senior Rreasercher**	Title: **Assist Professor**
Date: **/ / 2005**	Date: **/ / 2005**

In view of the available recommendations, I forward this thesis for debate by the examination committee.

Signature:

Name: **Dr. Ahmed K. Ahmed**

Title: **Head of the Department of Physics Science, Al-Nahrain University.**

Date: **/ / 200**5

Certification of the Examination Committee

We chairman and members of the examination committee, certify that we have studied the thesis entitled (**Fractal Audio Compression**) presented by the student **Wesam Fawzi Jassim Mohammed** and examined him in its contents and in what is related to it, and we have found it worthy to be accepted for the degree of Master of Science in Physics Science with grade **Very Good**.

Signature:
Name: **Dr. Geen Istefan**
Title: **Assistant Professor**
Date: **/ / 2005**
(Chairman)

Signature:
Name: **Dr. Bushra K. AL-Abudi**
Title: **Assistant Professor**
Date: **/ / 2005**
(Member)

Signature:
Name: **Dr. Ali A. Dawood**
Title: **Assistant Professor**
Date: **/ / 2005**
(Member)

Signature:
Name: **Dr. Loay A. George**
Title: **Senior Rreasercher**
Date: **/ / 2005**
(Supervisor)

Signature:
Name: **Dr. Laith A. Al-Ani**
Title: **Assist Professor**
Date: **/ / 2005**
(Supervisor)

Signature:
Name: **Dr. Laith A. Al-Ani**
Title: **Dean of College of Science**
Date: **/ / 2005**

CHAPTER ONE
GENERAL INTRODUCTION

1.1Digital Audio

Audio is the range of frequencies within human hearing (approx. 20Hz at the low to a high of 20,000Hz). In computers there is an audio card contains a special built-in processor and memory for processing audio data and sending them to speakers in the computer. An audio file is a record of captured sound that can be played back. Sound is a sequence of naturally analog signals that are converted to digital signals by the audio card, using a microchip called an analog-to-digital converter (ADC). When sound is played, the digital signals are sent to the speakers where they are converted back to analog signals that generate varied sound **[Kie98]**.

Audio files are usually compressed for storage or faster transmission. And can be sent in short stand-alone segments (for example, as files in the Wave File format). In order for users to receive sound in real-time for a multimedia effect, listening to music, or in order to take part in an audio or video conference, sound must be delivered as streaming sound. More advanced audio cards support Wavetable, or precaptured tables of sound. The most popular audio file format today is MP3 (MPEG-1 Audio Layer-3) **[Kie98]**. The digital representation of audio data offers many advantages: high noise immunity, stability, and reproducibility. Audio in digital form also allows the efficient implementation of many audio processing functions (e.g., mixing, filtering, and equalization) through the digital computer. The conversion from the analog to the digital domain begins by sampling the audio input in regular, discrete intervals of time and quantizing the sampled values into a discrete number of evenly spaced levels. The digital audio data consists of a sequence of binary values representing the number of quantizer levels for each audio sample. The method of representing each sample with an independent codeword is called Pulse Code Modulation (PCM) **[Pan93]**.

1.2 Data Compression [Sal00]

Data transmission and storage cost money. The more information being dealt with, the more it costs. In spite of this, most digital data are not stored in the most compact form. Rather, they are stored in whatever way makes them easiest to use, such as: ASCII text from word processors, binary code that can be executed on a computer, individual samples from a data acquisition system, etc. Data compression is the general term for the various algorithms and programs developed to address this problem. A compression program is used to convert data from an easy-to-use format to one optimized for compactness. Likewise, an uncompressing program returns the information to its original form.

1.3 Types of Data Compression

By considering the characteristics of the reconstructed data after the process of compression and decompression, data compression can be divided generally into two major types: lossless and lossy.

1.3.1 Lossless Data Compression

Lossless data compression has a property that after the compression and decompression operation, an identical duplicate of the original is reproduced. Figure (1.1) shows the typical block diagram of lossless data compressor. The possibility of this operation exists, and Shannon **[Sha01]** has shown a theoretical limit for this compression operation by considering the statistical characteristics of the source data stream. In fact, lossless compression system can be built by making use of symbol probabilities of the data stream. By representing frequently appearing symbols with shorter codes and rarely appearing ones with longer codes, we can encode the original to a stream of codes with shorter total length. These systems are usually called statistical coding systems, since they rely on the statistics of the incoming source to determine coding symbol **[Zol98]**.

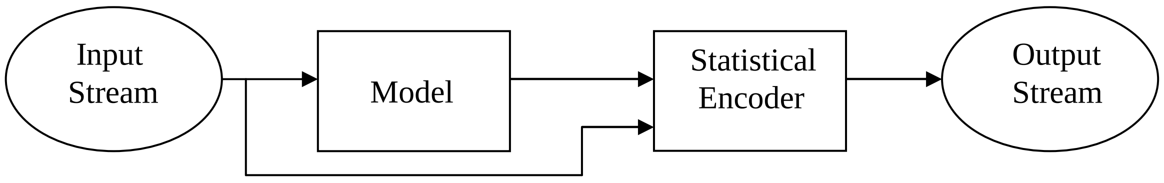


Figure (1.1) Lossless Compressor

Many lossless compression techniques exist nowadays. For examples, Huffman coding, arithmetic coding, and Lempel-Ziv algorithms are among the most efficient of these lossless compression techniques, they are suitable for wide range applications, from hard disk file compression to digital medical image archiving. In fact, the lossless property of these techniques makes them good for most applications universally, since data are compressed without any loss. However, lossless compression often has a low compression ratio. Sometimes the compression is so insignificant that making it not attractive enough in some applications, which require large compression ratio, such as image and speech compression **[Wat95, Zol98]**.

1.3.2 Lossy Data Compression

As an alternative to lossless compression, lossy compression techniques can be considered if we want to have a higher compression ratio. The important point of lossy compression is that, its compression procedure can be adjusted to sacrifice some accuracy in order to gain a lot of compression. In other words, degradations are allowed in the reconstructed data. Some times, the gain of lossy compression is so significant that it leads to a very small compressed data size, which contrasts lossless compression. Most of the lossy compression techniques include a quantization stage as shown in Figure (1.2) to perform lossy quantizing procedures **[Sal00, Wat95]**.

This is especially the case for lossy compression of multimedia signals including images, sounds, and moving pictures. The resulting quantized coefficients are then encoded losslessly to form an output stream. In some techniques, this quantization process is very efficient that it can achieve 50 times compression or even more. Nowadays, many multimedia applications use modern lossy compression techniques **[Zol98, Wan00]**.

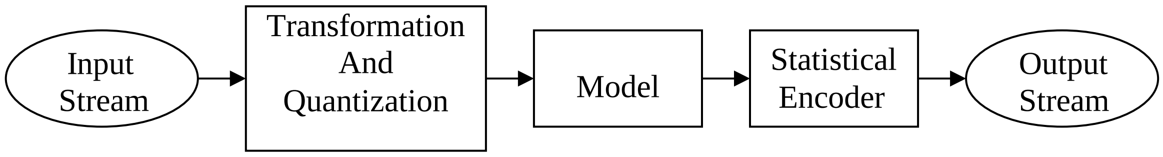


Figure (1.2) Lossy Compressor

These applications include image, speech, and video compression. Some popular examples are JPEG for image compression, GSM for speech compression, and MPEG-2 for video compression. Nevertheless, some mission-critical applications, such as medical imaging and satellite image transmission, still require lossless compression techniques. Since lossless and lossy compression techniques have different advantages, they are both important to multimedia compression **[Zol98, Wan00]**.

1.4 Audio Compression

In order to more efficiently broadcast or record audio signals, the amount of information required to represent the audio signals may be reduced. In the case of digital audio signals, the amount of digital information needed to accurately reproduce the original pulse code modulation (PCM) samples may be reduced by applying a *digital compression* algorithm, resulting in a digitally compressed representation of the original signal. (The term *compression* used in this context means the compression of the amount of digital information, which must be stored or recorded). The goal of the digital compression algorithm is to produce a digital representation of an audio signal which, when decoded and reproduced, sounds the same as the original signal, while using a minimum of digital information (bit-rate) for the compressed (or encoded) representation **[ATS01]**.

Digital compression of audio is useful wherever there is an economic benefit to be obtained by reducing the amount of digital information required to represent the audio. Typical applications are in satellite or terrestrial audio broadcasting, delivery of audio over metallic or optical cables, or storage of audio on magnetic, optical, semiconductor, or other storage media **[ATS01]**.

Audio compression algorithms were created to enable audio to be saved more effectively. A compressor codec takes an original uncompressed audio track and reduces its size. Because of the smaller size, the speed requirements for the storage devices are greatly reduced. To play back the compressed audio data, a decompression algorithm is used to decompress the data so it can be heard.

1.5 Fractal Compression [Xia04]

Fractal compression is a lossy compression method used to compress images using fractals. The method is best suited for photographs of natural scenes. The fractal compression technique relies on the fact that in certain images, parts of the image resemble other parts of the same image.

Fractal compression seems to be one of those technologies with "a great future behind it". It promised much in the late 1980s, when in some circumstances it appeared to compress much better than JPEG, its main competitor in those days.

Fractal compression is a radical departure from the conventional image compression techniques. The difference between it and the other techniques is much like the difference between bitmapped graphics and vector graphics. Rather than storing data for individual pixels, fractal compression stores instructions or formulas for creating the source (image or audio).

Like vector quantization, fractal compression is asymmetrical. Although it takes a long time for compression, decompression is very fast. These asymmetrical methods are well suited to such applications as video on a CD-ROM where the user doesn't care about compression but does expect to see results quickly. Decompression simply reads the mathematical formulas and recreates the source.

The tough part is generating the formulas to correctly represent the source. Fractal compression assumes that every image is composed of smaller images just like them. Blue sky in an image is composed of smaller patches of blue. Tree branches can be broken into smaller branches and then twigs those all have similar structure. The compression technique tries to find as many of these relationships in an image and then describe them with mathematical formulas. This is done within regions of an image called domain regions. These domain regions are determined by using techniques such as frequency analysis, edge detection, and texture-variation analysis.

Like other lossy compression schemes, fractal compression involves a tradeoff, which is little different from the other methods. The tradeoff is between image quality and compression time. The longer the encoder has to create the descriptive formulas, the higher the quality of the output image. Like all other lossy compression schemes, fractal compression also introduces artifacts. These include softness and substitution of details with other details. This substitution is typically undetected in natural images.

1.6 Aim of Thesis

This research aims to study the performance of a lossy audio compression system based on partitioned iterated function system (PIFS) coding method. The effect of all control parameters of the performance of the designed and implemented compression system will investigate in an attempt to achieve a good compression ratio with keeping the audio quality above an acceptable level.

1.7 Related Work

1. In 1993, Sinha and Tewfik **[Sin93]** presented a low bit rate transparent audio compression using a dynamic dictionary and optimized wavelets. The authors proposed an audio synthesis coding method, which employs an optimized wavelet transform (WT), based adaptive transform coding to exploit perceptual masking. In addition, a dynamic dictionary was used to extract source redundancies.

2. In 1995, Baharav, Krupnik, and Karnin **[Bah95]** presented in their paper the basic fractal coder suggested by Jacquin. The coder finds and encodes the parameters of a partitioned iterated function system (PIFS), which approximate the signal as a fixed point of a contractive transformation.

3. In 1995, Degener **[Deg95]**, her research group at the technical university of Berlin developed a speech compression part of the (GSM) Global System for Mobile telecommunication protocol suite that is currently used as the Europe's most popular protocol for digital cellular phones.

The GSM consists of an input of frame of 160 samples; 13-bit linear PCM values sampled at 8 kHz, so each frame covers 20 ms.

4. In 1996, Shamoon **[Sha96]**, had presented algorithms for encoding high fidelity audio at low bit rate. The ultimate goal of this work is to provide fast algorithms for sub-band perceptual coding that are capable of audio coding at bit rates that permit transmission of high fidelity audio over broadcast and telephone channels, and storage of audio on low capacity media.

5. In 2002, Mohammed Abbas **[Aba02]**, proposed a genetic algorithm as a new lossy compression method for multimedia data files (image and audio), and he concluded that using genetic algorithm with iterated function system can improve the compression performance in some respects, but it fails in others.

6. In 2003, Yokoyama, Watanabe, and Sugawara **[Yok03]**, proposed in their paper a new retrieval system of images using PIFS codes. In this paper they suggested a new retrieval technique that uses compression codes; especially they used fractal image compression (FIC) method. This compression method is relatively recent technique and based on the self-similarity of images. Exploiting the robust property of this compression method, they have developed a new similarity-based retrieval system.

CHAPTER TWO
BASIC CONCEPTS OF FRACTAL

2.1 Introduction

There are many shapes in the natural world; they look so complex that we could not express them by several simple functions. But after B. B. Mandelbrot (1977) devised the notion of ‘fractal’ [**Man77**], we are able to express many natural shapes by some simple algorithms. From that time, dedicated hardware and software of computer systems have been improved day by day. And various researches on how to construct fractal shapes or how to imitate natural creatures have been developed [**Bra88**].

The term *fractal* (from Latin *fractus* -means irregular, fragmented) applies to objects in space or fluctuations in time that possess a form of *self-similarity* and cannot be described within a single absolute scale of measurement. Fractals are recurrently irregular in space or time, with themes repeated like the layers of an onion at different levels or scales. Fragments of a fractal object or sequence are exact or statistical copies of the whole and can be made to match the whole by shifting and stretching. Sequential fractal scaling relationships are observed in many physiological processes. Spatial structures of many living systems are fractal. Fractal geometry has evoked a fundamentally new view of how both nonliving and living systems result from the coalescence of spontaneous self-similar fluctuations over many orders of time and how systems are organized into complex recursively nested patterns over multiple levels of space **[Klo00]**.

2.2 Definition of Fractal

The formal mathematical definition of fractal is defined by Benoit Mandelbrot. It says that a fractal is a set for which the Hausdorff Besicovitch dimension strictly exceeds the topological dimension **[Fis94, Man83]**. However, this is a very abstract definition. Generally, we can define a fractal as a rough or fragmented geometric shape that can be subdivided in parts, each of which is (at least approximately) a reduced-size copy of the whole. Fractals are generally self-similar and independent of scale.

2.3 Properties of Fractal

A geometric figure or natural object is said to be fractal if it combines the following characteristics [**Fis94**]:

a) Its parts have the same form or structure as the whole, except that they are at a different scale and may be slightly deformed.

b) Its form is extremely irregular or fragmented, and remains so, whatever the scale of examination.

c) It contains "distinct elements" whose scales are very varied and cover a large range.

d) They have a perimeter of infinite length but an area limited.

e) Fractional dimension: A non-integer dimension. We know that the dimension of lines, squares, and cubes are respectively 1, 2, and 3. For example the dimension of a fractal may be 1.342.

f) Formation by iteration.

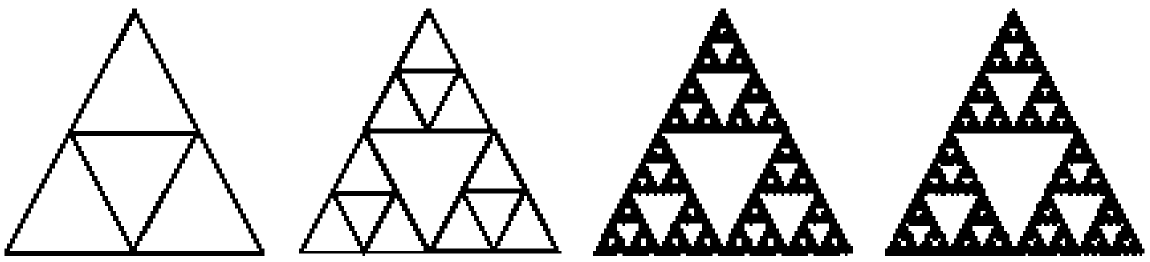
2.4 Fractal Development [Xia04]

Unfortunately, fractal was not developed for data compression in the first place, but at that time it was considered as a different kind of geometry introduced by the IBM mathematician Benoit B. Mandelbrot through his book ”The Fractal Geometry of Nature" published at 1977. At 1981, the mathematician John Hutchinson found the theory of iterated function system to model collections of contractive transformations in a metric space as dynamical systems, which later provided some theoretical support of recognizing fractal in the metric space. It was Michael Barnsley, eventually, who generated the fractal model using iterated function systems (IFS’s), and led to encode images to achieve significant compression. However, Barnsley’s image compression algorithm based on fractal mathematics was inefficient and unpractical suffering too big searching space problem. At 1988, one of Barnsley’s ph.D students, Arnaud Jacquin, suggested a modified scheme for representing images, called partitioned iterated function systems (PIFS’s). The basic idea of the algorithm is to convert an image into the PIFS’s, instead of looking at the whole image. It immediately made fractal image compression more practical, however with sacrificing the compression ratio. After Jacquin’s PIFS, there were many other modified schemes, but none of them made any significant progress. Most of later publications on the fractal subject stay on the PIFS, but focus on the possible improvements. The two big problems of Jacquin’s algorithm are the partition scheme selection for encoding, and the speed problem for encoding.

2.5 Fractal Examples

In this section two fractal examples are provided for demonstration purposes.

The first example is generating Sierpinski’s Triangle using IFS. We can see from Figure (2.1) that the Sierpinski’s triangle can be generated by infinitely repeating a procedure of connecting the midpoints of each side of the triangle to form four separate triangles, and cutting out the triangle in the center [**Kis99**].



(a) Initial Image **(b) First Iteration** **(c) Fifth Iteration** **(d) Sixth Iteration**

Figure (2.1) Sierpinski triangle example of generating sequence by iteration.

This process can be viewed as three transformations map the original triangle to the new 3 triangles as shown in Figure (2.2).

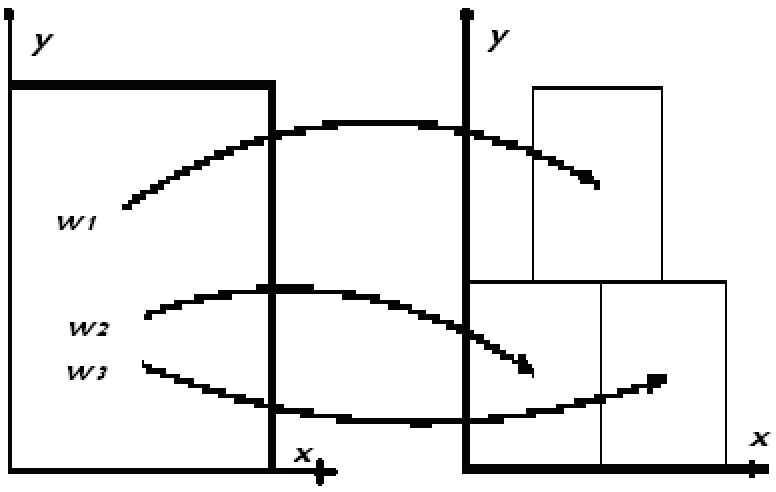


Figure (2.2) Sierpinski Triangle mapping using three affain transformations.

The mathematical expressions of the transformations are the followings:

$$w_1\begin{bmatrix}x\\y\end{bmatrix}=\begin{bmatrix}0.5&0\\0&0.5\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix},\ldots\ldots\ldots(2-1)$$

$$w_2\begin{bmatrix}x\\y\end{bmatrix}=\begin{bmatrix}0.5&0\\0&0.5\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}+\begin{bmatrix}0\\0.5\end{bmatrix},\ldots\ldots\ldots(2-2)$$

$$w_3\begin{bmatrix}x\\y\end{bmatrix}=\begin{bmatrix}0.5&0\\0&0.5\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}+\begin{bmatrix}0.5\\0\end{bmatrix},\ldots\ldots\ldots(2-3)$$

It is important to observe that there is hardly to remark any visible difference between the fifth and the sixth iterations shown in Figure (2.1). The Sierpinski's triangle sequence thus visually converges to one triangle within certain threshold. Reversely, if we take the sequence backward, for the Sierpinski triangle sequence, we only need to know the mapping which is w_1, w_2 and w_3. This is the essence of how fractal compression works.

The second example is shown in Figure (2.3). It demonstrates one domain-range match, which is calculated through the rotation and modification. And since the domain block is twice the size of the range block, the reduction is needed at last.

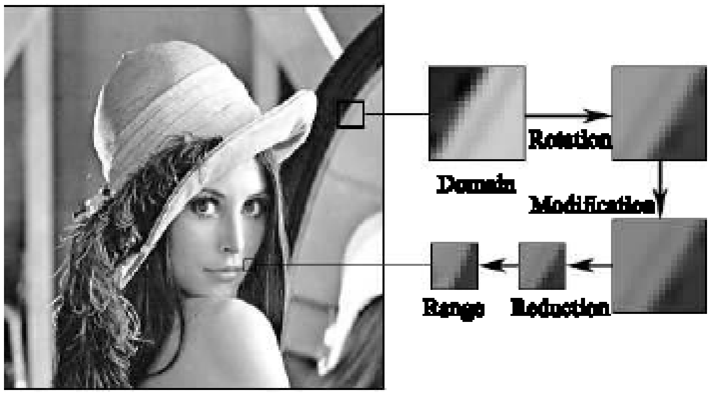


Figure (2.3) Fractal encoding from a domain mapping to a range domain for Lena image.

Essentially, in image sense, the rotation, and modification are the process to get a proper set of scalings and offsets, which adjust the luminance of the domain block (in this case for a grey-scale image). The Lena image is a widely accepted test figure in image compression. It gives the idea that natural images have self-similarities or patterns embedded. With this belief, our research work is an attempt to discover the possibility of using this concept on audio **[Xia04]**.

2.6 Fractal Coding

Fractal coding is one of the promising new coding techniques for high compression ratios, which are beginning to be adopted worldwide by the way of the Internet. It is based on the work of Barnsley and Jacquin on fractals and Iterated Function Systems (IFSs) **[Bar88-Fis94]**. Blocks of a source (image or audio) are considered as affine transformations of other blocks taken from the image itself.

2.6.1 Encoding **[Bah95]**

Encoding is done by partitioning the source (image or audio) into non-overlapping blocks, called *range blocks* (denoted R_i, where i is the block index), and into typically overlapping blocks to form what is called the *domain pool*. A spatial contraction is performed on the domain pool, by an operator φ(usually averaging), so that the resulting blocks (which we denote D_j) are of the size of the range blocks.

Each range block is linearly estimated from a contracted domain block according to $\hat{R}_i = a_i.D_{ji} + b_i$, where a_i is called the scale parameter, b_i the offset parameter, and ji is the index denoting the best domain block to get $\hat{R}_i$ as close as possible to R_i.

2.6.2 Decoding **[Bah95]**

Decoding can be done by starting with some arbitrary initialized range pool (image or audio), then the domain pool is generated by down sampling the range pool, estimating the range blocks of a new source from the contracted domain blocks, and iterating the above last two steps until a fixed point is reached, or nearly so (thus this method is called IFS – Iterated Function System). If the range block estimators are good enough, then the fixed point source is much like the original source.

2.7 Affine Transformations **[Bra93, Fis92]**

Most literatures in fractal compression define the mapping function *f to* be affine to simplify the computation. An ***affine transformation*** w: $R^n \to R^n$ can always be written as *w* = *Ax* + *b*, where $A \in R^{n \times n}$ *is an n* × *n matrix* and $b \in R^n$ is an offset vector. The transformation is contractive when its linear part is contractive. The contractive depends on the metric used to measure distance. And because affine transformation is linear, we can use norm $\|.\|$ in R^n to define the metric. Then

$$\|A\| = \sup_{\vec{x} \in R^n} \quad \text{If} \quad \|A\vec{x}\| / \|\vec{x}\| < 1, \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(2-4)$$

The contractive under the sup norm of a complete metric space is guaranteed if the above condition is satisfied (i.e., the left hand side is always less than one). However, Wohlberg and Jager in their review **[Woh99]** pointed out that this restriction is sufficient but not necessary for convergence; empirical evidence indicates that convergence is often achieved even if $\|A\,\vec{x}\| / \|\vec{x}\|$ greater than one, although smaller values provide more rapid convergence on decoding. The eventually contractive mapping function is sufficient to ensure the convergence of the mapping. Setting the equation less than one guarantees the contractive mapping, which is a stronger argument than eventually contractivity. But unfortunately, the affine transformation and sup norm metric cannot explicitly give us the bound of ensuring eventually contractive. Affine transformation also suffers from some problems as a linear transformation that has limited capabilities of mapping and requests geometrically identical mapping parties.

2.8 Contractive Transformations [Fis92]

A transformation w is said to be contractive if for any two points $P1, P2$ the distance:

$$d\,(w\,(P1), w\,(P2)) < sd\,(P1, P2), \quad \text{..} (2-5)$$

For some $s < 1$, where d is the distance. This formula says the application of a contractive map always brings points closer together (by some factor less than 1).

2.9 Fixed Point Theorem [Fis92]

This theorem says something that is intuitively obvious: if a transformation is contractive then when applied repeatedly starting with any initial point, we converge to a unique fixed point.

If X is a complete metric space and $W: X \rightarrow X$ is contractive, then W has a unique fixed point $|w|$.

2.10 Partitioned Iterated Function Systems (PIFS's)

The Sierpinski triangle (shown in Figure 2.1) demonstrates the way of using IFS. However, unlike the example; our real spaces are very irregular. In most cases, it would be rather impossible to find such a perfect mapping for the whole space. Thus Jacquin introduced the ***partitioned iterated function system*** (PIFS) in his works **[Jac92]**. A PIFS is a generalization of an IFS, and attempts to ease the IFS computation by partitioning the whole space into subspaces. In other words, PIFS is a restricted version of IFS.

One problem brought up from the PIFS is the partition. The space has to be partitioned into subspaces. It is necessary to be sure the addition of the subspaces covering the original space. Also, the partition scheme dominates the final map set, which means the compression process in general **[Jac92]**.

Jacquin first introduced the PIFS on the fractal image compression. Image space is naturally recognized as a 2D space. The partition scheme is simply partitioning the whole space twice to the range set and the domain set. Both sets cover the whole image space, with the domain set allowing overlapping. As a shortcoming of using affine transformation, the partition schemes for the domain set and the range set have to give the same geometric shaped domain and range blocks, which are usually squares or rectangles. Domain block is set to be twice as big as range block in Jacquin's original scheme in [**Jac92**], which is widely accepted in fractal image compression. The reason allowing domain overlapping is to smooth artifacts between blocks in decoding process. The mapping between the domain and the range blocks is as demonstrated in Figure (2.4). For each range block, find a proper domain block to map to. The final map set is composed of mappings for each range block from the range set.

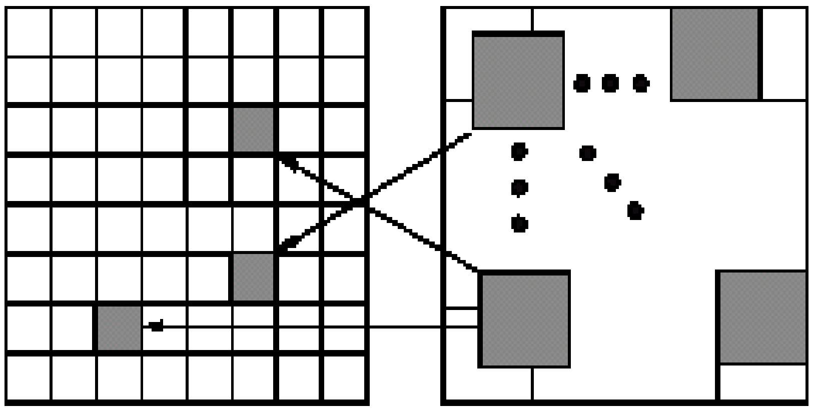
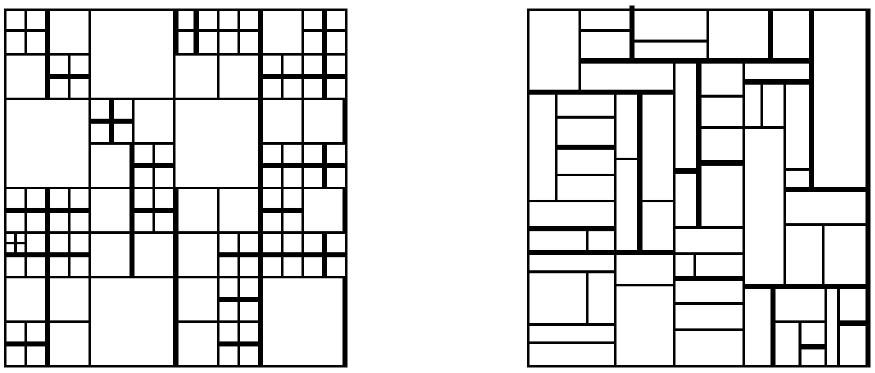


Figure (2.4) Mapping from the domain set to the range set.

Among most of fractal image compression range partition schemes appearing in literatures, Quadtree partition and Horizontal-Vertical (HV) partition are the most popular two schemes being used. We show two examples in Figure (2.5) for both partition schemes. The review in [**Woh99**] classifies range partition schemes into right-angled partition schemes and triangular and polygonal range partition schemes. Both quadtree and HV schemes belong to the first category.



(a) Quadtree range partition **(b) HV range partition**

Figure (2.5) Examples of quadtree and HV range partition schemes

PIFS is recognized as a significant improvement for IFS. It reduces large amount of searching time both theoretically and practically. Furthermore, it provides some possible aspects of improving fractal encoding like using different partition schemes or taking different mapping methods. Comparing with some more advanced methods of generating fractals such as Weighted Finite Automata, PIFS also has the beauty of simplicity. For the above reasons, our research on fractal audio compression uses PIFS in the same way like conventional fractal image compression methods.

2.11 Derivation of Scale and Offset Equations

To compute the s and o one should compute the values of a scale (s) and offset (o) which minimize the value of the distortion error equation (E). The minimum of the distortion error (E) occurs when the partial derivatives (of E) with respect to (s) and (o) are zero.

$$E=\sum_{i=n}^{n}(r-r')^2 \text{ ,………………………………………………………(2 - 6)}$$

Where: $r'=sd_i+o$ where (s) is the scale and (o) is the offset

$$E=\sum_{i=n}^{n}(r_i-sd_i-o)^2 \text{ ,……………………………………………………(2 - 7)}$$

Differentiate the distortion error equation (E) with respect to (s) and (o):

$\frac{\partial E}{\partial s}=0\,,\ \frac{\partial E}{\partial o}=0$, the differentiation of (E) with respect to (s) as follows:

$$\frac{\partial E}{\partial s}=\sum_{i=1}^{n}2(r_i-sd_i-o)(-d_i)=0\,,\sum_{i=1}^{n}(-r_id_i+sd_i^{\,2}+od_i)=0 \text{ ,…….(2- 8)}$$

$$-\sum_{i=1}^{n}r_id_i+\sum_{i=1}^{n}sd_i^{\,2}+\sum_{i=1}^{n}od_i=0 \text{ ,……………………………….(2 - 9)}$$

rearrange equation (2-9) to get:

$$s\sum_{i=1}^{n}d_i^{\,2}+\sum_{i=1}^{n}od_i=\sum_{i=1}^{n}r_id_i \text{ ,……………………………………………(2-10)}$$

Now differentiate the distortion error equation (E) with respect (o):

$$\frac{\partial E}{\partial o}=\sum_{i=1}^{n}2(r_i-sd_i-o)(-1)=0\,,\sum_{i=1}^{n}(-r_i+sd_i+o)=0 \quad \text{,…............(2-11)}$$

$$-\sum_{i=1}^{n}r_i+\sum_{i=1}^{n}sd_i+\sum_{i=1}^{n}o=0\ ,\ s\sum_{i=1}^{n}d_i+o\sum_{i=1}^{n}1=\sum_{i=1}^{n}r_i \text{ ,……………………(2-12)}$$

$$s\sum_{i=1}^{n}d_i+on=\sum_{i=1}^{n}r_i \text{ ,…………………………………………………(2-13)}$$

rearrange equation (2-13) to get:

$$o=\frac{\sum_{i=1}^{n}r_i-s\sum_{i=1}^{n}d_i}{n} \text{ ,…..(2-14)}$$

substitute equation (2-14) in equation (2-10) to get:

$$s\sum d_i^{\,2}+\left(\frac{\sum_{i=1}^{n} r_i - s\sum_{i=1}^{n} d_i}{n}\right)\sum_{i=1}^{n} d_i=\sum_{i=1}^{n} r_i d_i \;,\text{......................................(2-15)}$$

rearrange equation (2-15) to get:

$$s\left(\sum_{i=1}^{n} d_i^{\,2}-\frac{1}{n}\left(\sum_{i=1}^{n} d_i\right)^2\right)=\sum_{i=1}^{n} r_i d_i-\frac{1}{n}\sum_{i=1}^{n} r_i\sum_{i=1}^{n} d_i \;,\text{…………………(2-16)}$$

rearrange equation (2-16) to get:

$$s=\frac{\sum_{i=1}^{n} r_i d_i-\frac{1}{n}\sum_{i=1}^{n} r_i\sum_{i=1}^{n} d_i}{\sum_{i=1}^{n} d_i^2-\frac{1}{n}\left(\sum_{i=1}^{n} d_i\right)^2}\;,\text{………………………………………(2-17)}$$

multiply equation (2-17) by $\frac{n}{n}$ to get:

$$s=\frac{n\sum_{i=1}^{n} r_i d_i-\sum_{i=1}^{n} r_i\sum_{i=1}^{n} d_i}{n\sum_{i=1}^{n} d_i^2-\left(\sum_{i=1}^{n} d_i\right)^2}\;,\text{……..(2-18)}$$

also to get the offset equation rearrange equation(2-14) to get:

$$s=\frac{\sum_{i=1}^{n} r_r - on}{\sum_{i=1}^{n} d_i}\;,\text{……………………(2-19)}$$ substitute this equation in equation (2-10) to get:

$$o=\frac{\left(\sum_{i=1}^{n} d_i^2\right)\left(\sum_{i=1}^{n} r_i\right)-\left(\sum_{i=1}^{n} d_i\right)\left(\sum_{i=1}^{n} r_i d_i\right)}{n\sum_{i=1}^{n} d_i^2-\left(\sum_{i=1}^{n} d_i\right)^2}\;,\text{…………..……… ………(2-20)}$$

Where d_i are the samples values of the domain blocks.

r_i are the samples values of the range block.

n is the number of samples in each block (i.e. the block size).

2. 12 Advantage and Weakness of Fractal Compression

Fractal compression methods have been mainly studied developed for image compression. The advantages and weaknesses are apparently addressed with image compression. People generally realize that fractal compression works quite well at high compression ratio, usually around 40:1. Walle **[Wal95]** gives a very detailed analysis on fractal image encoding performance comparing with other conventional image compression methods.

2.12.1 Fractal Advantages

The most valuable advantage of fractal compression is the ability of achieving high compression ratio within certain acceptable threshold of recovery. However, the compression ratio is still highly related with identifiable patterns and self-similarities. Under this restriction, fractal compression with high compression ratio is not universally applied.

In audio compression, it can see that fractal audio compression is a much simpler scheme compared with the most popular *MP*3 encoding. This is one potential advantage that fractal compression may be used in audio world. Despite audio is a very continuous sequence; it still embeds patterns and self-similarities, especially those created by us, like music and instrumental sound, which gives us a hope that fractal compression may work well.

2.12.2 Fractal Weaknesses

Fractal compression has not been put into practical uses, even for image compression, for its numerous weaknesses. The success of the scheme seems to rely exclusively on exhibiting some self-similarities among part of the space. And there is no guarantee that the probability of matching domain and range blocks is sufficiently high to achieve compression.

The restriction of using affine mapping does not guarantee scaling α_i and offset β_i forming a set of independent random variables. This is to say that each transform w_i may not be able to be independent from others, so that w may not be equal to the union of $w_i's$.

Furthermore, the fractal encoding requests a large amount of time because of the search for matching blocks, and the fractal decoding can also be a long iterating process.

2.13 Quantization

The definition of the term "quantization" is to restrict a variable quantity to discrete values, rather than to a continuous set of values. In the field of data compression, quantization is used as follows: If the data samples to be compressed are large numbers then, quantization is used to convert them to small numbers. Small numbers take less space than large ones, so quantization generates compression. On the other hand, small numbers generally contain less information than large ones, so quantization results in lossy compression, this aspect of quantization is used by several speech compression methods.

Quantization theorem says that the quantizer can be modeled as the addition of a uniform distributed random signal (e) and the original unquantized signal (x) as shown in the figure (2.6).

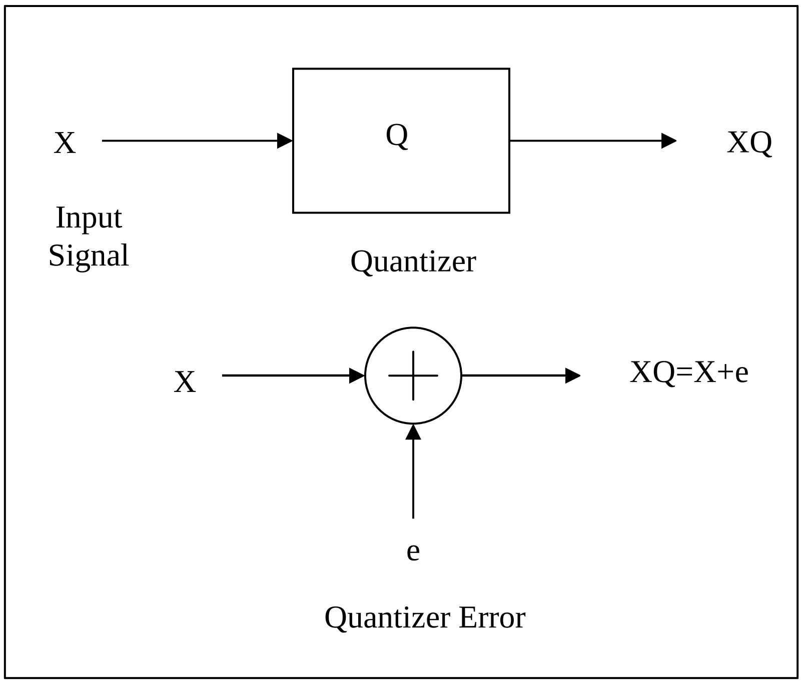


Figure (2.6) A quantizer

There are two types of quantization: Scalar Quantization and Vector Quantization. In scalar quantization, each input symbol is treated separately to produce the output, while in vector quantization the input symbols are assembled together in groups called vectors, and processed to give the output. Treating these assembles of data as a single unit could increase the optimality of the vector quantizer, but at the cost of increased computational complexity. Here, we'll take a look at scalar quantization.

A quantizer can be specified by its input partitions and output levels (also called reproduction points). If the input range is divided into levels of equal spacing then the quantizer is termed as a Uniform Quantizer, and if not it is termed as a Non-Uniform Quantizer. A uniform quantizer can be easily specified by its lower bound and the step size. Also, implementing a uniform quantizer is easier than a non-uniform quantizer. Take a look at the uniform quantizer shown in figure (2.7). If the input falls between n*r and (n+1)*r, the quantizer outputs the symbol n.

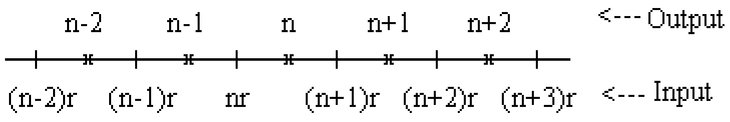


Figure (2.7) A uniform quantizer

Just in the same way a quantizer partitions its input and outputs discrete levels, a dequantizer is one which receives the output levels of a quantizer and converts them into normal data, by translating each level into a 'reproduction point' in the actual range of data. The optimum quantizer (encoder) and optimum dequantizer (decoder) must satisfy the following conditions:

1. Given the output levels or partitions of the encoder, the best decoder is the one that puts the reproduction points **x'** on the centers of mass of the partitions. This is known as *centroid condition*.
2. Given the reproduction points of the decoder, the best encoder is the one that puts the partition boundaries exactly in the middle of the reproduction points, i.e. each **x** is translated to its nearest reproduction point. This is known as nearest neighbour condition.

The quantization error **(x - x')** is used as a measure of the optimality of the quantizer and dequantizer.

2.14 Fidelity Criteria

A natural way to determine the fidelity of a recovered audio is to find the difference between the original and reconstructed values. The two popular measures of distortion are the squared error measure and the absolute difference measure, which are called *difference distortion measures*. If *X* is the source output and *Y* is the reconstructed sequence, the sum of squared error measure is given by **[Xia01]**:

$$d = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2 \quad ,\ldots\ldots\ldots\ldots(2\text{-}21)$$

And the sum of absolute difference measure is given by **[Xia01]**:

$$d = \frac{1}{n}\sum_{i=1}^{n}| x_i - y_i | \quad ,\ldots\ldots\ldots\ldots(2\text{-}22)$$

Practically, it is difficult to examine the difference on a term-by-term basis; so, some average measures are used for this examination. The most often used average measure is the average of squared error measure. This is called the *mean squared error* **(MSE)** and is given as:

$$MSE = \frac{1}{N}\sum_{i=1}^{n}(x_i - y_i)^2 \quad ,\ldots\ldots\ldots\ldots(2\text{-}23)$$

Sometimes it is more interesting to measure the size of the error relative to the peak value of the signal rather than the size of the error relative to the average squared value of the signal. This ratio is called the *peak-signal-to-noise ratio* **(PSNR)** and is calculated by the following equation:

$$PSNR(dB) = 10\log_{10}\left(\frac{255^2}{MSE}\right) ,\ldots\ldots\ldots\ldots(2\text{-}24\ )$$

PSNR is the most commonly used value to evaluate the objective audio compression quality.

CHAPTER THREE
SYSTEM DESIGN AND IMPLEMENTATION

3.1 Introduction

The encoding unit of the implemented audio fractal compression (AFC) is based on partitioned iterated function system (PIFS), which is basically based on affine transformation. So, for encoding the audio data it is necessary to divide it into non-overlapped blocks called (ranges, R), and then each block is transformed separately. By partitioning the audio data into blocks (called ranges), the partitioning will let the encoding of a wave with complicated shaped is mostly possible, taken into consideration that audio is not composed of copies and doesn't imply exact similarity, so it can't be coded as one single piece by using the IFS.

So, the PIFS is used in the suggested system to find for each range block the best approximation the best approximation is found by searching in the domain pool, compute the corresponding PIFS parameters and storing these parameters in the compression file.

The steps of the implemented algorithms for the two units of fractal audio compression system (Encoding unit and Decoding unit) are given in details in this chapter.

3.2 Audio Fractal Compression System (System Model)

The implemented fractal audio compression system consists of two major units:

1. Encoding unit
2. Decoding unit

Each of these two units consists of many modules as illustrated in figures (3.1), and (3.2).

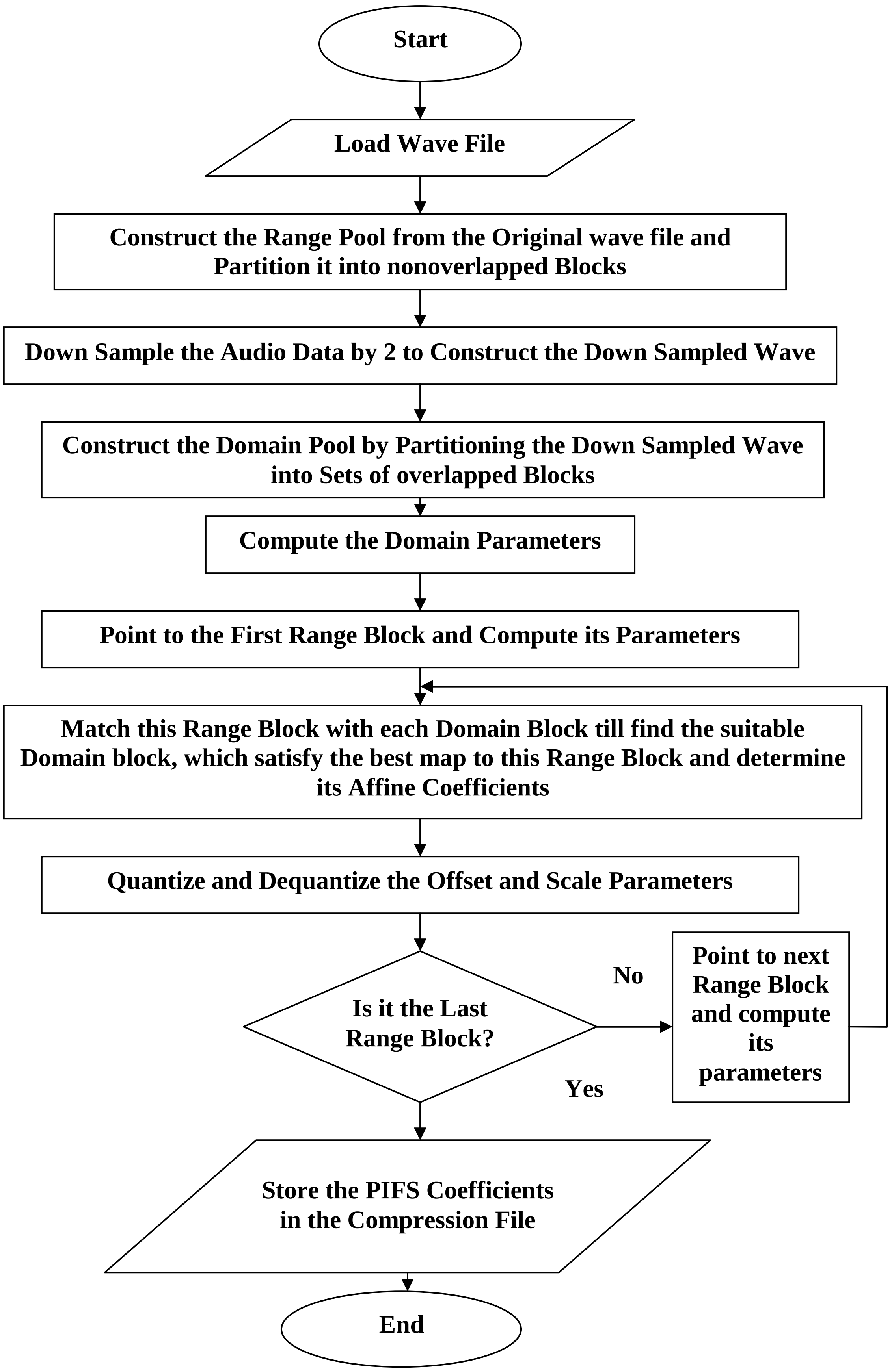

Start
Load Wave File
Construct the Range Pool from the Original wave file and Partition it into nonoverlapped Blocks
Down Sample the Audio Data by 2 to Construct the Down Sampled Wave
Construct the Domain Pool by Partitioning the Down Sampled Wave into Sets of overlapped Blocks
Compute the Domain Parameters
Point to the First Range Block and Compute its Parameters
Match this Range Block with each Domain Block till find the suitable Domain block, which satisfy the best map to this Range Block and determine its Affine Coefficients
Quantize and Dequantize the Offset and Scale Parameters
Is it the Last Range Block?
No
Point to next Range Block and compute its parameters
Yes
Store the PIFS Coefficients in the Compression File
End

Figure (3.1) The flow chart of the Fractal Audio Encoding unit

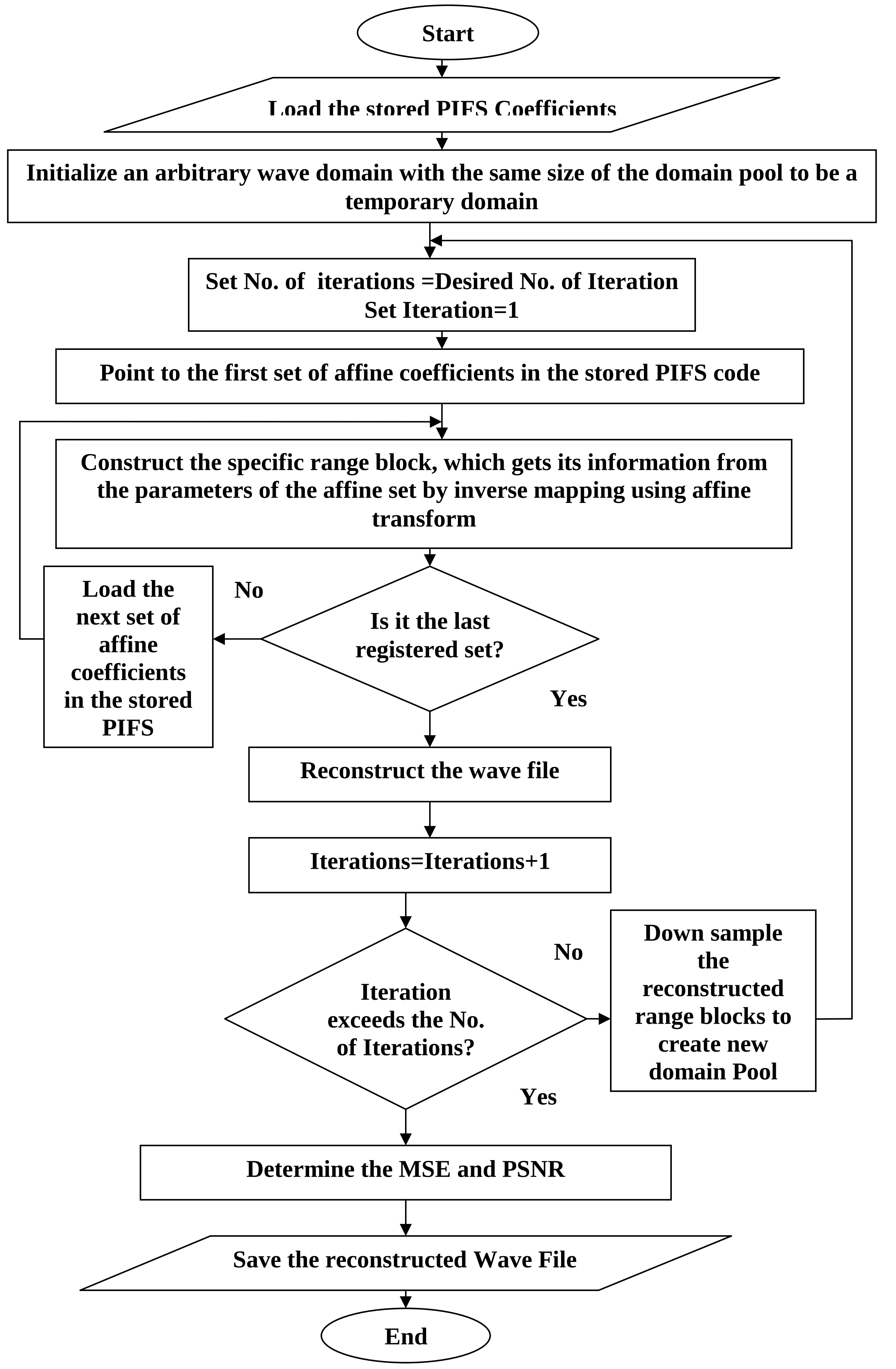

Start
Load the stored PIFS Coefficients
Initialize an arbitrary wave domain with the same size of the domain pool to be a temporary domain
Set No. of iterations =Desired No. of Iteration
Set Iteration=1
Point to the first set of affine coefficients in the stored PIFS code
Construct the specific range block, which gets its information from the parameters of the affine set by inverse mapping using affine transform
Is it the last registered set?
No
Load the next set of affine coefficients in the stored PIFS
Yes
Reconstruct the wave file
Iterations=Iterations+1
Iteration exceeds the No. of Iterations?
No
Down sample the reconstructed range blocks to create new domain Pool
Yes
Determine the MSE and PSNR
Save the reconstructed Wave File
End

Figure (3.2) The flow chart of the Fractal Audio Decoding unit

3.2.1 Encoding Unit

The encoding process is mainly based on the PIFS. This unit consists of numbers of modules, which are all together responsible for reducing the size of the desired audio data and construct the compression file.

As shown in figure (3.1), the encoding unit consists of the following modules:

1. Loading wave file.
2. Construct the Range pool: Partition the given wave data using a fixed block size partitioning to construct the Range pool *R*, and partition it into nonoverlapped blocks.
3. Down sample the audio data by 2: Take the mean value of every two successive samples values listed in the Range vector and put it in the Domain vector.
4. Construct the Domain pool: From the domain array construct the domain pool D by partitioning it into overlapped blocks with the same size of the range blocks.
5. Compute the Domain parameters (for all domain blocks) and put them in array.
6. Point to the first Range block *R* and compute its parameter.
7. Search the Domain pool, and match the first Range block with each domain block using affine transform. Find the best matched domain block, and register the corresponding affine coefficients (scale, offset, symmetry, and position), of this domain block.
8. Quantize and dequantize the scale and offset coefficients.
9. Store the determined affine transform coefficients for the current range block (which consist of the indices of S and O) in the compression stream (file).

10. If the matched range block on the range pool is the last one, then end the encoding process, if not then point to the next range block on the range pool and go to step 7.

3.2.1.1 Loading Wave File

The wave file format is a subset of Microsoft Resources Interchange File Format (RIFF) specification; it is adopted for the storage of multimedia data. This file starts out with a file header followed by a sequence of data chunks. A WAVE file is often containing single "WAVE" chunk, which consist of two sub-chunks, a format "fmt" sub-chunk specifying the data format and the "data" sub-chunk containing the actual sampling data.

Some wave data were used as test material in this research work. The specifications of the input sound waves for testing and measuring the performance of the suggested audio compression method are: 8-bits sample length, and MONO (i.e., single channel). A detailed description of the wave file format is presented in Appendix (A).

In the current work the way used for loading audio file is by loading wave (*.wav) files using the steps listed in algorithm (3.1).

Algorithm (3.1) Loading the Audio Data

```
Input: Wave File Name
Output: Wave Data Buffer Wav (WaveSize-1)
Open wave File
    • Load the header of wave file
  If Number of bits per sample = 8 and Number of Channels = 1 Then
  WaveSize = (Length of Wave file - 60) do
    • Load the audio data into a wav (WaveSize-1) buffer.
```

3.2.1.2 Range Pool Construction

There are many possible methods for partitioning the audio data that can be used to select the range blocks. The main goal of the partitioning process is to divide the audio data into regions that show similarity with other regions in the domain pool, and generate a non-overlapping region, which referred to as range pool (range blocks), that can be utilized in audio coding. It must be noted that the constraint of non-overlapping range blocks is an important condition to achieve correct decoding process.

The way of partitioning used in this research is a fixed size-partitioning scheme, because it requires less computational time than the other schemes. This done only by choosing the size of the block one time in the program.

The goal of partitioning is to improve the approximation between the samples values of range block with those of a domain block, because small blocks can probably be matched with each other better than large blocks.

Choosing the block size must be done accurately, since although the small block size perform a good matching between range and domain blocks, but this is time consuming which leads to long encoding time because of searching process. While, if the block size is big, then the encoding time is reduced but this may influence the quality of the reconstructed wave file. The test results illustrated in the next chapter will explain the effect of the block size on the compression ratio, encoding time, PSNR, and MSE.

As a first step, the header of the wave file is analyzed to get the necessary parameters (such as data size, bits per sample, sampling rate, etc.... as listed in algorithm 3.1) that required loading the audio information. After reading the audio data from the opened stream then the audio data will be partitioned uniformly, and put the partitioned data (blocks) in a temporary buffer to manipulate them as blocks of samples (of fixed size). The set of partitions is called the Range pool (Range blocks), algorithm (3.2) illustrate the implemented steps of partitioning.

Algorithm (3.2) Partitioning and Constructing the Range Pool

Input: Wave Data Buffer Wav (WaveSize-1), the size of the Wave Data Buffer WaveSize, the size of the range-block B.

Output: Wav partitioned into R_B range-blocks,(R_B is the number of range-blocks).

```
$R_B$= WaveSize/ B
For I = 1 to $R_B$
  For j = 1 to B
  $r_i$ (j)=  Wav ((I-1)* B + j)
  Next j
Next I
```

r is called the *range-block* and is of size *B*. r_i is the i'th range-block.

3.2.1.3 Down Sample the Audio Data By 2

The loaded audio data is down sampled by 2 (by averaging method) to construct the domain array.

3.2.1.4 Domain Pool Construction

This module is responsible for constructing another one-dimensional array, called the Domain, with size half the size of the range array.

The data of the domain is produced from the Range as illustrated in algorithm (3.3), there are many ways to select the data from Range to fill the Domain but all of them deal with choosing one element from every two adjacent elements in Range to be in the Domain, different selections rules were used, some of these rules are based on:

1. Choosing the minimum value of the two elements.
2. Choosing the maximum value of the two elements.

3. Choosing the average value of the two elements.
4. Choosing the nearest or farthest element to the average

The rule that is used in this research is taking the average of every two adjacent samples in range array and put it in its corresponding position in domain array.

A fixed size-partitioning scheme is used to partition the domain pool for the same purpose in the range pool partitioning. Thus the domain will be divided into “Domain blocks” with the same size of the range blocks, but possibly into overlapped domain blocks, where the partitioning jump size of samples may take values less than the block size. Overlapping blocks leads to many possible domain blocks in the domain pool, and thus good approximation will be obtained. As the jump size is small, the domain pool will be large and this satisfies the good approximation and high quality in the reconstructed wave file. But at the same time this will lead to high encoding time because searching a large pool of domain blocks is time consuming. Choosing a big jump size will serve reducing the encoding time but the reconstructed wave data will have low quality. This will be demonstrated by the results of the compression ratio, encoding time, PSNR, and the MSE in different jump sizes presented in the next chapter.

Its important to notice that the jump size must be less than or equal to the block size and of course greater than zero, and in the case of choosing the step size equal to the block size then the domain blocks will be non-overlapped.

Algorithm (3.3) Construct the Domain Pool

Error!

```
Input: Wave Data Buffer (Wav (WaveSize-1))
Output: One-dimensional array called domain pool, Domain (Domain size), with
        half size of the range pool.
  Domain Size = WaveSize \ 2 - 1
  For Pd = 0 To Domain Size do
     Pr = Pd + Pd
     Domain (Pd) = (round (Wav (Pr)) + Wav (Pr + 1)) \ 2 - 128
```

3.2.1.5 Compute the Domain Parameters

The algorithm (3.4) illustrates the implemented steps to compute the list of domain parameters.

Algorithm (3.4) Compute Domain Parameters

E

```
Input: Domain Pool array (Domain (Domain Size)), Domain Size, BlockSize and
       JumpStep.
Output: Domain parameters (SumD (No.DomainBlocks), SumD2
        (No.DomainBlocks)).
 No.DomainBlocks = (DomainSize+1-BlockSize)\JumpStep-1: Pd=0
 For I= 0 to No.DomainBlocks do
   S=0:  Ss = 0
   For Pp = Pd to Pd + BlockSize do
      S = S + Domain (Pp):  Ss = Ss + Domain (Pp) ^2
   End
   SumD (I) = S:  SumD2 (I) = Ss
```

3.2.1.6 Compute the Range Parameters

The algorithm (3.5) illustrates the implemented steps to compute the list of range parameters.

Algorithm (3.5) Compute the Range Parameters

Error!

```
Input: Wave Data Buffer (Wav (WaveSize-1)), Wave Size, Block Size.
Output: Range parameters SumR, SumR2 and RangeBlock (BlockSize-1)
 No.RangeBlocks = WaveSize \ BlockSize - 1
 Pr = 0
 For I = 0 to No.RangeBlocks do
 SumR = 0: SumR2 = 0: SumR4 = 0
   For J = 0 To BlockSize-1 do
            R = round (Wav (Pr + J)) – 128
            R2 = R * R
            SumR = SumR + R
            SumR2 = SumR2 + R2
            RangeBlock (J) = R
    End
```

3.2.1.7 Quantization and Dequantization for Scale and Offset

The actual effective part of the audio compression is quantization. It is simply the process of reducing the number of bits needed to store coefficients values by reducing its precision from float type to integer. The determined PIFS coefficients values are real-valued, and in order to increase the compression, they must converted to integer values, in order to increase the compression performance, so they must be quantized before storage.

This is done by assigning the number of bits will used to encode each scale and offset coefficients. The quantization and dequantization for scale values were computed by applying the following equations:

Quantization

$$S_I = \begin{cases} round\left(\dfrac{S}{Q_{SN}}\right) & if\ S < 0 \\ round\left(\dfrac{S}{Q_{SP}}\right) & otherwise \end{cases} \quad , \ldots\ldots \text{(3-1)}$$

Dequantization

$$S' = \begin{cases} S_I \times Q_{SN} & if\ S_I < 0 \\ S_I \times Q_{SP} & otherwise \end{cases} \quad , \ldots\ldots \text{(3-2)}$$

Where:

$$Q_{SN} = \frac{2^{bs-1} - 1}{|S_{\min}|}, \ldots\ldots \text{(3-3)}$$

$$Q_{SP} = \frac{2^{bs-1} - 1}{S_{\max}}, \ldots\ldots \text{(3-4)}$$

S is the scale coefficients.

Q_{SN} is the quantization step for negative S-values.

Q_{SP} is the quantization step for positive S-values.

S_I is the quantization index.

bs is the number of bits allocated to encode the scale coefficient.

$S_{\min}$ is the minimum allowable values for scale coefficients.

$S_{\max}$ is the maximum allowable values for the scale coefficients.

While the quantization and dequantization for offset values were computed by applying the following equations:

Quantization

$$O_I = \begin{cases} round\left(\dfrac{O}{Q_{ON}}\right) & if\ O < 0 \\ round\left(\dfrac{O}{Q_{OP}}\right) & otherwise \end{cases} \quad ,\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(3\text{-}5)$$

Dequantization

$$O' = \begin{cases} O_I \times Q_{ON} & if\ O_I < 0 \\ O_I \times Q_{OP} & otherwise \end{cases} \quad ,\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(3\text{-}6)$$

Where:

$$Q_{ON} = \frac{|O_{\min}| + |O_{\max}|}{2^{bo-1}} ,\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (3\text{-}7)$$

$$Q_{OP} = 2^{bo-1}\left(1 - \frac{|O_{\min}|}{|O_{\min}| + |O_{\max}|}\right) ,\ldots\ldots\ldots\ldots\ldots\ldots \quad (3\text{-}8)$$

O is the offset coefficients.

Q_{ON} is the quantization step for negative O-values.

Q_{OP} is the quantization step for positive O-values.

O_I is the quantization index.

bo is the number of bits allocated to encode the offset coefficient.

$O_{\min}$ is the minimum allowable values for offset coefficients.

$O_{\max}$ is the maximum allowable values for the offset coefficients.

Algorithm (3.6) Quantize and dequantize the Scale and Offset values

E

```
Input: Scale and Offset of each Range blocks.
Output: quantized and dequantized scale and offset of each range blocks.
No.RangBlocks = WaveSize \ BlockSize - 1
  For I = 0 to No. RaneBlocks do
        If Scl < 0 Then
           SI = round (Scl * StpScaleN)
           Scl = SI / StpScaleN
        Else
          SI = round (Scl * StpScaleP)
          Scl = SI / StpScaleP
       End If
       If Ofs < 0 Then
           OfsI = round (Ofs * StpOfsetN)
           Ofs = OfsI / StpOfsetN
       Else
           OfsI = round (Ofs * StpOfsetP)
```

3.2.1.8 Matching between Range Blocks and Domain Blocks

After generating range blocks and domain blocks, all range blocks should be matched with the domain blocks to determine the affine transform parameters (scale, offset, symmetry, and position) for each range block.

Now for each range block, search through all domain blocks to find the best matched domain block (block with minimum distortion error). The best matched domain block is that block whose affine transformed block has a minimum distortion error relative to other domain blocks. So, the scenario of domain search is to check each domain block and determine the scale, offset and symmetry coefficients that minimize the error between the checked domain block with the range block, the matching is continued over all the domain blocks till finding the domain block whose difference (error) with range block is the minimum in comparison with errors registered by other domain blocks. Each domain block is subjected to some isometric (symmetry) transformations (consist of reflections and rotations) to get different symmetry state for each domain block, and then the transformed domain block is considered as individual domain block, which should be matched with range blocks as a separate or individual case.

Algorithm (3.7) Searching Domain Pool and Matching

```
Input: Range blocks with number No.RangeBlocks
Output: OfsetIdx, ScaleIdx, Pos, SymState of each range block.
For I = 0 to No.RangeBlocks
    • Rb is the ith Range block.
    • Compute range parameters 'Algorithm (3.5)'.
    • Set MinError = 9.9E+19
  For j = 0 to No.DomainBlocks
        • Db is the jth Domain block.
      Pd=0: SumRD1=0: SumRD2=0
      For M=0 to BlockSize-1 do
          SumRD1=SumRD1+Domain (M+Pd)*RangeBlock (M)
```

```
For symmetry = 0 To 1
  If symmetry = 0 Then
     SumRD = SumRD1
  Else
     SumRD = SumRD2
     • Compute scale and offset values of Rb and Db using equations (2-
       18),and (2-20)
     • Quantize and dequantize Scale and Offset values of Rb using
       'Algorithm (3.6)'.
     • Compute the distortion error (E) between Rb and Db using equation
       (2-7).
 If E < MinError Then
End
Pd = Pd + JumpStep
```

3.2.1.9 Save the Affine Coefficients in the PIFS code

Saving original audio file as collections of transformations could lead to audio compression, which is done by describing the original audio in terms of few parameters of affine transformations (PIFS code).

So, the results of matching process between every range block and the domain blocks are the affine transformation parameters (scaleIdx, offsetIdx, symmetry, and position of the best matched domain block), the whole transformation informations for all range blocks would be collected in the PIFS code or (compression file) as illustrated in algorithm (3.8). In other words, fractal audio coding process implies the determination of all matching parameters, and then they are quantized, coded, and stored sequentially as arrays of PIFS parameters vectors. The elements of this array are equal to the number of range blocks in the range pool.

In addition to the affine parameters, some overhead informations are also coded and stored in compression file, these informations are important in the decoding stage, such as the minimum and the maximum boundaries of the scale and offset parameters, the number of bits used to represent the values of the scale factor, offset factor, and the position of each matched domain block, the size of the range and domain blocks, the jump step, also the audio sampling rate and the actual wave size.

Table (3.1) lists the PIFS parameters

Table (3.1) PIFS Parameters.

Parameter	**Description**
Pos	The position of the best matched domain block
ScaleIdx	The scale index value
OfsetIdx	The offset index value
SymState	The symmetry state (0:identity, 1: Reflected)

Algorithm (3.8) Saving PIFS Code

Error!

Input: Affine transform parameters

Output: Compressed file

- Prepare storage buffer.
- Encode Sample Rate and put in the storage buffer as 16 bits word.
- Encode Maximum Scale and put in the storage buffer as 11 bits word.
- Encode Minimum offset and put in the storage buffer as 11 bits word.
- Encode Maximum offset and put in the storage buffer as 11 bits word.
- Encode No. of bits per scale and put in the storage buffer as 5 bits word.
- Encode No. of bits per offset and put in the storage buffer as 5 bits word.

3.2.2 Decoding Unit

In encoding unit the encoded wave data is transformed into a set of PIFS codes. While, in the decoding unit these PIFS codes are used to iteratively reconstruct the wave data. At every iteration the decoded wave becomes closer to the original wave.

Decoding process is considerably easier and faster than the encoding process because it involves little computations. The decoding process is iterated until the fixed point is approximated, that is until further iteration does not significantly change the reconstructed wave data. Typically, 8 iterations are sufficient.

As shown in figure (3.2) this unit consists of the following items:

1. Load the stored PIFS coefficients.
2. Set Iteration =1.
3. Initialize in arbitrary manner the domain pool.
4. Point to the first set of PIFS coefficients.
5. Construct the specific range block by applying the affine transform that gets its information from the PIFS parameters set.
6. Repeat step 5 till all the range blocks are reconstructed.
7. If the iteration less than the maximum number of iterations then down sample the reconstructed range blocks to create a new domain pool, and set Iteration=Iteration+1, and go to step 4.
8. If the iteration reaches the maximum number of iterations then call MSE and PSNR subroutine (only for efficiency assessments).
9. Save the reconstructed wave data.

3.2.2.1 Load PIFS code

The first step in the decoding process is loading and decoding the affine transform parameters (ScaleIdx, OffsetIdx, Symmetry, and Position of the best matched domain block), also loading the overhead informations needed in the decoding process stage such as the minimum and the maximum boundaries of the scale and offset coefficients, the number of bits used to represent the scale factor, offset factor, and position of each matched domain block, the block size, the jump step and the actual wave size.

Algorithm (3.9) Load PIFS Code

```
    Input: compression file
    Output: Decoded affine parameters
      • Open the storage buffer.
      • Extract Sample Rate from the storage buffer as 16 bits word.
      ▪ Extract Maximum scale from the storage buffer as 11 bits word.
      ▪ Extract Minimum offset from the storage buffer as 11 bits word.
      ▪ Extract Maximum offset from the storage buffer as 11 bits word.
      ▪ Extract No. of bits per scale from the storage buffer as 5 bits word.
      ▪ Extract No. of bits per offset from the storage buffer as 5 bits word.
      ▪ Extract block size from the storage buffer as 9 bits word.
 For I = 0 to No. Range Blocks
     • Extract SymState (I) from the storage buffer.
     • Extract ScaleIdx (I) from the storage buffer.
     • Extract OfsetIdx (I) from the storage buffer.
     • Extract Pos (I) from the storage buffer.
```

3.2.2.2 Decoding Using Affine Transform Equation

The decoding process is simple, and fast. By applying the resulted PIFS on any arbitrarily generated wave the original wave at the decoder can be successively regenerated after a number of PIFS decoding iterations.

The decoder uses the affine parameters set (ScaleIdx, OfsetIdx, Position, and Symmetry) to transform the pointed (by Pos.) domain block to construct the approximate of the range block. Algorithm (3.10) illustrates the implemented steps of the decoding process.

So, the decoding phase of the affine transform involves with reconstruction of an optimal approximation for each range block by multiplying it corresponding matched domain block by the scale value and adding to the result the corresponding offset value, which is:

$$R_i = sD_i + o$$

Where:

R_i represents the value of a sample in the reconstructed (approximate) range block,

D_i represents the value of the corresponding sample in the best-matched domain block,

s represents the scale value for mapping the domain block to the range block, and

o represents the offset value.

In audio it is important to say that, the scale factor is an indication to the rate of change in the wave, while the offset factor represents wave loudness.

The reconstructed range block may transformed (reflected) according to its corresponding symmetry coefficient value, as illustrated in algorithm (3.11).

Algorithm (3.10) Decoding Equation

Error!

```
Input: affine transform parameters set (Scl, Ofs, and Pos).
Output: decoded range block DRb (BlockSize-1).
For J = 0 to BlockSize-1
        K = Scl * Domain (Pos + J) + Ofs + 128
        If K > 255 Then K = 255
        Else If K < 0 Then K = 0
        DRb (J) = K
```

E

```
Output: Reflected range block
Pr=0
For J = 0 to BlockSize-1
```

3.2.2.3 Wave Reconstruction

In this stage the domain data is initialized by setting the samples values equal to zero, and assign the number of iterations required to make the reconstructed wave close to the attractor (fixed point).

After performing the affine transform to all affine parameters sets (saved in the compression file), the produced reconstructed wave, must be used to generate a new domain pool as illustrated in algorithm (3.12). The range reconstruction process is repeated by re-applying the same affine transform sequence on the new domain pool.

This process will repeated for several times (assuming NoIter is the number of iterations), until we reach the fixed point as listed in algorithm (3.13).

Algorithm (3.12): Reconstruct the Domain

```
Input: Reconstructed range pool, DRb (Wavesize-1).
Output: Domain pool, Domain (DomSize-1).
For Pd = 0 to DomSize-1 do
  Pr = Pd + Pd
 Domain (Pd) = (round (DRb (Pr)) + DRb (Pr + 1)) \ 2 – 128
```

Algorithm (3.13) Reconstruct the original wave

```
Input: number of range blocks (No.RangeBlocks).
Output: reconstructed wave DRb (WaveSize-1).
For I = 0 to DomSize-1
    Domain (I) = 0
End
```

CHAPTER FOUR

Performance Measures and Test Results

4.1 Performance Measures

Some performance measures were taken into consideration to evaluate the performance efficiency of the suggested fractal audio compression system.

The adopted measures are the fidelity criteria (i.e., MSE, and PSNR), the compression ratio, and entropy measures.

4.1.1 Compression Ratio

The ratio of the original (uncompressed audio file) and the compressed audio file is referred to as the *Compression Ratio*, (i.e. the term compression ratio is used to refer to the ratio of uncompressed data to compressed data). The compression ratio is denoted by [**Umb98**]:

$$Compression\,Ratio = \frac{uncompressed\ file\ size}{compressed\ file\ size}$$

$$= \frac{size_u}{size_c}, \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4-1)$$

And it is often written as $size_u : size_c$

Thus an audio with a 10:1 compression ratio has a compressed data size 10 times smaller than the original audio file [**Mar98**].

The uncompressed audio file size is computed from the following equation:

$$size_u\,(Bit) = S_{Audio} \times 8, \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (4-2)$$

Where:

S is the size of the original audio file, and 8 is the number of bits required to assign the each sample value

While the compressed file size is computed from:

$$size_c\,(Bit) = (BitScl + BitOfs + BitPos + 1) \times No.RangeBlock, \ldots\ldots (4\text{-}3)$$

Where:

BitScl is the number of bits required to store the scaling values,

BitOfs is the number of bits required to store the offset values,

BitPos is the number of bits required to store the position of the best matched domain block.

The number 1 is the bits required to store the symmetry value (which is the reflection state), and *No. Range Blocks* is the number of the range blocks in the original audio file.

4.1.2 Entropy Measures

Entropy, which is a measure of the inherent randomness in a probability distribution (or set of observed data). And it can be computed using the equations:

$$Entropy = -\sum_{i=0}^{255} p * \log_2(p), \ldots\ldots (4\text{-} 4)$$

$$p = \frac{His(i)}{n}, \ldots\ldots (4\text{-} 5)$$

Where:

n is the number of samples.

4.1.3 Energy Measures

A term encounter frequently when measuring sound is the RMS, or the root mean square, value. The RMS value is a special kind of mathematical average value, which is directly related to the energy contents of the sound. The energy content of the sound computed from the relation:

$$Energy = 1/n \sum_{k=1}^{n} (wav(k) - 128)^2 \text{ ,……………………….…………(4 - 6)}$$

Where:

n is the number of samples.

4.2 Performance Parameters

Several parameters were taken into consideration to study the performance of the suggested fractal audio compression system. The considered control parameters are: the block size, jump size, quantization steps for both scale and offset, maximum and minimum values for both scale and offset.

4.3 Audio Test Samples

Table (4.1) demonstrates the attributes of five audio test files. Figures (4.1) to (4.5) present the waveform of the adopted five test samples.

All these five test samples are Wave Sound type with 8 bits sample size, PCM format, and 1(mono) which is the number of channels.

Table (4.1) The Attributes of the audio test samples

Name	**Test Sample1**	**Test Sample2**	**Test Sample3**	**Test Sample4**	**Test Sample5**
Size	332 KB	82.4 KB	235 KB	70.3 KB	74.9 KB
Sampling Rate	22kHz	22kHz	22kHz	11kHz	11kHz
Behavior	Music	Music	Music	Speech	Speech

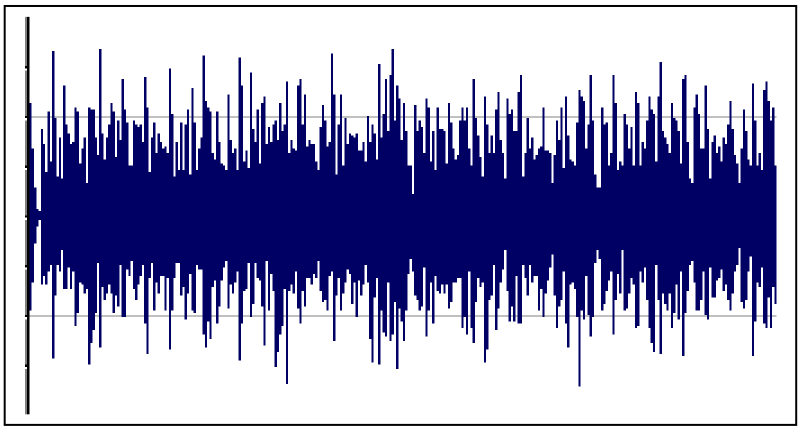


Figure (4.1) The waveform of the test sample 1

(Entropy= 6.51: Energy= 496.61)

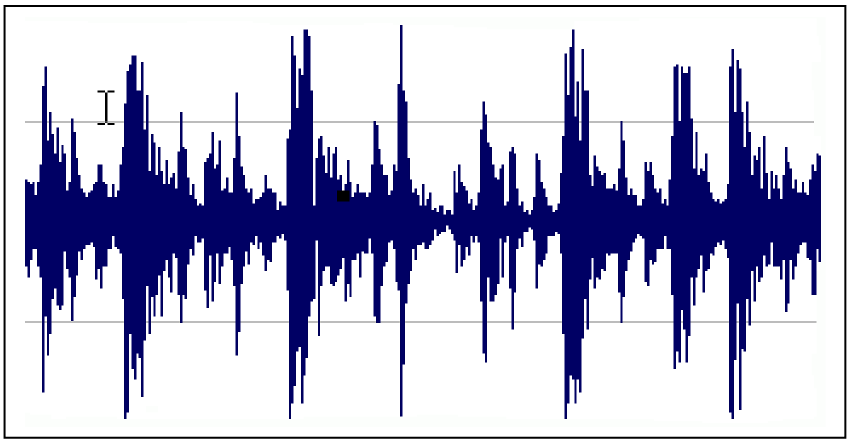


Figure (4.2) The waveform of the test sample 2

(Entropy= 6.3: Energy= 550.52)

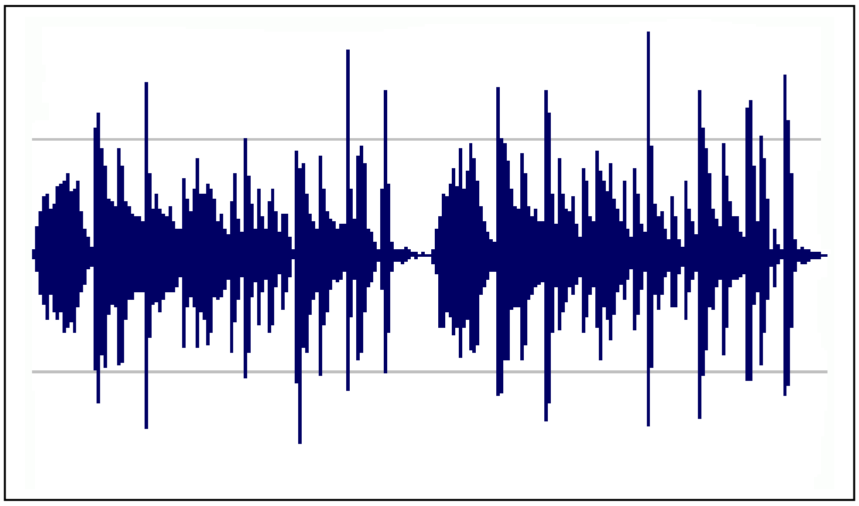


Figure (4.3) The waveform of the test sample 3

(Entropy= 5.27: Energy=130.22)

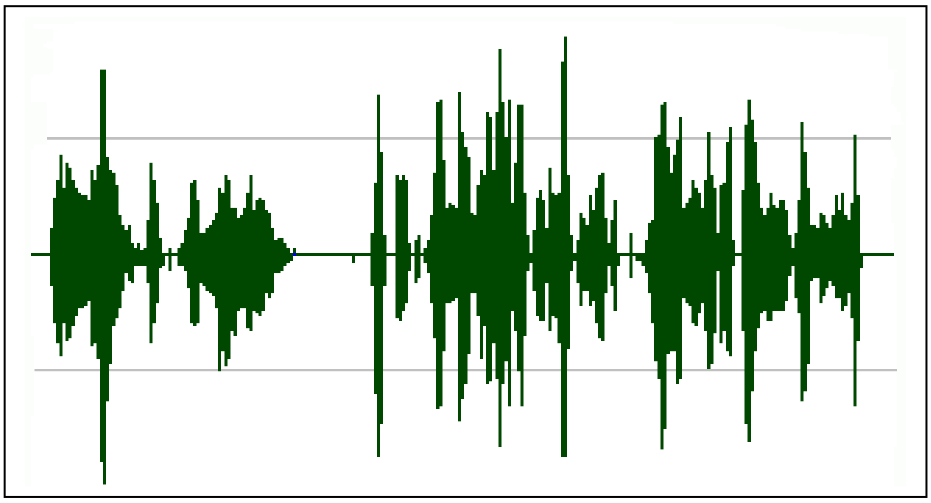


Figure (4.4) The waveform of the test sample 4

(Entropy= 5.2: Energy=358.59)

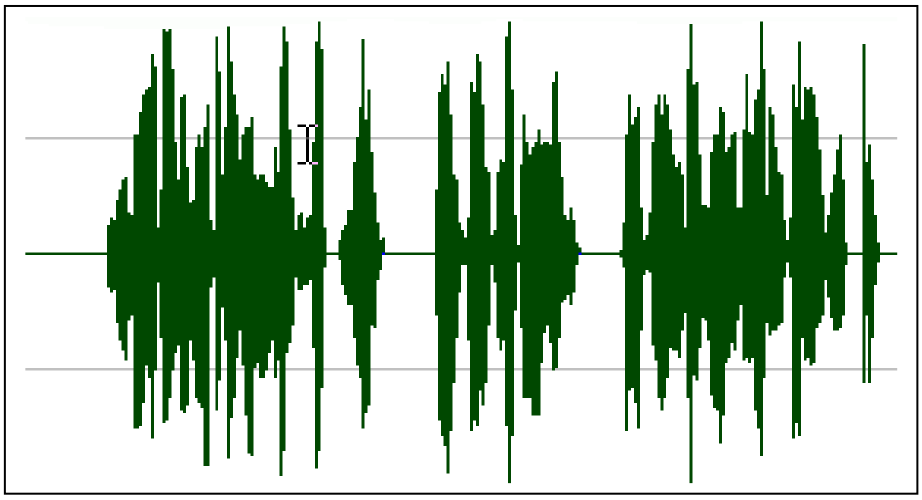


Figure (4.5) The waveform of the test sample 5

(Entropy= 5.36: Energy=563.91)

4.4 Test Results

In this section, the five test audio files were tested for examining the performance of the proposed fractal audio compression system; in these tests the effects of the control parameters on the performance of the compression system were investigated as follows:

Test (1): Block Size Effect

The effect of block size in this test for case (test sample-4) is investigated. The other compression parameters were taken as in table (4.2).

Table (4.2) Coding parameters

Maximum Scale	**1.5**
Minimum scale	**-1.5**
Maximum Offset	**128**
Minimum Offset	**-128**
Scale Bits	**8**
Offset Bits	**8**

Different values for the block size were taken, and the results of applying the compression system of (test sample-4) are listed in table (4.3). The tests results of applying same test on other samples have shown same behavior.

Table (4.3) The Resulted MSE and PSNR of the reconstructed Wave File

Block Size	**MSE**	**PSNR**	**Compression Ratio**	**Encoding Time (sec.)**
4	0.063	60.11	1.4:1	1151.77
14	2.21	44.68	4.68:1	484.48
18	5.26	40.92	5.95:1	432.44
24	25.07	34.13	7.83:1	393.62
30	51.58	31.0	9.7:1	346.62

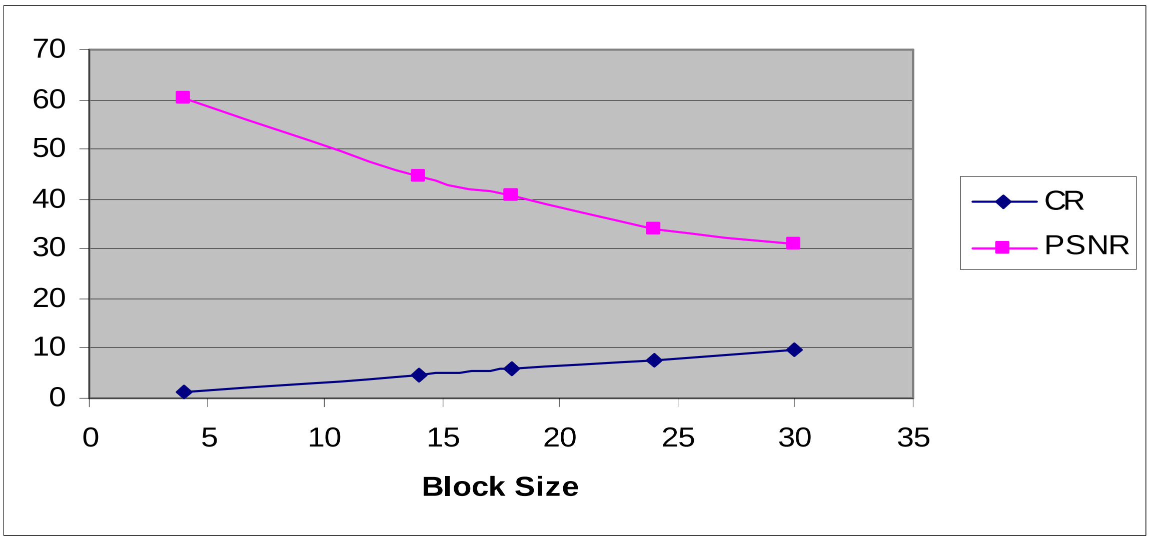


Figure (4.6) The effect of different block sizes on the compression ratio and the PSNR of the reconstructed wave data.

It's clear from this figure that the following points can be concluded:

1. By increasing the block size the compression ratio increases.
2. By increasing the block size the quality of the reconstructed wave is negatively affected, i.e. the value of the PSNR becomes lesser and the value of the MSE becomes larger.
3. The encoding time decreases with increase of the block size.

Test (2): Quantization Effect

In this test (test sample-4) was taken to demonstrate the effect of quantization steps (for both scale and offset) on the compression performance.

Different values of both (Scale bits and Offset bits) were taken.
Values of other coding parameters were taken fixed as in table (4.4).

Table (4.4) coding parameters

Maximum Scale	**1.5**
Minimum scale	**-1.5**
Maximum Offset	**128**
Minimum Offset	**-128**
Block Size	**14**
Jump Size	**1**

The test results of the quantization steps, for the case (test sample-4); on the MSE and the PSNR of the reconstructed wave data are listed in table (4.5).

Table (4.5) The Resulted MSE and PSNR of The Reconstructed Wave File

Scale Bits	**Offset Bits**	**MSE**	**PSNR (dB)**	**Compression Ratio**	**Encoding Time (sec.)**
5	5	353.67	22.64	5.66:1	557.78
6	6	421.16	21.88	5.32:1	547.65
7	7	26.23	33.94	4.48:1	546.75
8	8	2.21	44.68	4.68:1	484.48
5	7	20.95	34.91	5.32:1	543.60
6	7	23.73	34.37	5.13:1	545.12
5	8	2.66	43.87	5.13:1	543.37

Notes:

1. When the number of bits used to represent the scale value (Scale Bits) is equal to 5, and the number of bits used to represent the offset value (Offset Bits) is equal to 5 also, the MSE value is too large, so to decrease the MSE value the values of the Scale Bits and Offset Bits were increased, so when the values of Scale Bits and Offset Bits increased to 6 the PSNR value decreases.
2. So increasing the values of the Scale Bits and Offset Bits will lead to make the value of the MSE become well.
3. It’s noticeable that by decreasing the value of the Scale Bits and at the same time increasing the value of the Offset Bits the PSNR value would be better.
4. The best values for both Scale Bits and Offset Bits led to acceptable values of MSE, PSNR, and compression ratio is when Scale Bits equal 5 and Offset Bits equal 8, therefore in the next

(other) tests Scale Bits and Offset Bits are taken 5 and 8, respectively.

5. By increasing the value of the offset Bits the PSNR value increases rather than the Scale Bits, that's means that the value of the Offset Bits has effect on the audio quality more than Scale Bits.

Test (3): Jump Size Effect

In this test the values of other coding parameters were taken fixed as in table (4.6).

Table (4.6) coding parameters

Maximum Scale	**1.5**
Minimum scale	**-1.5**
Maximum Offset	**128**
Minimum Offset	**-128**
Scale Bits	**5**
Offset Bits	**8**
Block Size	**4**

To demonstrate the effect of the Jump Size on the MSE and PSNR values and the encoding time. Different values for the jump size were considered.

The test results for the case (test sample-5) are listed in table (4.7).

Table (4.7) The Resulted MSE and PSNR of the reconstructed wave data (Test-Sample-5)

Jump Size	MSE	PSNR (dB)	Compression Ratio	Encoding Time (sec.)
4	0.13	56.96	1.6:1	392.01
10	0.13	56.78	1.71:1	174.37
16	0.92	48.45	1.7:1	118.7

20	0.96	48.35	1.78:1	96.23
26	0.99	48.17	1.78:1	82.34
36	19.95	35.13	1.78:1	69.52

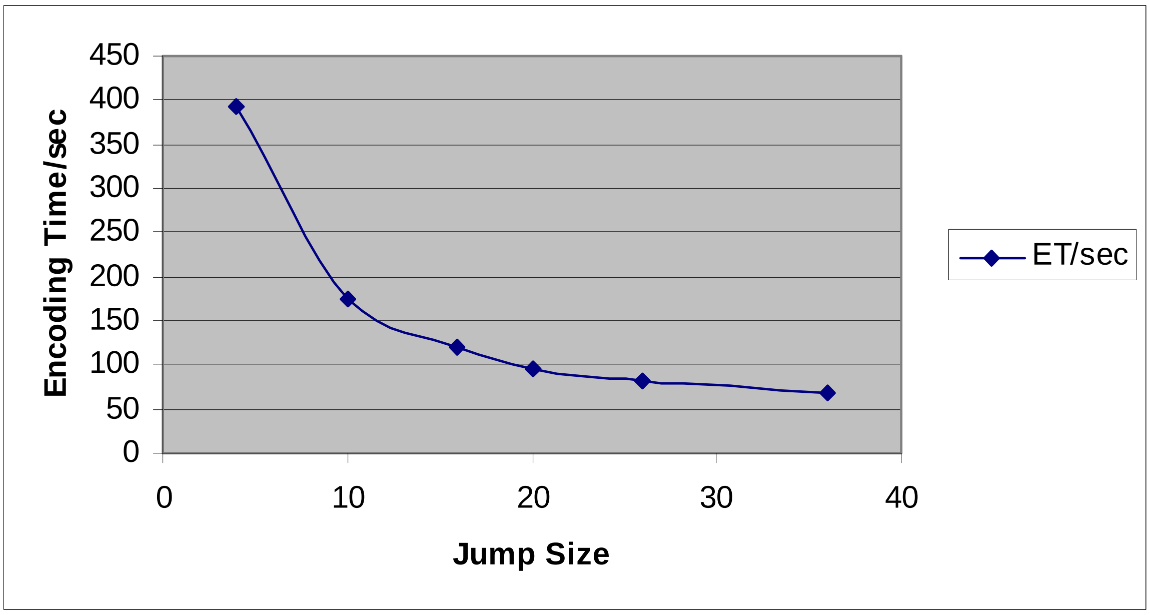


Figure (4.7) The Effect of the jump size on the encoding time

Notes:

1. By fixing the block size and varying the jump size the encoding time will be highly affected.
2. By increasing the jump size the encoding time will be decrease, because the increase in jump size will lead to small domain pool, so the searching process for the best match domain block will need smaller number of tests.
3. Increasing the jump size will affect the audio quality of the reconstructed file, as shown in the table (4.7) the value of PSNR decreases when jump size increases.
4. Increasing jump size rather has no effect on the compression ratio.

Test (4): Maximum Scale and Minimum Scale Effects

In this test three test samples were taken as test materials. The first one is (test sample-4). The values of other coding parameters were taken fixed as in table (4.8).

Table (4.8) Coding parameters

Block size	**20**
Jump Size	**8**
Maximum Offset	**128**
Minimum Offset	**-128**
Scale Bits	**5**
Offset Bits	**8**

To demonstrate the effect of different maximum scale and minimum scale values on the **PSNR** and **MSE** of the reconstructed wave data. Different values for maximum and minimum scale were considered.

The test results for the case (test sample- 4) are listed in table (4.9).

Table (4.9) The MSE and PSNR of the reconstructed wave data for the first test set.

Min. Scale	**Max. Scale**	**MSE**	**PSNR (dB)**	**Compression Ratio**
-1.5	1.5	20.91	34.92	7.92:1
-2	2	15.22	36.30	7.92:1
-3	3	14.45	36.53	7.92:1
-4	4	14.05	36.65	7.92:1
-5	5	14.93	36.38	7.92:1
-8	8	17.60	35.67	7.92:1

The second test is applied on (test sample -5) with the following fixed coding parameters as in table (4.10).

Table (4.10) Coding parameters

Block size	**20**
Jump Size	**8**
Maximum Offset	**128**
Minimum Offset	**-128**
Scale Bits	**5**
Offset Bits	**8**

The test results of the second test are listed in table (4.11).

Table (4.11) The MSE and PSNR of the reconstructed wave data for the second test set.

Min. Scale	**Max. Scale**	**MSE**	**PSNR**	**Compression Ratio**
-1.5	1.5	69.90	29.68	7.89:1
-2	2	61.04	30.27	7.89:1
-3	3	48.23	31.3	7.89:1
-4	4	45.24	31.57	7.89:1
-5	5	43.68	31.72	7.89:1
-8	8	45.22	31.57	7.89:1

The third test is applied on (test sample 2) with the following fixed coding parameters as in table (4.12).

Table(4.12) Coding parameters

Block size	**10**
Jump Size	**10**
Maximum Offset	**128**
Minimum Offset	**-128**
Scale Bits	**5**
Offset Bits	**8**

The test results of the third test are listed in table (4.13).

Table (4.13) The MSE and PSNR of the reconstructed wave data for the third test set.

Min. Scale	**Max. Scale**	**MSE**	**PSNR**	**Compression Ratio**
-1.5	1.5	28.92	33.51	2.87:1
-2	2	37.00	32.44	2.87:1
-3	3	20.18	35.08	2.87:1
-4	4	17.77	35.63	2.87:1
-5	5	7.94	39.13	2.87:1
-8	8	18.27	35.51	2.87:1

Notes:

1. The results of the test set (first) indicate that the scale boundary has an effect on the **MSE**. In other words the increase in value of Min. scale (up to 4) will decrease the value of MSE. But the further increase in value of Min.scale will cause slight increase in MSE.

2. The results of the test set (second) indicate that the value of the MSE decreases by increasing the Min.scale (up to 5) further increase in value of Min.scale will increase the value of the MSE.
3. The results of the test set (third) indicate that the value of the MSE decreases by increasing the Min.scale (up to 5) further increase in Min.scale value will increase MSE value.

Test (5): Maximum Offset and Minimum Offset Effects

In this test set two test samples were taken: the first part of this test set is applied on (test sample-4). In this test the following coding parameters are taken fixed as in table (4.14).

Table (4.14) Coding parameters

Block size	**20**
Jump Size	**8**
Maximum Scale	**4**
Minimum Scale	**-4**
Scale Bits	**5**
Offset Bits	**8**

To demonstrate the effect of maximum offset and minimum offset values on the **PSNR** and **MSE** of the reconstructed wave data. Different values for maximum and minimum offset were considered.

The test results of (test sample 4) are listed in table (4.15).

Table (4.15) The resulted MSE and PSNR of the reconstructed wave data for (test sample- 4)

Min. Offset	**Max.Offset**	**MSE**	**PSNR**	**Compression Ratio**
-128	128	14.05	36.65	7.92:1
-256	256	13.95	36.68	7.92:1
-512	512	16.21	36.03	7.92:1

The second part of this test set is applied on (test sample-2). In this test the following coding parameters are taken fixed as in table (4.16).

Table (4.16) Coding parameters

Block size	**40**
Jump Size	**40**
Maximum Scale	**5**
Minimum Scale	**-5**
Scale Bits	**5**
Offset Bits	**8**

The test results of (test sample -2) are listed in table (4.17).

Table (4.17) The resulted MSE and PSNR of the reconstructed wave data for (test sample-2)

Min. Offset	**Max. Offset**	**MSE**	**PSNR**	**Compression Ratio**
-128	128	45.66	31.53	12.3:1
-256	256	726.8	19.51	12.3:1
-512	512	728.3	19.50	12.3:1

Notes:

1. In the first part of this test set, taking the offset value 128, 256 or 512 has no real effect on PSNR value it's rather stay the same, according to the PSNR values obtained.
2. In the second part of the test set the change may appear larger when taking Max.Offset equal to 256 and 512, where PSNR value decrease. But when Max.Offset value equal to 128 PSNR value will be larger than that of 256 and 512, according to the PSNR values obtained.

Further tests made on all five test samples, the resulted MSE and PSNR of the reconstructed wave data are listed in table (4.18).

Table (4.18) The resulted MSE and PSNR and compression ratio of the reconstructed wave data

Test Samples	Block Size	Jump Size	MSE	PSNR	Compression Ratio	Encoding Time (sec.)
Sample1	40	40	12.42	37.18	11.44:1	308.08
Sample2	40	40	44.28	31.61	12.3:1	22.95
Sample3	40	40	32.38	33.02	12.91:1	138.71
Sample1	10	10	1.77	45.63	2.7:1	2284.36
Sample2	10	10	28.92	33.51	2.87:1	149.6
Sample3	10	10	6.41	40.05	3.06:1	1014.61
Sample1	60	4	16.98	35.83	15.51:1	2317.38
Sample2	40	4	37.72	32.36	11.03:1	162.85
Sample3	20	4	17.22	35.76	5.86:1	1610.09
Sample4	10	1	1.09	47.72	3.7:1	621.39
Sample5	10	1	3.75	42.38	3.65:1	820.38
Sample4	14	4	5.07	41.08	5.5:1	121.45
Sample5	14	4	18.97	35.35	5.42:1	160.50
Sample4	20	8	20.91	34.92	7.92:1	54.71
Sample5	20	8	69.90	29.68	7.89:1	62

Table (4.19) Test samples and corresponding PSNR and energy

Test Samples	Block Size	PSNR	Energy
Sample3	40	34.02	130.22
Sample4	40	33.92	358.59
Sample1	40	31.18	496.61
Sample2	20	30.61	550.52
Sample5	20	29.68	563.91

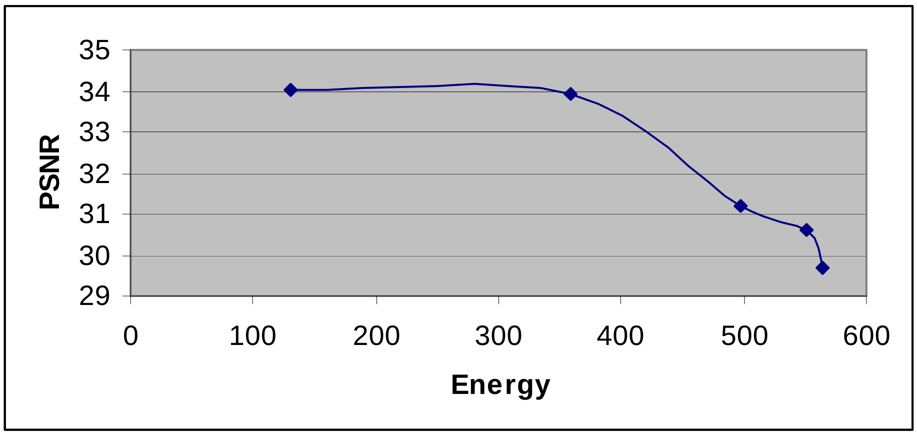


Figure (4.8) The relationship between PSNR and energy

Notes:

1. This figure is to study the behavior of the test samples sound quality with its corresponding energy for each test sample.
2. The figure (4.8) shows that the PSNR value decreases when the energy increases.

Table (4.20) Test samples and corresponding PSNR and entropy

Test Samples	Block Size	PSNR	Entropy
Sample4	40	33.92	5.2
Sample3	40	34.02	5.27
Sample5	40	29.68	5.36
Sample2	20	30.61	6.3
Sample1	20	31.18	6.51

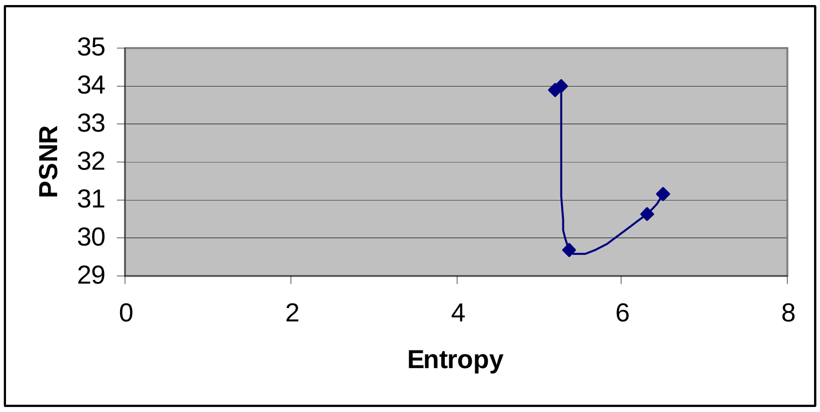


Figure (4.9) The relationship between PSNR and entropy

CHAPTER FIVE

Conclusions and Future Work Suggestions

5.1 Conclusions

In this work an attempt is made to design and implement a fractal audio compression system.

From the test results presented in the previous chapter, some remarks related to the behavior and performance of the suggested fractal audio compression system were concluded, among these remarks are the following:

1. The encoding time is inversely proportional with both Block size and Jump size.
2. The compression ratio and the MSE value of the fractal audio compression system are direct proportional with both block size and jump size, while the PSNR value is inversely proportional with both block size and jump size.
3. As the jump size is small, the domain pool is large, so a better quality of the reconstructed audio will obtain.
4. The IFS coefficients (Scale and Offset) highly affect the compression ratio and it was improved when they are quantized. But these coefficients do not have any effect on the encoding time.
5. Fractal method can provide good compression performance for sounds.
6. In this work, the implemented fractal audio compression method has very long encoding time. This can considered as the main weak point in fractal compression method.
7. The long time process implied in the encoder is resulted from the matching module, where for each range block the searching process is trying to find the domain block, which satisfy the best match with the considered range block among the other whole domain blocks taking into

consideration the symmetry states. So this searching with its matching, transformation, will lead to long encoding time for fractal method.

5.2 Future Work Suggestions

The followings are recommendations for the future work:

1. Since the resulted IFS code consists of only the parameter vectors, then it is possible to use additional lossless data compression method to further compress the PIFS code and obtain better compression performance such as Huffman coding method.
2. Trying to use other audio partitioning scheme (variable block size), which may cause a better quality for the reconstructed audio.
3. using some classification methods to classify the domain blocks and the range blocks.
4. Develop the software system to open the coded file directly (decode it and play it at the same time).
5. In order to reduce the long encoding time of fractal compression and make it reasonable. We suggest an approach depends on the idea of reducing the matching search operation by suggesting new searching mechanism.
6. Elimination of the unvoiced data samples from the original input data to decrease the compression time.

Chapter Four Performance Measures and Test Results

Chapter One General Introduction

Chapter Three System Design and Implementation

Chapter Two
Basic Concepts of Fractal

Chapter Five Conclusions and Future Work Suggestions

Dedication

I dedicate my work to all the researchers and scientists who use the science to make the world a better place.

To all the people who sacrifice in their lives for a better future for their country and for their children.

To my country as a simple gift, to memory of my best friend (Oras), to my family.

Wesam

الاسم: وسام فوزي جاسم محمد
القسم: علوم الفيزياء
الموضوع: Fractal Audio Compression
التاريخ: ٢٠٠٥/٨/١
التقدير: Very Good
العنوان: بغداد ، الشعلة
الهاتف: ٥١٧٠٣٨٧
الموبايل: ٠٧٩٠١٩٦٩٢٧٠

بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

وَيَسۡـَٔلُونَكَ عَنِ ٱلرُّوحِۖ قُلِ ٱلرُّوحُ مِنۡ أَمۡرِ رَبِّي

وَمَآ أُوتِيتُم مِّنَ ٱلۡعِلۡمِ إِلَّا قَلِيلٗا ﴿٨٥﴾

صدق الله العلي العظيم

الأسراء-٨٥

List of Abbreviations

Abbreviation	**Meaning**
ADC	Analog-to-Digital Converter
ASCII	An acronym for American Standard Code for Information Interchange
AFC	Audio Fractal Compression
CD-ROM	Compact Disk – Read Only Memory
CR	Compression Ratio
dB	Decibel
FIC	Fractal Image Compression
GSM	Global System for Mobile Telecommunication Protocol
HV	Horizontal-Vertical Partition
IFS	Iterated Function System
JPEG	Joint Photographic Expert Group
MP3	MPEG Audio Layer 3
MPEG	Moving Picture Expert Group
MSE	Mean Squared Error
PCM	Pulse Code Modulation
PIFS	Partitioned Iterated Function System
PSNR	Point to Point Signal to Noise Ratio
RIFF	Resource Interchange File Format
SNR	Signal- to- Noise Ratio
WT	Wavelet -Transform

References

[ATS01]

ATSC Standard: Digital Audio Compression (AC-3), Revision A, Advanced Television Systems Committee, Doc. A/52A, 20 August 2001.

[Aba02]

Abas M.; “Genetic Algorithm and Vector Quantization for Image Compression”, Ph.D. Thesis, Computer Science and Information System dep., Baghdad University, 2002.

[Bar93]

Barnsley M. F. and Hurd L. P.; “Fractal Image Compression”, AK peters, Wellesley, Massachusetts, USA, 1993.

[Bar88]

Barnsley M. F.; “Fractals everywhere”. New York: Academic Press, 1988.

[Bah95]

Baharav Z., Krupnik H., and karnin E.; “A Multi-Resolution Framework for Fractal Image Representation and its application”, Technion-Israel Institute of Technology, Israel, 1995.

[Deg95]

Degener J.; “Speech Compression Algorithm to Support the Real-Time Video Conferencing Research”, A research, Technical University of Berlin, 1995.

[Fis94]

Fisher E. Y.; “Fractal Image Compression: Theory and application”. Springer Verlag, 1994.

[Fis92]

Fisher E. Y.; “Fractal Image Compression”, course notes, Volume 12, ACM SiGGRAPH, 1992.

[Jac92]

Jacquin A. E.; “Image Coding Based on a Fractal Theory of Iterated Contractive Image Transformations”, IEEE Transactions on Image Processing, Vol. 1, no. 1, January 1992.

[Kis99]

Kishimoto N. and Natori N. C.; “Basic Consideration of Structures with Fractal Properties and Their Mechanical Characteristics”, Paper, The institute of Space and Astronautically Science, Kanagawa, Japan, 1999.

[Kie98]

Kientzle T.; “A Programmer’s Guide to Sound”, Addison-Wesely Developers press, 1998.

[Klo00]

Klonowski W., “Signal and Image Analysis Using Chaos Theory and Fractal”, Polish Academy of Sciences, Poland, 2000.

[Man83]

Mandelbrot B. B.; “the fractal geometry of nature", Freeman W. H. and company, 1983.

[Man77]

Mandelbrot B. B. “The Fractal Geometry of Nature”, Freeman W. H. and Company, 1977.

[Mor98]

Morris J.; “Center for Intelligent Information Processing Systems”, University of Western Australia, 1998

[Mar98]

Martensson A.; “Error Resilient Wavelet Image Compression”, Stockholm University, 1998.

[Pan93]

Pan, D. Y.; “Digital Audio Compression” Digital Technical Journal, Vol. 5 No. 2, Spring 1993.

[Sal00]

Salomon, D.; “Data Compression the Complete Reference”, Addison Wesley Company, Second Edition, 2000.

[Sha96]

Shamoon T. G.; “Algorithms for encoding high fidelity audio at low bit rate”, Cornell University, 1996.

[Sin93]

Sinha D., and Tewfik A. H.; “Low Bit Rate Transparent Audio Compression Using a Dynamic Dictionary an Optimized Wavelets”, Dept. of Electr. Eng. Minnesota Univ., IEEE International Conference on Acoustics, Speech, and signal processing, pp.197-200, 1993.

[Sha01]

Shannon C. E.; “a Mathematical Theory of Communication”, Bell Labs, 2001.

[Umb98]

Umbaugh S. E.; “Computer Vision and Image Processing”, Prentice-hall, Inc., USA, 1998.

[Wat95]

Watkinson J.; “Compression in video and audio”, Reed Educational Company, 1995.

[Wan00]

Wang A.; “Data Coding”, Information and Computer Science Dept., University of California Irivin, 2000. Site:
http:www.cs.tut.fil/~ypsilon/80545/coding of As.html#HRD2, 2000.

[Woh99]

Wohlberg B. and Jager G., “A Review of the Fractal Image Coding Literature”, IEEE Transactions on Image Processing, Vol. 8, no. 12, December 1999.

[Wal95]

Walle A. V.; “Relating Fractal Image Compression to Transform Methods”, M.Sc Thesis, College of Science, University of Waterloo, 1995.

[Xia01]

Xiao P.; “Image Compression by Wavelet Transform”, M.Sc. Thesis, college of science, East Tennessee State University, 2001

[Xia04]

Xiao H.; “Fractal Compression”, Queen’s University, Kingston, Ontario, Canada, April 2004. Site:
http://www.cs.queensu.ca/home/xiao/doc/fractal.pdf

[Yok03]

Yokoyama T., Watanab T., and Sugawara E.; “Similarity-based Image Retrieval System Using PIFS Codes”, University of Electro–Communications, Tokyo, JAPAN, 2003.

[Zol98]

Zolzer U.; “Digital Audio Signal Processing”, Wiley Company, 1998.

Table of Contents

Republic of Iraq
Al-Nahrain University
College of Science

Fractal Audio Compression

**A THESIS
SUBMITTED TO THE
COLLEGE OF SCIENCE, Al-NAHRAIN UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE IN
PHYSICS**

By

Wesam Fawzi Jassim Mohammed

(B.Sc. 2002)

SUPERVISORS

Dr. Loay A. George **Dr.Laith A. Al-ani**

June 2005 **Jamadi Al-awal 1425**