Abstract

There are two main alternatives to compress a video. The first one, usually called intraframe approach, pretends to remove the spatial redundancy of an image without destroying important information. These methods are suitable for still image applications such as multimedia, image database, etc. Nevertheless, in applications that use a sequence of image, data such as TV scenes, video conferencing etc, time redundancy can be exploited to increase the compression ratio since consecutive frames are usually highly correlated. This second group of methods is called interframe approach, and pretends to remove temporal redundancy. In the second approach, motion estimation of sequence frames must be computed. In the proposed work, the interframe approach is implemented.

In the field of motion estimation for video compression many techniques have been applied. Block-based motion estimation approaches are the most common procedures applied using various algorithms. The full search algorithm (FSA) provides the best performance but at very expensive computational cost. To reduce this computational requirement, fast search algorithms have been developed, among them being the conventional three-step algorithm (TSA). In the proposed work OTS, and TSS methods of ME are implemented in addition to a new developed Hybrid Method (HM).

The interframe approach select a number of frames that will compress using compression system that is different than ME techniques, these frames are called Anchor frames (AF). In the proposed work there are two models that developed for video coding, the first one develop a compression system that depend on FDCT transform that a new derivation of DCT, where this transform is speed up through a new derivation that fully documented in the proposed work, and the second model develop the Fractal coding as compression system for AF. The disadvantage of Fractal coding is the expensive time that Fractal needs to complete its search. This problem is solved in the proposed work through fasting Fractal Search using distributed system that divide the Fractal search on the total number of Servers that shared on the network. The proposed work is implemented using Visual Basic 6.0 as a programming language. The fidelity measure MSE and PSNR are used to check the result of the whole developed techniques.

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 "InterFrame Compression Using Distributed Systems" And perform the points that I felt were important.

Also I wish to thank, Dr. Venus W. Samawi, my supervisor for her valuable advice and encouragement. Grateful thanks for the Head of Department of Computer Science Dr. Taha S. Bashaga.

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

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

AVI File Format

AVI Files

The Microsoft Audio/Video Interleaved (AVI) file format is a RIFF file specification used with applications that capture, edit, and playback audio/video sequences. In general, AVI files contain multiple streams of different types of data. Most AVI sequences will use both audio and video streams. A simple variation for an AVI sequence uses video data and does not require an audio stream. Specialized AVI sequences might include a control track or MIDI track as an additional data stream. The control track could control external devices such as an MCI videodisc player. The MIDI track could play background music for the sequence. While a specialized sequence requires a specialized control program to take advantage of all its capabilities, applications that can read and play AVI sequences can still read and play an AVI sequence in a specialized file. (These applications ignore the non-AVI data in the specialized file.) This chapter primarily describes AVI files containing only audio and video data. This chapter covers the following topics:

- The required chunks of an AVI file
- The optional chunks of an AVI file
- Developing routines to write AVI files

AVI RIFF Form

AVI files use the AVI RIFF form. The AVI RIFF form is identified by the four-character code “AVI”. All AVI files include two mandatory LIST chunks. These chunks define the format of the streams and stream data. AVI files might also include an index chunk. This optional chunk specifies the location of data chunks within the file. An AVI file with these components has the following form:

```
RIFF ('AVI'
LIST ('hdrl'
.
.
.
)
LIST ('movi'
.
.
.
)
['idx1'<AVI Index>]
)
```

The LIST chunks and the index chunk are subchunks of the RIFF “AVI” chunk. The “AVI” chunk identifies the file as an AVI RIFF file. The LIST “hdrl” chunk defines the format of the data and is the first required list chunk. The LIST “movi” chunk contains the data for the AVI sequence and is the second required list chunk. The “idx1” chunk is the optional index chunk. AVI files must keep these three components in the proper sequence. The LIST “hdrl” and LIST “movi” chunks use subchunks for their data. The following example shows the AVI RIFF form expanded with the chunks needed to complete the LIST “hdrl” and LIST “movi” chunks:

```
RIFF ('AVI'
LIST ('hdrl'
'avih'(<Main AVI Header>)
LIST ('strl'
'strh'(<Stream header>)
'strf'(<Stream format>)
'strd'(additional header data)
.
.
.
)
.
.
.
)
LIST ('movi'
{SubChunk | LIST ('rec'
SubChunk1
SubChunk2
.
.
.
)
.
.
.
}
.
.
.
)
['idx1'<AVIIndex>]
)
```

The following sections describe the chunks contained in the LIST “hdrl” and LIST “movi” chunks as well as the “idx1” chunk.

Data Structures for AVI Files

Data structures used in the RIFF chunks are defined in the AVIFMT.H header file. The reference section at the end of this chapter describes the data structures that can be used for the main AVI header, stream header, AVIIndex, and palette change chunks.

The Main AVI Header LIST

The file begins with the main header. In the AVI file, this header is identified with “avih” four-character code. The header contains general information about the file, such as the number of streams within the file and the width and height of the AVI sequence. The main header has the following data structure defined for it:

```
typedef struct {
DWORD dwMicroSecPerFrame;
DWORD dwMaxBytesPerSec;
DWORD dwReserved1;
DWORD dwFlags;
DWORD dwTotalFrames;
DWORD dwInitialFrames;
DWORD dwStreams;
DWORD dwSuggestedBufferSize;
DWORD dwWidth;
DWORD dwHeight;
DWORD dwScale;
DWORD dwRate;
```

DWORD dwStart;

DWORD dwLength;

} MainAVIHeader;

The **dwMicroSecPerFrame** field specifies the period between video frames. This value indicates the overall timing for the file.

The **dwMaxBytesPerSec** field specifies the approximate maximum data rate of the file. This value indicates the number of bytes per second the system must handle to present an AVI sequence as specified by the other parameters contained in the main header and stream header chunks.

The **dwFlags** field contains any flags for the file. The following flags are defined:

AVIF_HASINDEX

Indicates the AVI file has an "idx1" chunk.

AVIF_MUSTUSEINDEX

Indicates the index should be used to determine the order of presentation of the data.

AVIF_ISINTERLEAVED

Indicates the AVI file is interleaved.

AVIF_WASCAPTUREFILE

Indicates the AVI file is a specially allocated file used for capturing real-time video.

AVIF_COPYRIGHTED

Indicates the AVI file contains copyrighted data. The AVIF_HASINDEX and AVIF_MUSTUSEINDEX flags apply to files with an index chunk. The AVI_HASINDEX flag indicates an index is present. The AVIF_MUSTUSEINDEX flag indicates the index should be used to determine the order of the presentation of the data. When this flag is set, it implies the physical ordering of the chunks in the file does not correspond to the presentation order. The AVIF_ISINTERLEAVED flag indicates the AVI file has been interleaved. The system can stream interleaved data from a CD-ROM more efficiently than non-interleaved data. For more information on interleaved files, see “Special Information for Interleaved Files." The AVIF_WASCAPTUREFILE flag indicates the AVI file is a specially allocated file used for capturing real-time video. Typically, capture files have been defragmented by user so video capture data can be efficiently streamed into the file. If this flag is set, an application should warn the user before writing over the file with this flag. The AVIF_COPYRIGHTED flag indicates the AVI file contains copyrighted data. When this flag is set, applications should not let users duplicate the file or the data in the file.

The **dwTotalFrames** field of the main header specifies the total number of frames of data in file.

The **dwInitialFrames** is used for interleaved files. If you are creating interleaved files, specify the number of frames in the file prior to the initial frame of the AVI sequence in this field.

The **dwStreams** field specifies the number of streams in the file. For example, a file with audio and video has 2 streams.

The **dwSuggestedBufferSize** field specifies the suggested buffer size for reading the file. Generally, this size should be large enough to contain the largest chunk in the file. If set to zero, or if it is too small, the playback software will have to reallocate memory during playback which will reduce performance. For an interleaved file, the buffer size should be large enough to read an entire record and not just a chunk.

The **dwWidth** and **dwHeight** fields specify the width and height of the AVI file in pixels. The **dwScale** and **dwRate** fields are used to specify the general time scale that the file will use. In addition to this time scale, each stream can have its own time scale. The time scale in samples per second is determined by dividing **dwRate** by **dwScal**e.

The **dwStart** and **dwLength** fields specify the starting time of the AVI file and the length of the file. The units are defined by **dwRate** and **dwScal**e. The **dwStart** field is usually set to zero.

The Stream Header (“strl”) Chunks

The main header is followed by one or more “strl” chunks. (A “strl” chunk is required for each data stream.) These chunks contain information about the streams in the file. Each “strl” chunk must contain a stream header and stream format chunk. Stream header chunks are identified by the four-character code “strh” and stream format chunks are identified with the four-character code “strf”. In addition to the stream header and stream format chunks, the “strl” chunk might also contain a stream data chunk. Stream data chunks are identified with the four-character code “strd”. The stream header has the following data structure defined for it:

```
typedef struct {
FOURCC fccType;
FOURCC fccHandler;
DWORD dwFlags;
DWORD dwReserved1;
DWORD dwInitialFrames;
DWORD dwScale;
DWORD dwRate;
DWORD dwStart;
DWORD dwLength;
DWORD dwSuggestedBufferSize;
DWORD dwQuality;
DWORD dwSampleSize;
} AVIStreamHeader;
```

The stream header specifies the type of data the stream contains, such as audio or video, by means of a four-character code. The **fccType** field is set to “vids” if the stream it specifies contains video data. It is set to “auds” if it contains audio data.

The **fccHandler** field contains a four-character code describing the installable compressor or decompressor used with the data.

The **dwFlags** field contains any flags for the data stream. The AVISF_DISABLED flag indicates that the stream data should be rendered only when explicitly enabled by the user.

The AVISF_VIDEO_PALCHANGES flag indicates palette changes are embedded in the file.

The **dwInitialFrames** is used for interleaved files. If you are creating interleaved files, specify the number of frames in the file prior to the initial frame of the AVI sequence in this field. The remaining fields describe the playback characteristics of the stream. These factors include the playback rate **(dwScale** and **dwRat**e), the starting time of the sequence **(dwStar**t), the length of the sequence **(dwLengt**h), the size of the playback buffer **(dwSuggestedBuffe**r), an indicator of the data quality **(dwQualit**y), and sample size **(dwSampleSiz**e). See the reference section for more information on these fields. Some of the fields in the stream header structure are also present in the main header structure. The data in the main header structure applies to the whole file while the data in the stream header structure applies only to a stream. A stream format (“strf”) chunk must follow a stream header (“strh”) chunk. The stream format chunk describes the format of the data in the stream. For video streams, the information in this chunk is a BITMAPINFO structure (including palette information if appropriate). For audio streams, the information in this chunk is a WAVEFORMATEX or PCMWAVEFORMAT structure. (The WAVEFORMATEX structure is an extended version of the

WAVEFORMAT structure.) For more information on this structure, see the *New Multimedia Data Types and Data Techniques Standards Updat*e. The “strl” chunk might also contain a stream data (“strd”) chunk. If used, this chunk follows the stream format chunk. The format and content of this chunk is defined by installable compression or decompression drivers. Typically, drivers use this information for configuration. Applications that read and write RIFF files do not need to decode this information. They transfer this data to and from a driver as a memory block. An AVI player associates the stream headers in the LIST “hdrl” chunk with the stream data in the LIST “movi” chunk by using the order of the “strl” chunks. The first “strl” chunk applies to stream 0, the second applies to stream 1, and so forth. For example, if the first “strl” chunk describes the wave audio data, the wave audio data is contained in stream 0. Similarly, if the second “strl” chunk describes video data, then the video data is contained in stream 1.

The LIST “movi” Chunk

Following the header information is a LIST “movi” chunk that contains chunks of the actual data in the streams; that is, the pictures and sounds themselves. The data chunks can reside directly in the LIST “movi” chunk or they might be grouped into “rec ” chunks. The “rec ” grouping implies that the grouped chunks should be read from disk all at once. This is used only for files specifically interleaved to play from CD-ROM. Like any RIFF chunk, the data chunks contain a four-character code to identify the chunk type. The four-character code that identifies each chunk consists of the stream number and a two-character code that defines the type of information encapsulated in the chunk. For example, a waveform chunk is identified by a two-character code of “wb”. If a waveform chunk corresponded to the second LIST “hdrl” stream description, it would have a four-character code of “01wb”. Since all the format information is in the header, the audio data contained in these data chunks does not contain any information about its format. An audio data chunk has the following format (the ## in the format represents the stream identifier):

WAVE Bytes '##wb'

BYTE abBytes [];

Video data can be compressed or uncompressed DIBs. An uncompressed DIB has BI_RGB specified for the **biCompression** field in its associated BITMAPINFO structure. A compressed DIB has a value other than BI_RGB specified in the **biCompression** field. A data chunk for an uncompressed DIB contains RGB video data. These chunks are identified with a two-character code of “db” (db is an abbreviation for DIB bits). Data chunks for a compressed DIB are identified with a two-character code of “dc” (dc is an abbreviation for DIB compressed). Neither data chunk will contain any header information about the DIBs. The data chunk for an uncompressed DIB has the following form:

DIB Bits '##db'

BYTE abBits [];

The data chunk for a compressed DIB has the following form:

Compressed DIB '##dc'

BYTE abBits [];

Video data chunks can also define new palette entries used to update the palette during an AVI sequence. These chunks are identified with a two-character code of “pc” (pc is an abbreviation for palette change). The following data structure is defined palette information:

typedef struct {

BYTE bFirstEntry;

BYTE bNumEntries;

WORD wFlags;

```
PALETTEENTRY peNew;
} AVIPALCHANGE;
```

The **bFirstEntry** field defines the first entry to change and the **bNumEntries** field specifies the number of entries to change. The **peNew** field contains the new color entries. If you include palette changes in a video stream, set the AVITF_VIDEO_PALCHANGES flag in the **dwFlags** field of the stream header. This flag indicates that this video stream contains palette changes and warns the playback software that it will need to animate the palette.

The “idx1” Chunk

AVI files can have an index chunk after the LIST “movi” chunk. The index chunk essentially contains a list of the data chunks and their location in the file. This provides efficient random access to the data within the file, because an application can locate a particular sound sequence or video image in a large AVI file without having to scan it. Index chunks use the four-character code “idx1”. The following data structure is defined for index entries:

```
typedef struct {
DWORD ckid;
DWORD dwFlags;
DWORD dwChunkOffset;
DWORD dwChunkLength;
} AVIINDEXENTRY;
```

The **cki**d, **dwFlag**s, **dwChunkOffse**t, and **dwChunkLength** entries are repeated in the AVI file for each data chunk indexed. If the file is interleaved, the index will also have these entries for each “rec” chunk. The “rec” entries should have the AVIIF_LIST flag set and the list type in the **ckid** field. The **ckid** field identifies the data chunk. This field uses four-character codes for identifying the chunk. The **dwFlags** field specifies any flags for the data. The AVIIF_KEYFRAME flag indicates key frames in the video sequence. Key frames do not need previous video information to be decompressed. The AVIIF_NOTIME flag indicates a chunk does not affect the timing of a video stream. For example, changing palette entries indicated by a palette chunk should occur between displaying video frames. Thus, if an application needs to determine the length of a video sequence, it should not use chunks with the AVIIF_NOTIME flag. In this case, it would ignore a palette chunk. The AVIIF_LIST flag indicates the current chunk is a LIST chunk. Use the **ckid** field to identify the type of LIST chunk. The **dwChunkOffset** and **dwChunkLength** fields specify the position of the chunk and the length of the chunk. The **dwChunkOffset** field specifies the position of the chunk in the file relative to the 'movi' list. The **dwChunkLength** field specifies the length of the chunk excluding the eight bytes for the RIFF header. If you include an index in the RIFF file, set the AVIF_HASINDEX in the **dwFlags** field of the AVI header. (This header is identified by “avih” chunk ID.) This flag indicates that the file has an index.

Chapter One
Introduction

1.1 Image and Video Compression [Ama02]

Image compression addresses the problem of reducing the amount of data required to represent a digital image. In general, image compression is possible because of the existing redundancy in uncompressed images. In digital image compression, three basic data redundancies can be identified and exploited:

1. **Coding Redundancy:** It occurs when the data used to represent the image are not utilized in an optimal manner. That is, it occurs when the gray levels of an image are coded in a way that uses more codes than absolutely necessary to represent each gray level. For example, if an 8-bit/pixel image, which allows 256 different intensity levels, is used to represent a 16-color image, actually only 4-bit/pixel is needed to represent the image. In general, coding redundancy is perfect when the codes assigned to the set of gray levels have not been selected to take the full advantage of the probabilities of gray levels.
2. **Inter Pixel Redundancy:** It occurs because the adjacent pixels tend to be highly correlated. This is a result of the fact that in most images the brightness levels do not change rapidly, but change gradually, so that adjacent pixels values tend to be relatively close to each other in value (for video, or motion images), this concept can be extended to include interframe redundancy (i.e. redundancy between frames of image data).

3. **Psycho-Visual Redundancy:** refers to the fact that some information are more important to the human visual system than other types of information. For example, we can only perceive spatial frequencies below about 50 cycles per degree, so that any higher frequency information is of little interest to us.

1.2 Standard Image Compression Methods [Mic98]

Standardization of still images and video compression techniques has become a high priority issue, because only a standard can reduce the high cost and resolve the critical problem of interoperability of equipment's from different manufactures.

The following summaries the most commonly known compression standards;

JPEG

The JPEG is the standard developed by Joint Photographic Experts Group for compressing still pictures (e.g. Photographs). JPEG had worked toward establishing the first international digital image compression standard for continuous-tone still image, both grayscale and color.

MPEG

MPEG (Moving Picture Experts Group) is one of the developments of international standards for compression, decompression, and representation of moving pictures and audio.

MPEG-1

The first finalized standard was MPEG-1 (International Standard). Its goal was to produce video recorder quality output (352×240) using a bit rate of 1.2 Mbps. Since the uncompressed video alone can run to 77.4 Mbps, getting it down to 1.2 Mbps is not entirely trivial, even at this lower resolution.

MPEG-2

MPEG-2 (International Standard) was originally designed for compressing broadcast quality video into 4 to 6 Mbps. Later, MPEG-2 was expanded to support higher resolutions, including HDTV.

MPEG-4

The third generation of MPEG is based upon the same technique. Once again, the new project focused on new application usages. The most important new features of MPEG-4, concerning video compression are the support of even lower bandwidth consuming applications, e.g. mobile units, and on the other hand applications with extremely high quality and almost unlimited bandwidth. The making of studio movies is one such an example.

1.3 Motion Estimation (ME) for Video Compression [Hyc^{2}01]

Motion estimation (ME) has been a hot research topic for years. It is the most important part in video compression and coding, it exploits as much temporal redundancies as possible to reduce the size of the data required in digital video storage and transmission. Low bit rate video transmission is impossible without the use of motion estimation. Although motion estimation is such a useful method in reducing the size of a coded video sequence, it is computationally intensive which makes real-time video coding a difficult task, but not impossible, to be accomplished. In a typical video encoding system, motion estimation can take 50%~75% (for the case of full search block matching) of the computation time. In the past two decades, extensive researches were conducted to develop motion estimation techniques. Many motion estimation techniques like pel-recursive techniques, gradient techniques, frequency domain techniques and block based matching techniques were evolved. Among these motion estimation techniques, block-based matching had been widely adopted by international standards such as the H.261-11, H.263-12, MPEG1-13 and MPEG2-14 due to its effectiveness and robustness.

1.4 Block Based Motion Estimation [Hyc^{2}01]

The principle of block based motion estimation in most of the video standards is that the video image frame is partitioned into blocks, where each block is the elementary unit. Motion estimation is performed by matching each block in the current frame against a region in a reference frame to find the best match. The matching criteria for the best match is well accepted to be the block in the search region such that the error or energy of the residue obtained from the subtraction of corresponding pixels between the blocks is minimized.

1.5 The Goals Factors in Motion Estimation [Moh99]

Most of the research works have been concentrated on optimizing the block-based motion estimation technique. As the demand for real-time video applications (like video recording, video conferencing, video phone, etc) the needs for video coding had been grown. Fast video encoding with good compression ratio as well as high signal to noise ratio is highly essential. Good compression ratio means reducing the size of the coded video with little degradation of quality. Motion estimation is exactly a technique designed to achieve good compression ratio in video compression. However, speed and quality are often two contradicting goals. Nowadays, researchers are still actively investigating for an optimum trade-off between these two factors. Most of the motion proposed estimation algorithms tends to bias toward speed by sacrificing visual quality. In view of this, we were motivated to find a good trade-off between the speed and quality. That is to increase the speed up as much as possible with good visual results. We focused on the block based motion estimation technique since it is widely adopted by most international standards. In this work we have proposed a model to formulate a method to predict the blocks motion, the suggested ME method is hybrid, its search mechanism is based on combining the search ideas utilized in the two standards ME methods.

1.6 Literature survey

Many researches were study the field of image compression; few of them focus on video compression, some of them are listed below:

1. **H. Y. Chunge, Adaptive Search Center Non-Linear Three Step Search, 2001, University of Hong Kong [Hyc^{1}01]**

   This research presents a new motion estimation algorithm using an adaptive search center predicted from its adjacent blocks. It does not have the problem of being trapped by local minimum, and is characterized by finding the majority motion vector in one step. When compared with six other block-based search algorithms including the full-search and three-step-search, the new algorithm has an average PSNR very close to that of full search.

2. **Alice Yu, Motion Search Performance using the H.263 Encoder, 1997, EE392c [Ali97]**

   In this research six different motion search algorithms are implemented within the context of the baseline H.263 encoder. This approach allowed considering the motion search algorithms in two different ways: first, in terms of the prediction error variance, which is indicative of the entropy of the resulting information; and, secondly, in terms of overall encoder performance, which considers how the motion search performs in an H.263 video system.

   The motion searches that considered are:

   a. Exhaustive search

   b. Two-dimensional logarithmic search

   c. Three-step hierarchical search (TSHS)

   d. One-at-a-time search (OTS)

   e. Full axis search

   f. Projection SAD method

3. **Amal Abbas Kadhim, H.263 Image Video Compression [Ama02]**

This project aims to implement the H263 video compression by developing all the required programs. In this work an adaptive mechanism was proposed and implemented to handle the time delay associated with all searching methods. The goal of the proposed mechanism is to not affect the compression efficiency and image quality.

4. **Marcin Chady, Application of the Bulk Synchronous Parallel Model in Fractal Image Compression, University of Birmingham [Mar00]**

   In this research was present with the results of an investigation into parallel implementations of fractal image compression algorithms. In particular, the research addressed the applicability for this purpose of the new Bulk Synchronous Parallel model. The research provides a scalable and predictable framework for developing parallel software, with a reliable and straightforward cost model, taking advantage of this model to arrive at an optimal parallel fractal image compression algorithm.

5. **Raouf Hamzaoui, Fractal Image Compression, Leipzig University [Rao01]**

   The research speed up Fractal searching through using nearest neighbor search, where Range blocks and domain blocks are assigned d-dimensional feature vectors such that searching in the pool of domain blocks can be restricted to the domain blocks whose feature vectors are the nearest neighbors of the feature vector of the current domain pool. The research studied the effects on computation time, image fidelity and compression ratio. The research showed that there is no need for keeping domains with low intensity variance in the pool.

6. Auday Ali H. Al-Dulaimy, Fractal Image Compression [Aud00]

This work aimed to develop FIC. The main scheme of FIC method was implemented, which lead to a good compression performance with a significant reduction in coding time. A speeding-up operation based on a new mathematical approach for determining the IFS-codes between the range and the domain blocks was introduced, also the utilization of the parallel processing to perform the encoding operation was discussed.

1.7 Aim of Thesis

The research aims to design video compression systems that compress a sequence of video frames into a compact version keeping the quality of the decompressed sequence of video frames. The designed video compression systems are mainly constructed by implementing different compression techniques.

The proposed work aims to develop two different models for video compression. In the implementation stages most of the well-known motion search methods are implemented, tested and their performances are investigated.

In the proposed work, the Fractal image compression (FIC) technique is implemented, in addition to the DCT (discrete cosine transform) as an image transform coding technique. Each model implemented with two different approaches, one for compressing the anchor frames, and one for compressing the estimated frames using motion estimation methods.

The first model implemented with compression system based on DCT transform for anchor frames after developing the DCT transform through speeding up its computations by a new derivation that eliminate much of its mathematical operations, and the model use three different methods (TSS, OTS, HM) for motion estimation compressing the estimated frames.

The second model implemented with compression system based on Fractal Image Compression for anchor frames after developing the FIC through speeding up its computations by design and implement a distributed system that will divide the fractal search on the total number of computers shared on the LAN, and the model use the same motion estimation methods compressing the estimated frames.

1.8 Thesis Layout

The work in this thesis is organized as follows:

- **Chapter (2):** explains the image and video compression techniques in details including Fractal image compression, and the methods of Motion Estimation, and show the importance of distributed systems for speeding up the implementation of video compression techniques.
- **Chapter (3):** this chapter includes all the details of the designed and implemented video compression models. All the algorithms used in this work are presented.
- **Chapter (4):** this chapter contains the results of some tests applied on some samples of movies used as test material in this work; the used criteria are the fidelity measures (MSE, PSNR) beside the compression ratios.
- **Chapter (5):** includes the derived conclusions and some suggestions for future work.

Chapter Two
Image and Video Compression

2.1 Introduction

Image compression had been pushed to the forefront of the image processing field. As a result of: ***the rapid growth in computer power, the corresponding growth in the multimedia market***, and ***the advent of the World Wide Web which makes the Internet easily accessible for everyone***. Additionally, the advances in video technology, including high-definition television, had created a demand for new, better, and faster image compression algorithms. Compression algorithm development started with applications of two-dimensional (2-D) still images. Because video and television signals consist of consecutive frames of 2-D image data, the development of compression methods for 2-D still data is of paramount importance. After the development of different still image compression schemes, some of them are often extended to video (motion imaging) **[Sco98]**.

The increasing demand to incorporate video data into telecommunications services, the corporate environment, the entertainment industry, and even at home had made digital video technology a necessity. However, the problem is that still image and digital video data rates are very large, typically in the range of 150Mbits/sec. Data rates of this magnitude would consume a lot of the bandwidth, storage and computing resources in the typical personal computer. For this reason, video compression is needed to reduce the data to be stored or transmitted through eliminate picture redundancy **[Arr97]**.

This chapter explores the theoretical concept of image and video compression, in addition to multimedia networking concepts.

2.2 Image Compression

Compression process takes an input X and generates a representation X_C that hopefully requires fewer bits. While the reconstruction algorithm operates on the compressed representation X_C to generate the reconstruction Y.

Based on the difference between original and the reconstructed version, data compression schemes can be divided into two broad classes (see figure (2.1)).

The first is lossless compression, at which Y is identical to X, while other is lossy compression, which generally provides much higher compression than lossless compression but makes Y different from X **[Add00]**.

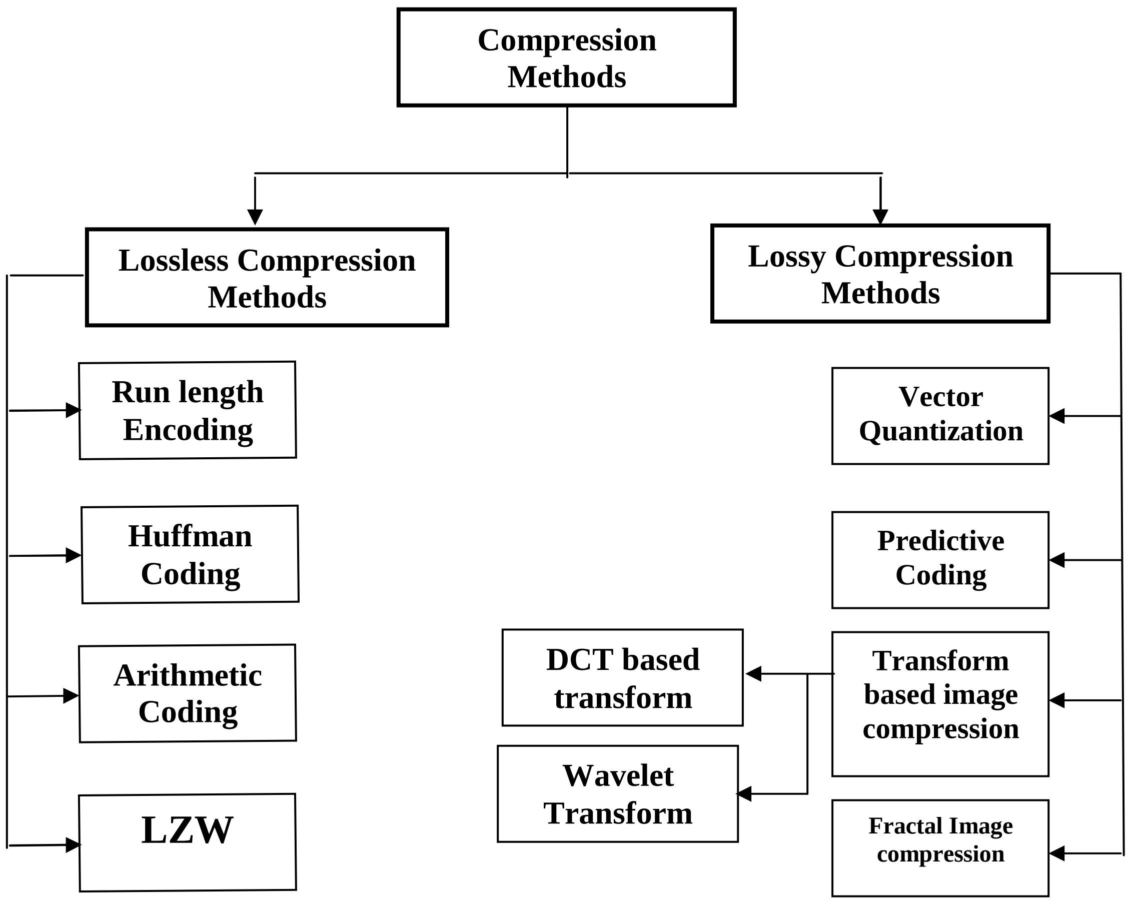


Figure 2.1: The Most Popular Image Compression Methods.

2.2.1 Lossless Compression Methods

Lossless compression techniques provide the guarantee that no pixel difference between the original and the decompressed image, i.e lossless schemes result in reconstructed data that exactly matches the original. It is generally used for applications that cannot allow any difference between the original and reconstructed data. The most popular lossless compression methods are ***Run Length coding, S-shift coding, Huffman coding, Lempel/Ziv algorithms, and arithmetic coding*** **[Ism02].**

A. Run Length Encoding [Sco98, Aud00]

Run length encoding (RLE), sometimes called recurrence coding, is one of the simplest data compression algorithms. It is effective for data sets that are comprised of long sequences of a single repeated character. For instance, text files with large runs of spaces or tabs may be compressed well with this algorithm. RLE finds runs of repeated characters in the input stream and replaces them with a three-byte code. The code consists of a flag character (called a sentinel byte), a count byte, and the value of repeated characters. For instance, the string “AAAAAABBBBCCCCC” could be more efficiently stored as “*6A*4B*5C”, that saves us six bytes. Of course, since it does not make sense to represent runs less than three characters in length which may exist in the code. Thus “AAAAAABBCCCDDDD” might be represented as “*6ABBCCC*4D”.

B. S-Shift Coding [Raf00, Ibr04]

The idea of this method is to encode the sequence of numbers by codewords whose bit length is less than the number of bits required to represent the maximum value of the sequence of numbers to be coded. The numbers whose values are large may splitted into a sequence of codewords, by using the formula (2.1):

$$X = nWm + Wr \quad ....(2.1)$$

Where:

X is the number to be coded.

n is the number of codeword to be coded.

W_m is the largest integer value, which must be coded by using a single codeword.

W_r is the value of the last codeword used to encode X.

The values of ***W_m*** and ***W_r*** are determined by using the following equations:

$$Wm = 2^b - 1 \quad ....(2.2)$$

$$Wr = X \bmod Wm \quad ....(2.3)$$

Where **b** is the number of bits used to represent each single shift codeword.

The performance of shift coding is better where the sequence of numbers has a histogram whose shape is highly peaked. The shift coding performance is better than especially when the histograms have long tails.

C. Huffman Coding [Pan01]

Huffman coding, developed by D.A. Huffman, is a classical data compression technique. It has been used in various compression applications, including image compression. It uses the statistical property of characters in the source stream and then produces respective codes for these characters. These codes are of variable code length using an integral number of bits. The codes for characters having a higher frequency of occurrence are shorter than those codes for characters having lower frequency. This simple idea causes a reduction in the average code length, and thus the overall size of compressed data is smaller than the original. Huffman coding is based on building a binary tree that holds all characters in the source at its leaf nodes, and with their corresponding characters' probabilities at the side. The tree is built by going through the following steps:

1. Each of the characters is initially laid out as leaf node; each leaf will eventually be connected to the tree. The characters are ranked according to their weights, which represent the frequencies of their occurrences in the source.
2. Two nodes with the lowest weights are combined to form a new node, which is a parent node of these two nodes. This parent node is then considered as a representative of the two nodes with a weight equal to the sum of the weights of two nodes. Moreover, one child, the left, is assigned a "0" and the other, the right child, is assigned a "1".
3. Nodes are then successively combined as above until a binary tree containing all of nodes is created.
4. The code representing a given character can be determined by going from the root of the tree to the leaf node representing the alphabet. The accumulation of "0" and "1" symbols is the code of that character.

By using this procedure, the characters are naturally assigned codes that reflect the frequency distribution. Highly frequent characters will be given short codes, and infrequent characters will have long codes. Therefore, the average code length will be reduced.

2.2.2 Lossy Compression Methods

Lossy compression techniques involve some loss of information, and data cannot be recovered or reconstructed exactly. In some applications, exact reconstruction is not necessary. For example, it is acceptable that the reconstructed video signal is different from the original as long as the differences do not result in annoying artifacts. Generally, lossy compression can produce a higher compression ratio than is possible with lossless compression. The most popular lossy compression methods are vector quantization, predictive coding, transform based image compression and fractal image compression. This work concerned with Transform Image Compression (TBIC), and Fractal Image Compression (FIC) [Jan03].

A. Transform Based Image Compression

Transform based compression implies the most popular and efficient coding schemes. Combined with other compression techniques this technique allows efficient transmission, storage, and display of images that otherwise would be impractical **[Pan01]**.

The basic transform encoding method for image compression works as follows:

1. **Image Transform**: Divide the source image into blocks and apply the transformations to each block.
2. **Parameter Quantization:** The data generated by the transformation are quantized to reduce the amount of information. Quantization is irreversible operation because of its lossy property.
3. **Encoding:** Encode the results of the quantization. This last step can be error free by using Run Length encoding or Huffman coding. It can also be lossy if it optimizes the representation of the information to further reduce the bit rate.

The discrete cosine transform (DCT) is a technique for converting a signal into elementary frequency components. It is widely used in image compression. It is a popular transform used by the JPEG (Joint Photographic Experts Group) image compression standard for lossy compression of images. Since it is used so frequently, DCT is often referred to in the literature as JPEG-DCT. JPEG-DCT is a transform coding method comprising four steps. The source image is first partitioned into sub-blocks of size 8×8 pixels in dimension. Then each block is transformed from spatial domain to frequency domain using a 2D DCT basis function. The resulting frequency coefficients are quantized and finally output to a lossless entropy coder. DCT is an efficient image compression method since it can decorrelate pixels in the image since the cosine basis is orthogonal, Orthogonal waveforms are waveforms that are independent of each other and compact most image energy to a few transformed coefficients. Moreover, DCT coefficients can be loosely quantized according to some human visual characteristics [Ken02, Jim99].

B. Fractal Image Compression (FIC)

The application of fractals in image compression was introduced by M.F. Barnsley and A. Jacquin. FIC is a process to find a small set of mathematical equations that can describe the image. By sending the parameters of these equations to the decoder, the original image can be reconstructed. In general, the theory of fractal compression is based on the mapping theorem in the mathematics of metric spaces. Analyzing the image forms the Partitioned Iterated Function System (PIFS), which is essentially a set of mappings. Those mappings can exploit the redundancy that is commonly present in most images. This redundancy is related to the similarity of an image with itself, that is, part *A* of a certain image is similar to another part *B* of the same image, by doing an arbitrary number of contractive transformations that can bring *A* and *B* together. These contractive transformations are actually common geometrical operations such as ***rotation, scaling, skewing*** and ***shifting*.** By applying the resulting PIFS on an initially blank image iteratively, the original image at the decoder can be completely regenerated. Since the PIFS often consists of a small number of parameters, a huge compression ratio can be achieved by representing the original image using these parameters. However, FIC has its disadvantages since it is usually involves a large amount of matching and geometric operations (i.e. it is time consuming). The coding process is so asymmetrical that encoding of an image takes much longer time than decoding [Mar00].

In fractal coding, the image to be coded is divided into non-overlapping range blocks, and larger possibly overlapping domain blocks. Every range block is expressed as an affine transformed version of one decimated domain block. Since domain blocks are larger than range blocks and if the maximum scaling factor is less than 1, this transform is a contraction. The reconstructed image, obtained by iterations of the transform *f* from any initial image, may be constructed to be still close to the original image [Fcc00].

Fractal image compression takes advantage of the fact that real life images are to a great extent self-similar. In other words, many parts of the image can be approximated by transforming another part of the same image by applying some ***affine spatial transformation*** and a (usually linear) ***brightness transformation***. Based on the theory of fractals, for a given image S, the compression process tries to find a ***Partitioned Iterated Function System*** **(PIFS)**, $\boldsymbol{W = \{w_i : i = 1, ..., k\}}$, whose attractor is a non-overlapping tiling of the image, where each of the "tiles" is formed by applying a contractive affine transformation w_i on a section of S [Fcc00].

$$S = W(S) = \bigcup_{i=1}^{k} w_i(d_i) \text{..........} \quad (2.4)$$

Where ***k*** is the number of range blocks.

Where $\mathbf{d_i}$ is an arbitrary section of the image, called ***domain***. The "tile" approximated by $\mathbf{w_i}$ $\mathbf{(d_i)}$, is further referred to as ***range*** or $\mathbf{r_i}$. Each transformation $\mathbf{w_i}$ $\mathbf{(d_i)}$ gives the best possible approximation of $\mathbf{r_i}$. This is usually measured with the ***Root Mean Square*** **(RMS)** metric of the following form:

$$RMS(r_i, w_i(d_i)) = \sqrt{\sum_{p \in w_i(d_i)} (r(p) - \tilde{d}(p))^2} \text{..........} .(2.5)$$

Where $\tilde{d}(p)$ represents the brightness of pixel ***p*** in the transformed domain fragment, and ***r*** **(p)** represents the brightness of the corresponding pixel in the range fragment. For given ***d*** and ***r*** the optimal brightness transformation can be found by minimizing the RMS distance. For a linear brightness transformation of the form

$$\tilde{d} = S \times d(p) + O \text{............} (2.6)$$

Where ***S*** is the contrast scaling, and ***O*** is the luminance shift, the optimal values of the coefficients, obtained by calculating the following:

$$S = \frac{n \sum_{i=1}^{n} d(p_i) r(p_i) - \sum_{i=1}^{n} d(p_i) \sum_{i=1}^{n} d(p_i)}{n \sum_{i=1}^{n} d^2(p_i) - (\sum_{i=1}^{n} d(p_i))^2} \text{..........} (2.7)$$

and

$$O = \frac{1}{n}(\sum_{i=1}^{n} r(p_i) - S \sum_{i=1}^{n} d(p_i)) \text{.........} (2.8)$$

Where ***n*** is the number of pixels in the image fragment, **d(*p*$_i$)** is the grey level of the ***i***th pixel in ***d,*** and **r(*p*$_i$)** is the grey level of the ***i***th pixel in ***r***. These formulas give the following lowest possible RMS error for d and r.

$$RMS=\sqrt{\frac{1}{n}[\sum_{i=1}^{n} r^2(p_i)+S(S\sum_{i=1}^{n} d^2(p_i)-2\sum_{i=1}^{n} d(p_i)r(p_i)+2O\sum_{i=1}^{n} d(p_i))+O(nO-2\sum_{i=1}^{n} r(p_i))]}$$

……….. (2.9)

In practice, a given image is typically partitioned into ***k*** rectangular or square ***range block*s**. Each of them is then encoded by searching for a twice bigger ***domain bloc*k**, so that the mapping transformation is contractive. Before determining the minimum RMS error between the two blocks should the domain block is ***decimate*d**, i.e. “shrunk” by a factor of 2, so that it has the same number of pixels as the range block. Isometric transformations (i.e. rotation and reflection) of the domain block are also allowed. Many coding schemes use ***quad-tree partitionin*g**, i.e. image blocks are divided recursively into smaller blocks, if no satisfactory encoding for the mother block can be found. Another common practice is to classify range and domain blocks into non-overlapping categories, so that comparisons between incompatible blocks can be avoided. The final result of all schemes is a chain of fractal coefficients (such as ***s*** and **o**, displacement, rotation, etc.) assigned to different parts of the image. When the transformations defined by these coefficients, are applied repetively to any initial image, they will yield the compressed image [Mar00].

2.3 Video Compression (3-D Image Compression)

A video stream can be considered as a sequence of two-dimensional images, or as a three-dimensional image. By breaking the stream into a series of blocks in the time direction, the result is three-dimensional image blocks suitable for application of a three-dimensional version. The video stream may be grayscale or color data. Color streams are handled in the same manner as color still images. Video compression is complicated by a number of factors, among these factors is the speed at which practical decompression may be performed. Another factor in extending two-dimensional methods to three-dimensional is the fact that humans perceive motion differently than still images. As in case of still images, only PSNR results are used to quantify distortion **[Sim01].**

2.3.1 Video Compression Techniques [Nic03]

Video compression technologies can be divided into two groups depending on the characteristics they use:

- Inter frame compression techniques
- Intra frame compression techniques

Table 2.1 below describes the main algorithms for each group and its main characteristics.

Table 2.1 – Compression Techniques

	Inter Frame	**Intra Frame**
Technique Name	MPEG-4 Main Profile, H.263+, MPEG-x	Wavelet, M-JPEG
Main Characteristics	Use both spatial redundancy and temporal redundancy	Use only spatial redundancy

The explanation of the characteristics given in the table is:

1. Using only **spatial redundancy** means that each frame is compressed separately.

2. Using **spatial redundancy** together with **temporal redundancy** means utilizing the similarity between consequent frames and motion estimations.

In order to better understand the differences between the different compression algorithms, it is important to review the conceptual differences between Inter and Intra frame algorithms.

Intra frame: at which compression algorithms process each frame separately, without analyzing the correlation between consequent frames (i.e spatial redundancy). M-JPEG and Wavelet use different methods for processing the frame and creating a compact presentation of it (either by filters (in Wavelet), or by a DCT transform (in M-JPEG)).

Inter frame: Interframe processing is the key to exploit and reduce the temporal redundancy in digital video compression (i.e utilizing the similarity between consequent frames and motion estimation). Temporal redundancy exists due to the similarity between the sequential neighboring frames. In video compression, knowledge of motion helps to exploit this similarity and remove the temporal redundancy between neighboring frames in addition to the spatial and spectral redundancies. Motion estimation (ME) or motion compensation (MC) are the basic approaches to find out and represent the motion between frames. These techniques are widely used in video standards including H.26x and MPEG to achieve high data compression rate.

Compression algorithms, such as MPEG-4 Main Profile and H.263+, use temporal redundancy as well as spatial redundancy. It actually means that these algorithms process consequent frames and estimate the motion within the frames. Then the algorithm codes only the difference between the frames instead of the whole frame. This coding method results in higher video quality per given bit-rate, since the given bit-rate is used for representing a smaller amount of video data ***(i.e. the***

frame difference). This is opposed to compression methods, which code the whole frame.

2.3.2 Video Compression Structure [Nic03]

The video coding structure will be build from the original video data that is represented as sequence of frames, the inter frame compression technique will divide the whole frames into two types, the first type consist of the ***Anchor frames***, and the second consist of the ***Estimated frames***. The Anchor frames are coded (compressed) independently and separately without any considerations to the correlations may exist with the neighboring frames, while the estimated frames are coded (compressed) using motion estimation methods, which will encode the estimated frames according to the correlations with the neighboring Anchor frames, as shown in figure 2.2 (video coding structure).

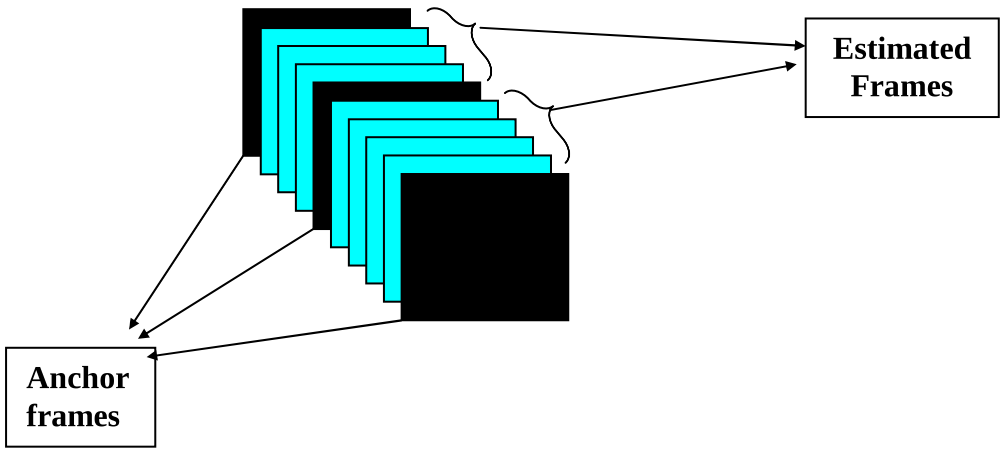


Figure 2.2 (Video coding structure)

2.3.3 Video Compression Standards **[Arr97, Mic98]**

During the '80s and '90s, Discrete Cosine Transform (DCT) based compression algorithms and international standards were developed to alleviate storage and bandwidth limitations imposed by digital still image and motion video applications. Today there are three DCT-based standards that are widely used and accepted worldwide:

- JPEG (Joint Photographic Experts Group)
- H.261 (Video codec for audiovisual services)
- MPEG (Motion Picture Experts Group)

Each of these standards is well suited for particular applications: JPEG for still image compression, H.261 for video conferencing, and MPEG for high-quality, multimedia systems.

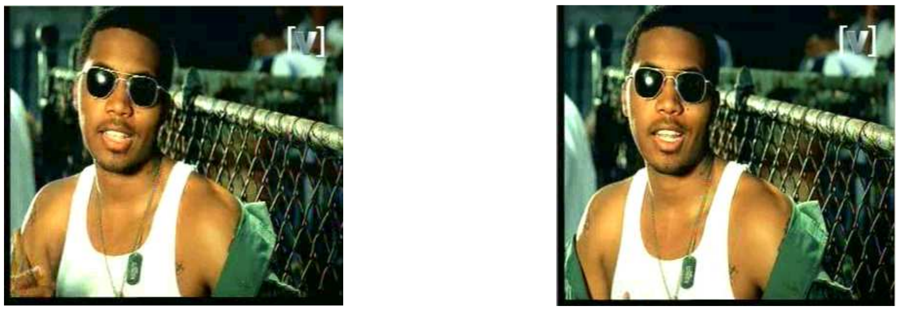
MPEG was set up in 1988 to develop a set of standard algorithms for applications that require storage of video and audio on digital storage media. The basic structure of compression algorithm proposed by MPEG is simple. An input image is divided into blocks of 8×8 pixels. For a given 8×8 block, we subtract the prediction generated using the previous frame. The difference between the block being encoded and the prediction is transformed using a DCT. The transform coefficients are quantized and transmitted to the receiver **[Pan01]**.

2.3.4 Motion Compensation (MC) and Motion Estimation (ME)

Video compression is involved with the removal of the spatial and temporal redundancies. The most widely used compression schemes are interframe and intraframe coding. The intraframe compression is a vital means of exploiting the spatial redundancies and interframe compression is used for exploiting the temporal redundancies. Due to the slow movement of videos such as 'head-and shoulders' video sequence, the two consecutive frames will not have much dramatic scheme change. So the current frame can be predicted from the translation of the previous frame. The method that is commonly used is ***Motion Compensation prediction***. Predictive coding is widely used in video transmission, especially for low bit rate coding. The vital part of the Motion Compensation is Motion Estimation, which is used for extracting the motion activity that exists between the frames. Block-Based Matching Algorithms (BMA) are popular methods for Motion Estimation because of their simplicity and ease of implementation [Som01].

2.3.4.1 The Slow Movement in Video [Gle01]

The two frames, shown in figure (2.3) illustrate the slow movement in video where the frame (a) is the frame number 90 in a video file, and the frame (b), is the frame number 95 at the same video file, where it obvious that, there is no dramatic changes, where there is a big similarity between the two frames, therefore the frame number 95 can be predicted from the previous frame.



a. Frame number 90 **b. Frame number 95**

Figure 2.3 Slow Movement in Video

MC involves removing the temporal redundancy. The basic idea behind motion compensation is to estimate the displacement of objects.

The methods used are called motion estimation (ME). Most of the standard ME methods are block matching methods **[Yil02]**.

2.3.4.2 Block-Based Motion Estimation Methods

The block matching is seemingly used by the video compression standards because it can achieve a good balance between complexity and coding efficiency. The goal of block matching is to find the best block from an earlier frame to reconstruct an area of the current frame. The block matching method can be categorized into frame based block matching and object based block matching method. There are many ME techniques used nowadays, ***the conventional frame based block matching*** technique is considered to be the full search or exhaustive search. This technique is often used for motion searching in relatively small search ranges due to the heavy computation and extension data fetching between the frame buffer and ME. In the last decade, many fast-searching algorithms have been developed to reduce the computation and data fetching by reducing the number of comparisons between the blocks, such as ***the 2-D logarithm, the three-step search*** (TSS), ***the orthogonal search and hierarchical block matching algorithms***. Also, there are proposals for improvements to fixed size block matching techniques by varying the size of blocks to more accurately match-moving areas known as variable size block matching (VSBM) methods. Since block MC causes visible block boundaries to appear in the predicted frame, the overlapped blocks MC was developed to reduce the blocking artifact **[Dde00]**.

The object block of the current frame is placed and moved around in the previous frame using a specific search strategy. A criterion is defined to determine how well the object block matches a corresponding block in the previous frame. These criteria can be the mean squared error

(MSE), minimum absolute difference (MAD) or the sum absolute difference (SAD). Since MSE gives the residual energy in the block difference, it is used as the criteria for BM ME **[Vir99]**.

A. Three Steps Search [Hyc^{1}01]

The Three Steps Search (TSS) will do three stages of matching, in each stage it will do eight matching blocks, these eight blocks are the eight neighbors around the signed block, and determine the best block, then will do the eight matching blocks around it, and so this matching will be repeated three times to determine the best block.

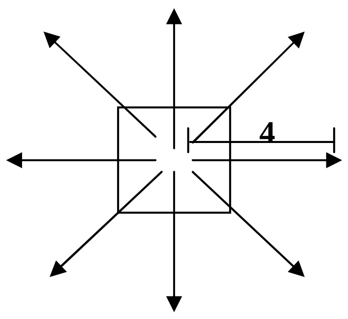


Figure 2.4 (a)

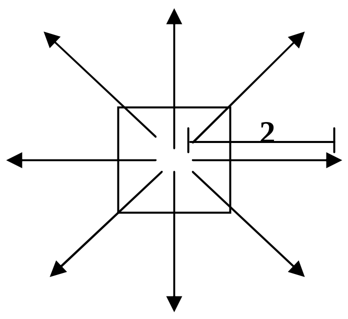


Figure 2.4 (b)

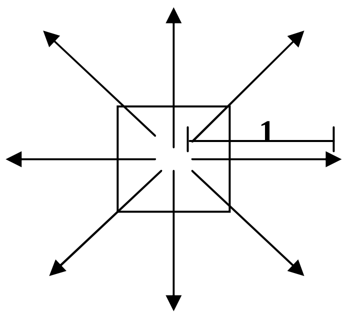


Figure 2.4 (c)

Figure (2.4): Three step search

The difference among the three stages is the offset of the eight blocks from the Macro block (MB), where in the **first stage**, the offset is

four pixels, in the second stage is ***two*** pixels, and in the **third stage** is only ***one*** pixel shifted, as shown in figure (2.4).

B. Once Time Search (OTS) [Ali97, Kar04]

The ***Once Time Search*** (OTS) behaved very differently from most other searching algorithms in that its behavior is almost entirely dependent on the sequence. Beyond its operative range, the size of the search area had little influence on its behavior.

The ***one at a time search*** (OTS) first evaluates points within the search window that are on the same row as the center point. If the center point is designated as (m, n), then the three positions (m-1, n), (m, n), and (m+1,n) are evaluated. If the best sum absolute difference (SAD) value corresponds to the central point, then the horizontal evaluation ends. If the best point corresponds to one the end points, then the end point becomes the center point, and three more points are evaluated. This continues until the best point is found at the center of the three points. Now the same procedure is conducted in the vertical direction, i.e., at the points (m, n-1), (m, n) and (m, n+1), beginning at the best point found in the horizontal search. Finally the method will continue search for the best SAD but in a diagonal way toward the original centre (first centre). The result of the diagonal search is the designated best vector location.

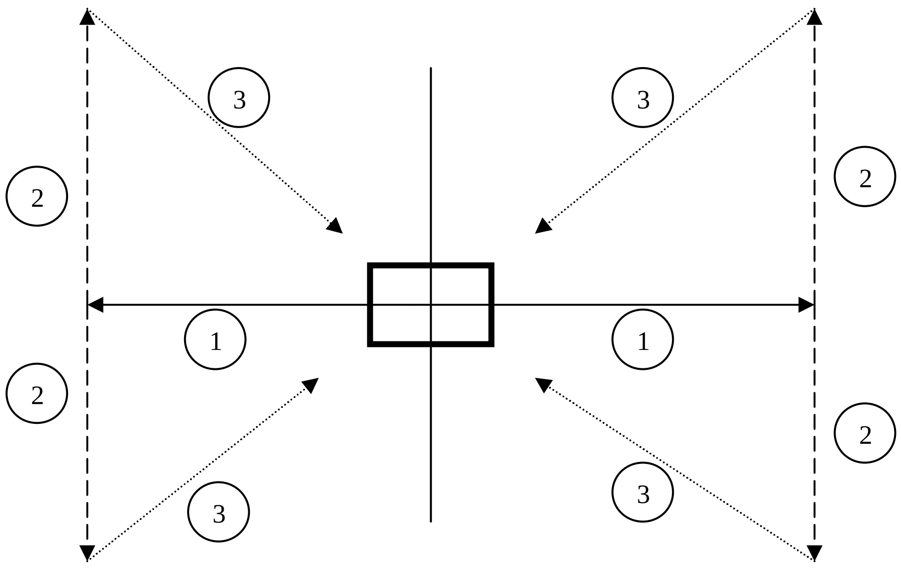


Figure (2.5): OTS Motion estimation

Figure (2.5) illustrates the three stages for the OTS search, as indexed on the figure, where at the first, the search will be on the horizontal axis, either to the left or to the right, as the motion that numbered (1), the second stage is a motion on the vertical axis for the new center as the motion number (2) on the figure, and finally, the third motion is a diagonal motion toward the original center as motion number (3).

2.4 Multimedia Networking [Jim99, Jud98]

Although traditional networks still exist for the separate delivery of voice, data, and video, new networks are often implemented as a combination of different types of remote communications merged into one delivery system. Indeed, many legacy networks are seeing the addition of multimedia applications as the greatest pressure to effective network throughput.

Virtually all modern Web sites contain rich graphic and image content, and many add simple animation and real-time audio. Real-time, voice, and video transmission are already feasible over the Internet. Point-to-point and point-to-multipoint transmission is easily accomplished for both audio and video, and multipoint conferences can be provided with little additional equipment. Broadcast, or so-called "net-cast," media has begun to provide streaming information content for such applications as news, financial data, and weather with multimedia.

Audio and video entertainment "channels" has begun in early on the Internet. At the present time, while sitting comfortably at the computer, one can tune in to his favorite jazz, classical, rock, easy listening, or alternative music from a variety of audio. Special purpose audio feeds are often available from live concerts. Video servers are now offering image transmission that includes periodic, still images as well as an increasing number of full-motion video feeds that may be broadcast to thousands.

2.5 Reasons for Multimedia Distributed Systems [Jud98]

1. ***Cost-of-Business Requirement***: Three basic approaches can be taken to managing networks.
    a. The first, and perhaps most common, is that network management is a necessary cost of business.
    b. It needs to be done to **keep the network functioning**.
    c. Certain expenses and actions must be undertaken.

    But as long as these functions are adequately performed, further spending and efforts are not justified as network management.

2. ***Ensuring Network Success***: The second approach is to view network management as a significant contribution to the success of the network. Network management contributes to the value of the network through the usage optimization of resources. Users of this approach tend to have a coherent strategy. This approach uses a large variety of the tools available and deploys them methodically.
3. ***Maintaining Service Levels***: The third approach is to view network management as a strategic component of maintaining service levels. Providing new services that are a result of management capabilities and even providing products to support the business are all characteristics of this approach to network management.
4. ***Focus on Services***: Another way to characterize these approaches is to note what uses and business purposes the network supports. Users of the first approach essentially provide data transmission. Those who take the second approach provide data services, and those who take the third approach are able to provide their users with a comprehensive set of information services. In this sense, data transmission is the movement of data from an input to an output location without regard to the content, or value of information, other than the external values assigned by the selection of the transport mechanism. The appropriate measurements for data transmission are the bit error rates, the packets lost, and the delay through the network.

Services are concerned about issues such as data integrity, rerouting, and other such service elements that come together to ensure timely and accurate delivery of the data. The measurements that are more appropriate to a data service are up time, throughput, effective data rates, and reachability. Information services deliver value from the network.

Through the monitoring, evaluation, and analysis of data transported through the network, and through specific information related services, they provide sources of information, as well as provide the service of data transport. The measures that are appropriate for information service are accuracy, response times, and consistency.

Chapter Three
Video Coding System

3.1 Introduction

The rapid growth of digital imaging applications, including multimedia and high-definition television (HDTV) has increased the need for effective and standardized image compression techniques. Among the emerging standards are JPEG (for compression of still images) and MPEG (for compression of motion video). Both of these standards employ a basic technique known as the discrete cosine transform (DCT). One of the most popular applications of DCT is image compression. The implementation of DCT was developed through a new derivation that fully documented in this chapter (see FDCT derivation).

Fractal coding is one of the promising image coding techniques, which provide high compression ratios, many software products start adopting fractal coding schemes. Blocks of an image are considered as affine transformations of other blocks taken from the image itself. Unfortunately, this coding costs too expensive time to complete its computation. A developed method was proposed in this work to speed up the standard fractal coding scheme, the proposed design and its implementation takes the advantages of the distributed system, which more than one computer is used to perform fractal coding (i.e., distribute the fractal coding operation on n-computers to speedup the coding operation).

Motion estimation is the process of estimating the motion of moving objects in a video scene. This is accomplished by determining the motion by which an object moves from one frame to the next. A given frame in a video sequence can be predicted from its previous frame by displacing all the moving objects in the previous frame by the estimated motion. Motion estimation and compensation form a major part in any video coding scheme. Many techniques are currently used for motion estimation. In the block-based approaches the most common applied procedure is the block matching based on various algorithms. Block based motion estimation forms the base of all video coding schemes. It involves finding a candidate block within a specified search area in the previous frame that is most similar to the considered (tested) block in the current frame. In the proposed work, standard methods OTS and TSS were implemented in addition to a new design method which represents a hybrid method (HM) combine the two standard methods for motion estimation.

3.2 Media File

The media file, which is used, as uncompress media file is the AVI file, where the system takes an AVI file as input media file and produce a compact file that represents the compressed video file.

The Microsoft AVI file format is a Resource Interchange File Format (RIFF), it is used with applications that capture, edit, and play back audio-video sequences. In general, AVI files contain multiple streams of different types of data. Most AVI sequences use both audio and video streams. A simple variation for an AVI sequence uses video data and does not require an audio stream, see appendix-A (AVI file format).

3.3 The Proposed Compressing Systems

In this work, two models of video coding are proposed and implemented. With each model, there are two major components: Anchor Frames

Compression (AFC), and Motion Estimation (or compensation) of Frames (MEF).

1. **Model-1:** In this model, a compression system based on FDCT transform was designed to encode Anchor frames. Through its implementation a new derivation of the original DCT transform (see section 3.4.6 **FDCT** Derivation**)** was adopted. While the Motion Estimation within Frames (MEF) are compressed using three different methods of Motion Estimation (ME), two standards methods, i.e ***Three Step Search (TSS)***, and ***Once Time Search (OTS)*** methods, in addition to a new designed hybrid method of Motion Estimation, it is based on combining some of the steps involved in the two used standard methods (TSS, and OTS).
2. **Model-2:** In this model, the compression system depends on Fractal Image Compression (FIC) technique was utilized to compress Anchor frames, while the adopted Motion Estimation methods are the same methods that utilized in the first model. The problem with Fractal Image Compression (FIC) is in its high computational requirements (i.e., time consuming), it needs an expensive time to complete Anchor Frame Compression (AFC). In this work, time consuming problem is solved by using multiprocessing system (i.e., distribute the fractal compression job among several PC's connected through LAN) to perform the fractal process.

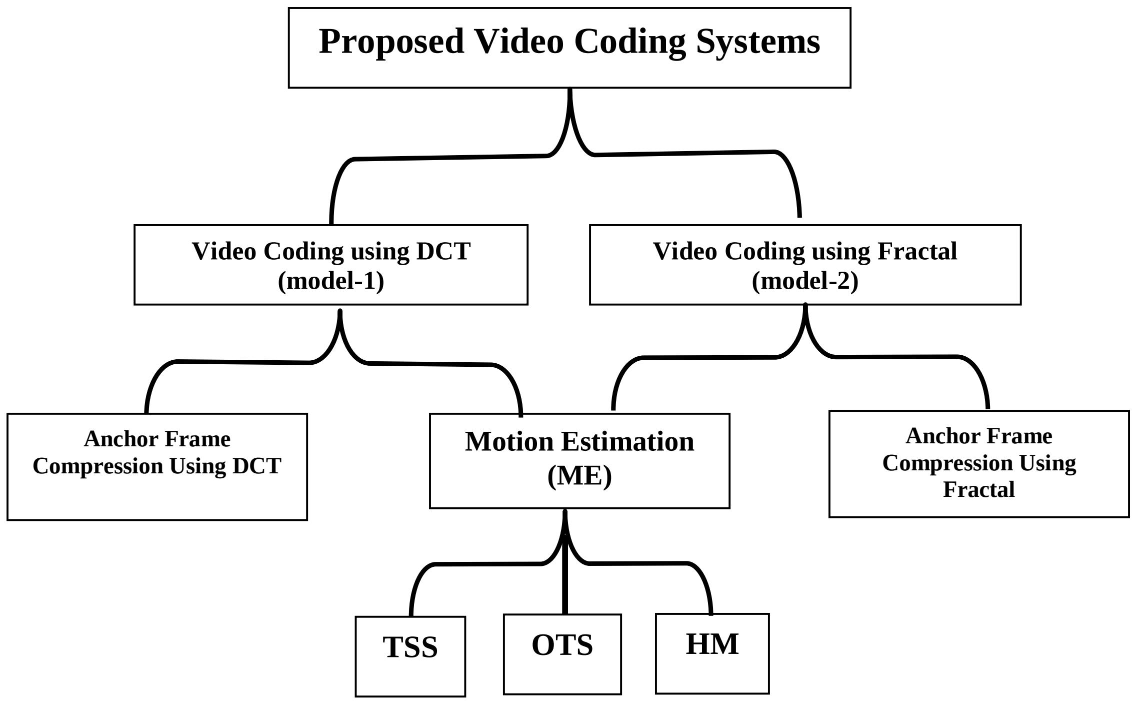

Proposed Video Coding Systems
Video Coding using DCT (model-1)
Video Coding using Fractal (model-2)
Anchor Frame Compression Using DCT
Motion Estimation (ME)
Anchor Frame Compression Using Fractal
TSS
OTS
HM

Figure (3.1) The Proposed Systems Structures

3.4 Video Coding Using FDCT (Model1)

As mentioned before, each model has two major jobs to do, Anchor Frame Compression (AFC), and Motion Estimation (ME). In this model, the Anchor Frame Compression uses a new version of DCT transform (FDCT), while the Motion Estimation uses one of the three Motion search methods (TSS, OTS, and HM).

3.4.1 The System Structure of Model1 Coding

Figure (3.2) illustrates the general coding scheme of the proposed model-1.

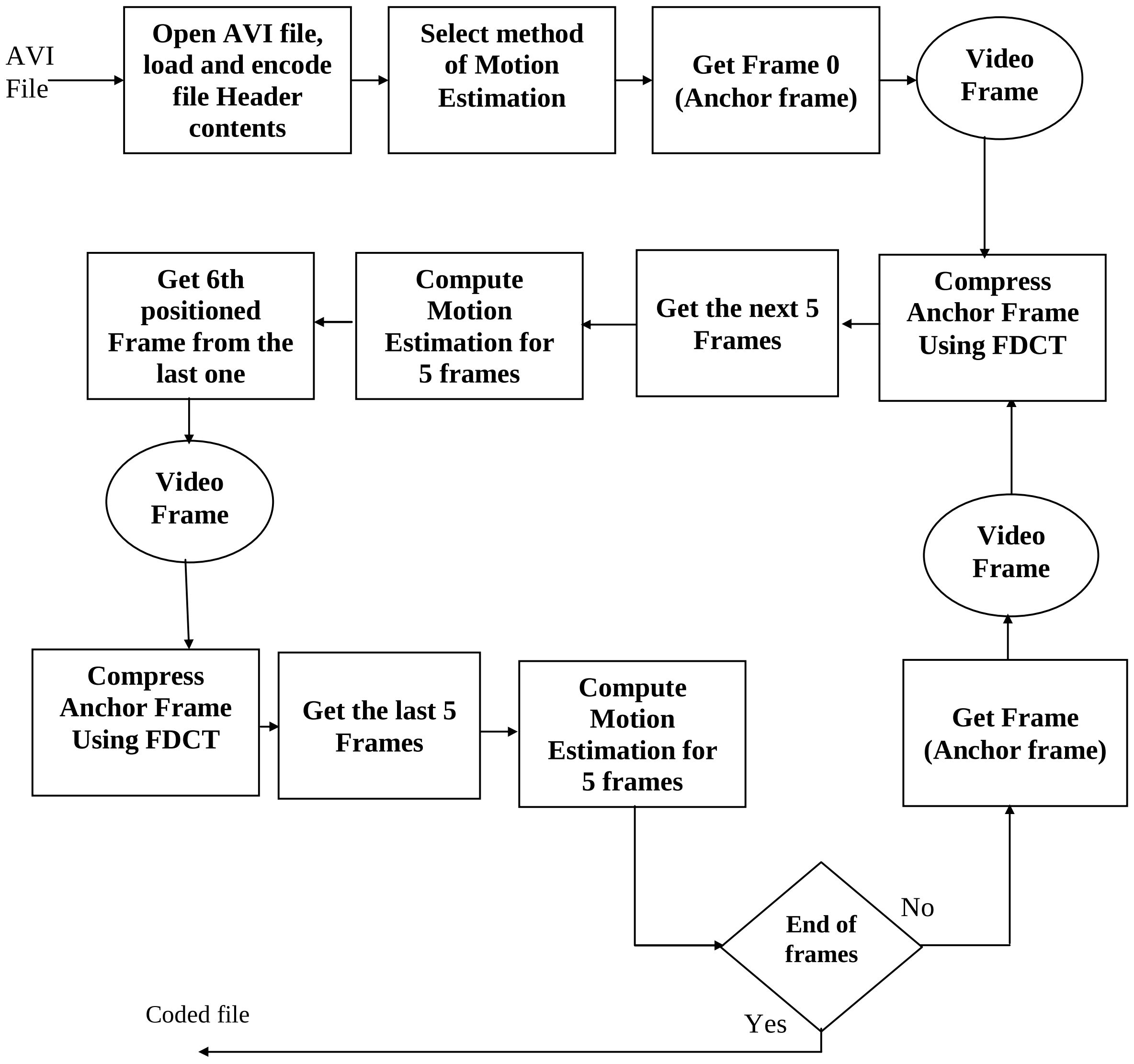

AVI File
Open AVI file, load and encode file Header contents
Select method of Motion Estimation
Get Frame 0 (Anchor frame)
Video Frame
Compress Anchor Frame Using FDCT
Get the next 5 Frames
Compute Motion Estimation for 5 frames
Get 6th positioned Frame from the last one
Video Frame
Compress Anchor Frame Using FDCT
Get the last 5 Frames
Compute Motion Estimation for 5 frames
End of frames
No
Yes
Get Frame (Anchor frame)
Video Frame
Coded file

Figure (3.2) The System Structure Model1 Coding

Where

1. At the first model, the system will load the header of AVI file and encode it (i.e., Header compression) after reducing the unnecessary information. The AVI header is a big header, it extends up to 127 kB, but after encoding the AVI header using S-shift encoder, the final size becomes 30 kB.
2. The user will select one of the three-available methods for motion estimation (TSS, OTS, or HM).
3. After getting a frame compress it as Anchor Frame (AF), compute the Motion Estimation (ME) for the next five sequenced frames, such that the determined motion estimation depends on the previous Anchor frame.
4. Skip five video frames and get a new video frame compressing it as Anchor Frame, where the skipped five frames will compared with this Anchor frame to compute their blocks motion.

3.4.2 Strategy of Video Coding

The video frame sequence will be divided into sets of frames groups, each group consists of 12 frames, the first and last frames in the group are considered as Anchor frames, and the other frames will be compressed by computing the motion estimation relative to the Anchor frames, considering the last frame is the first frame for the next group (i.e., the last frame of each group is shared with the next group).

The motion in the first five estimated frames is measured relative to the first Anchor frame, while the motion of last estimated frames depends on the last Anchor frame. Figure (3.3) illustrates the video coding strategy.

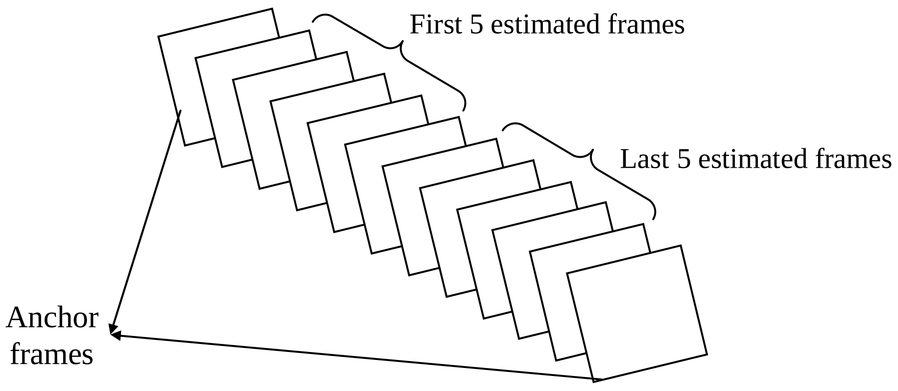


Figure (3.3) The Video Coding Strategy

3.4.3 Compress Anchor Frame Using FDCT

Figure (3.4) illustrates the implemented compression scheme, based on fast discrete cosine transform (FDCT), to compress the Anchor frames.

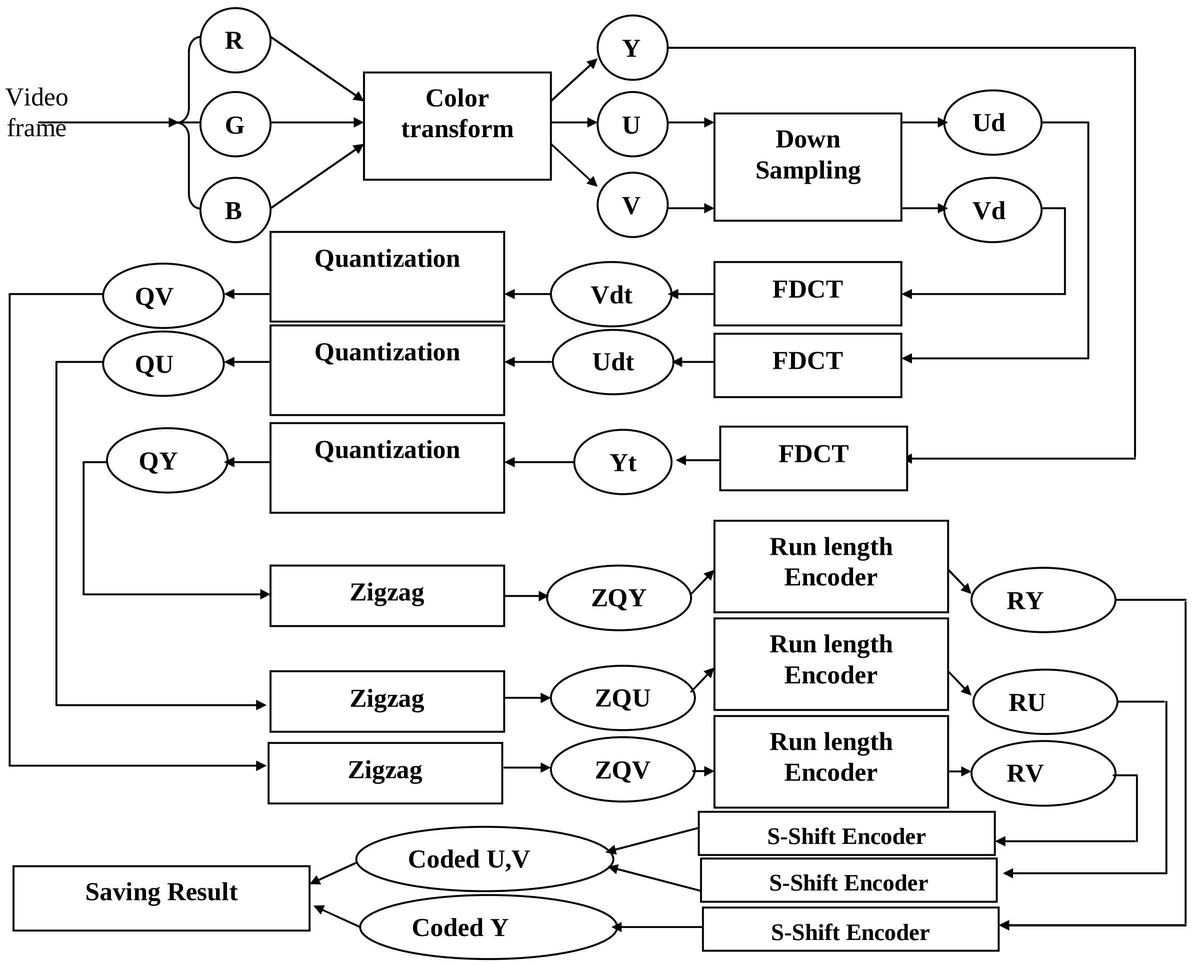


Figure (3.4) The Compression Scheme for Anchor Frames Using FDCT

The video frame is a 2D matrix of RGB components. The involved steps of the compression scheme could be summarized as follows:

1. In the first step the system will transform the color components (R, G, and B) to another color domain (i.e., YUV bands), see section 3.4.4 (the color transform).
2. Down sample the U, and V bands using the average method (see algorithm list 3.2). This is due to the fact the components U, and V hold 10% of the whole color information of the image matrix so this down sampling will reduce the required system computation without producing significant distortions in colors.
3. The result of down sampling for U, and V bands in addition to the Y band will DCT transformed using a new derivation of fast DCT transform, see section 3.4.6 (FDCT derivation).
4. Apply the quantization of the result of FDCT to reduce the number of bits needed to represent DCT coefficients, and then apply the zigzag ordering on the quantized FDCT coefficients, as shown in algorithm 3.5 (The Image Quantization and Zigzag).
5. Encode the result of zigzag process using Run length encoder, see algorithm 3.6 (Run Length Encoder).
6. And finally encode the output of the run length encoder using S-shift encoder, see algorithm 3.7 (Shift Key Encoder), and save the codeword in the output buffer.

3.4.4 The Color Transform

This standard is based on converting the RGB bands to YUV bands. Given a 24 bits/pixel RGB signal, we can find the Y, U, and V values as follows:

$Y = 0.299R + 0.587G + 0.114B$, ………………………… (3.1)

$U = -0.147R - 0.289G + 0.436B$, ………………………… (3.2)

$V = 0.615R - 0.515G - 0.11B$, ……………………. ……… (3.3)

From this transform, more than 90% of the information will concentrate into one band (Y), the remaining information will be in the other two bands. So the Y band is more important from the other two bands U, and V. The Color Transform (CT) is utilized in image compression schemes, because it will help to reduce the spectral redundancy, and also to exploit some of the characteristics of the human visual system (HVS) to improve the compression performance. Algorithm (3.1) shows the steps of the implemented algorithm.

<u>The Colors Transform</u>

Algorithm (3.1) the colors transform

```
Input: RGB Image (Img) H×W
Output: YUV Image (Rimg) H×W

For i=0 to H-1
   For j=0 to W-1
      • Compute Y band from Img(i,j), Y =(299*R(i,j)+587*G(i,j)+114*B(i,j))*0.001
      • Compute U band from Img(i,j) , U =(-147*R(i,j)-289*G(i,j)+436*B(i,j))*0.001
      • Compute V band from Img(i,j) , V =(615*R(i,j)-515*G(i,j)-11*B(i,j))*0.001
   Next j
Next i
```

3.4.5 Image Down Sampling

The goal of this method is to reduce the image size to quarter %25 of its original size, and it is done by replacing each four pixels by one pixel, whose value is the average of the four pixels.

The Down Sampling

Algorithm (3.2) The Down Sampling

```
Input: YUV Image (Img) H × W
Output: UV Image (UVimg) Dh × Dh

   • Dh=H div 2
   • Dw=W div 2

For i=0 to Dh-1: Id=i×2: Idp=Id+1
  For j=0 to Dw-1
   • Jd=j×2
   • Us=Img(Id, Jd).U+ Img(Idp, Jd).U+ Img(Id, Jd+1).U+ Img(Idp, Jd+1).U
   • UVimg(i, j).U=Us/4
   • Vs=Img(Id, Jd).V+ Img(Idp, Jd).V+ Img(Id, Jd+1).V+ Img(Idp, Jd+1).V
   • UVimg(i, j).V=Vs/4
  Next j
Next i
```

3.4.6 Fast DCT (a new derivation)

In the proposed work, a fast 8×8 DCT is developed to speedup the DCT execution, the derivation for final equations implemented in the fast 8×8 DCT transform is described in this section. The basic idea of speeding up the DCT transform is to reduce loops, mathematical floating operations, and some other factors. The algorithm of FDCT to transform one 8×8 block is given in algorithm list (3.3), so the system will divide the 2D image into 8×8 blocks, this transform must applied on the Y-components, and the down sampled components of U and V.

The DCT transform is one of the important image transforms, which used in verity image applications, but the disadvantage of the DCT transform is its long encoding/decoding time, so it is important to make this transform faster. In this work, a new implementation of fast 8×8 DCT (FDCT) transform is considered.

The basic idea of speeding up the DCT transform is to reduce loops, mathematical operations, and some other factors.

A. The Standard DCT Transform

The equation of 1D cosine transform is given as follows

$$F(u) = C(u)\sum_{x=0}^{N-1} f(x)\cos(\frac{u(2x+1)\pi}{2N}) \quad ....\,.(3.4)$$

$$C(u) = \begin{cases} \sqrt{\frac{1}{N}} & if u \neq 0 \\ \sqrt{\frac{2}{N}} & if u = 0 \end{cases}$$

For the case N=8, the above DCT equation could by simplified into equations (3.5):

$F(0)=(\sigma_0 f_0 + \sigma_1 f_1 + \sigma_2 f_2 + \sigma_3 f_3 + \sigma_4 f_4 + \sigma_5 f_5 + \sigma_6 f_6 + \sigma_7 f_7)/R$

$F(1)=(\varsigma_0 f_0 + \varsigma_1 f_1 + \varsigma_2 f_2 + \varsigma_3 f_3 + \varsigma_4 f_4 + \varsigma_5 f_5 + \varsigma_6 f_6 + \varsigma_7 f_7)/R$

$F(2)=(\upsilon_0 f_0 + \upsilon_1 f_1 + \upsilon_2 f_2 + \upsilon_3 f_3 + \upsilon_4 f_4 + \upsilon_5 f_5 + \upsilon_6 f_6 + \upsilon_7 f_7)/R$

$F(3)=(\varphi_0 f_0 + \varphi_1 f_1 + \varphi_2 f_2 + \varphi_3 f_3 + \varphi_4 f_4 + \varphi_5 f_5 + \varphi_6 f_6 + \varphi_7 f_7)/R$

$F(4)=(\psi_0 f_0 + \psi_1 f_1 + \psi_2 f_2 + \psi_3 f_3 + \psi_4 f_4 + \psi_5 f_5 + \psi_6 f_6 + \psi_7 f_7)/R$

$F(5)=(\alpha_0 f_0 + \alpha_1 f_1 + \alpha_2 f_2 + \alpha_3 f_3 + \alpha_4 f_4 + \alpha_5 f_5 + \alpha_6 f_6 + \alpha_7 f_7)/R$

$F(6)=(\beta_0 f_0 + \beta_1 f_1 + \beta_2 f_2 + \beta_3 f_3 + \beta_4 f_4 + \beta_5 f_5 + \beta_6 f_6 + \beta_7 f_7)/R$

$F(7)=(\delta_0 f_0 + \delta_1 f_1 + \delta_2 f_2 + \delta_3 f_3 + \delta_4 f_4 + \delta_5 f_5 + \delta_6 f_6 + \delta_7 f_7)/R$

Where

$\sigma i = \text{round}(C(0) \times \text{COS}(0) \times R)$

$\varsigma i = \text{round}(C(1) \times \text{COS}(((2 \times i+1) \times \pi)/16) \times R)$

$\upsilon i = \text{round}(C(2) \times \text{COS} ((2 \times (2 \times i+1) \times \pi)/16) \times R)$

$\varphi i = \text{round}(C(3) \times \text{COS} ((3 \times (2 \times i+1) \times \pi)/16) \times R)$

$\psi i = \text{round}(C(4) \times \text{COS} ((4 \times (2 \times i+1) \times \pi)/16) \times R)$

$\alpha i = \text{round}(C(5) \times \text{COS} ((5 \times (2 \times i+1) \times \pi)/16) \times R)$

$\beta i = \text{round}(C(6) \times \text{COS} ((6 \times (2 \times i+1) \times \pi)/16) \times R)$

$\delta i = \text{round}(C(7) \times \text{COS} ((7 \times (2 \times i+1) \times \pi)/16) \times R)$

Where R is the accuracy number, which represents the accuracy needed after the decimal point. The purpose of this factor is to convert the coefficients from real numbers to integer numbers, because the execution of mathematical operations deals with integer numbers (i.e., integer operations) are faster than those deal with real numbers (floating point operations) in computers.

The values of the parameters (**σ_i, ς_i, υ_i…. δ_i**) are shown the following table:

Table 3.1 : 8×8 DCT Coefficients Values With R=1000

i	0	1	2	3	4	5	6	7
σ_i	500	500	500	500	500	500	500	500
ς_i	347	294	197	69	-69	-196	-294	-347
υ_i	327	135	-135	-326	-327	-136	134	326
φ_i	294	-69	-347	-197	196	347	70	-293
ψ_i	250	-250	-250	249	251	-249	-251	249
α_i	197	-347	68	295	-293	-71	347	-194
β_i	135	-327	326	-134	-137	328	-326	132
δ_i	69	-197	295	-347	346	-292	194	-65

So, it could be noticed that:

1. $\sigma_0= \sigma_1= \sigma_2= \sigma_3= \sigma_4= \sigma_5= \sigma_6= \sigma_7= 500$
2. $\varsigma_0=347$, $\varsigma_1=294$, $\varsigma_2=197$, $\varsigma_3=69$, $\varsigma_4=-\varsigma_3$, $\varsigma_5= -\varsigma_2$, $\varsigma_6= -\varsigma_1$, $\varsigma_7= -\varsigma_0$
3. $\upsilon_0=327$, $\upsilon_1=135$, $\upsilon_2= -\upsilon_1$, $\upsilon_3= -\upsilon_0$, $\upsilon_4= -\upsilon_0$, $\upsilon_5= - \upsilon_1$, $\upsilon_6=\upsilon_1$, $\upsilon_7=\upsilon_0$
4. $\varphi_0= \varsigma_1$, $\varphi_1= -\varsigma_3$, $\varphi_2= -\varsigma_0$, $\varphi_3= -\varsigma_2$, $\varphi_4= \varsigma_2$, $\varphi_5= \varsigma_0$, $\varphi_6= \varsigma_3$, $\varphi_7= -\varsigma_1$
5. $\psi_0=250$, $\psi_1= -\psi_0$, $\psi_2= -\psi_0$, $\psi_3= \psi_0$, $\psi_4= \psi_0$, $\psi_5= -\psi_0$, $\psi_6= - \psi_0$, $\psi_7= \psi_0$
6. $\alpha_0= \varsigma_2$, $\alpha_1= -\varsigma_0$, $\alpha_2= \varsigma_3$, $\alpha_3= \varsigma_1$, $\alpha_4= -\varsigma_1$, $\alpha_5= -\varsigma_3$, $\alpha_6= \varsigma_0$, $\alpha_7= -\varsigma_2$
7. $\beta_0= \upsilon_1$, $\beta_1= -\upsilon_0$, $\beta_2= \upsilon_0$, $\beta_3= -\upsilon_1$, $\beta_4= -\upsilon_1$, $\beta_5= \upsilon_0$, $\beta_6= -\upsilon_0$, $\beta_7= \upsilon_1$
8. $\delta_0= \varsigma_3$, $\delta_1=-\varsigma_2$, $\delta_2= \varsigma_1$, $\delta_3= -\varsigma_0$, $\delta_4= \varsigma_0$, $\delta_5=-\varsigma_1$, $\delta_6= \varsigma_2$, $\delta_7=-\varsigma_3$

***Note*:** the comparing coefficients are performed ignoring small errors

So we can rewrite the system (3.5) into equations (3.6) as follow:

F (0)= $(\sigma_0 f_0 + \sigma_0 f_1 + \sigma_0 f_2 + \sigma_0 f_3 + \sigma_0 f_4 + \sigma_0 f_5 + \sigma_0 f_6 + \sigma_0 f_7)/R$

F (1)= $(\varsigma_0 f_0 + \varsigma_1 f_1 + \varsigma_2 f_2 + \varsigma_3 f_3 - \varsigma_3 f_4 - \varsigma_2 f_5 - \varsigma_1 f_6 - \varsigma_0 f_7)/R$

F (2)= $(\upsilon_0 f_0 + \upsilon_1 f_1 - \upsilon_1 f_2 - \upsilon_0 f_3 - \upsilon_0 f_4 - \upsilon_1 f_5 + \upsilon_1 f_6 + \upsilon_0 f_7)/R$

F (3)= $(\varsigma_1 f_0 - \varsigma_3 f_1 - \varsigma_0 f_2 - \varsigma_2 f_3 + \varsigma_2 f_4 + \varsigma_0 f_5 + \varsigma_3 f_6 - \varsigma_1 f_7)/R$

F (4)= $(\psi_0 f_0 - \psi_0 f_1 - \psi_0 f_2 + \psi_0 f_3 + \psi_0 f_4 - \psi_0 f_5 - \psi_0 f_6 + \psi_0 f_7)/R$

F (5)= $(\varsigma_2 f_0 - \varsigma_0 f_1 + \varsigma_3 f_2 + \varsigma_1 f_3 - \varsigma_1 f_4 - \varsigma_3 f_5 + \varsigma_0 f_6 - \varsigma_2 f_7)/R$

F (6)= $(\upsilon_1 f_0 - \upsilon_0 f_1 + \upsilon_0 f_2 - \upsilon_1 f_3 - \upsilon_1 f_4 + \upsilon_0 f_5 - \upsilon_0 f_6 + \upsilon_1 f_7)/R$

F (7)= $(\varsigma_3 f_0 - \varsigma_2 f_1 + \varsigma_1 f_2 - \varsigma_0 f_3 + \varsigma_0 f_4 - \varsigma_1 f_5 + \varsigma_2 f_6 - \varsigma_3 f_7)/R$

And we can to simplify the system (3.6) to system (3.7) as follow:

F (0)= $\sigma_0 (f_0+f_1+f_2+f_3+f_4+f_5+f_6+f_7)/R$

F (1)= $[\varsigma_0 (f_0-f_7) + \varsigma_1 (f_1-f_6) + \varsigma_2 (f_2-f_5) + \varsigma_3 (f_3-f_4)]/R$

F (2)= $[\upsilon_0 (f_0-f_3-f_4+f_7) + \upsilon_1 (f_1-f_2-f_5+f_6)]/R$

F (3)= $[\varsigma_1 (f_0-f_7) - \varsigma_3 (f_1-f_6) - \varsigma_0 (f_2-f_5) - \varsigma_2 (f_3-f_4)]/R$

F (4)= $[\psi_0 (f_0-f_1-f_2+f_3+f_4-f_5-f_6+f_7)]/R$

F (5)= $[\varsigma_2 (f_0-f_7)- \varsigma_0 (f_1-f_6) + \varsigma_3 (f_2-f_5) + \varsigma_1 (f_3-f_4)]/R$

F (6)= $[\upsilon_1 (f_0-f_3-f_4+f_7) + \upsilon_0 (f_2-f_1+f_5-f_6)]/R$

F (7)= $[\varsigma_3 (f_0-f_7) - \varsigma_2 (f_1-f_6) + \varsigma_1 (f_2-f_5) - \varsigma_0 (f_3-f_4)]/R$

Now suppose that:

U1=f_0+f_7

U2=f_3+f_4

U3=f_1+f_6

U4=f_2+f_5

V1=U1+U2

V2=U3+U4

V3=U1-U2

V4=U3-U4

V5=f_0-f_7

V6=f_1-f_6

V7=f_2-f_5

V8=f_3-f_4

Substituting the equations (V1, V2 … V8) in equations (3.7) will lead to system (3.8):

F (0)= σ_0 (V1+ V2)/R

F (1)= [ς_0 V5 + ς_1V6 + ς_2 V7 + ς_3 V8]/R

F (2)= [υ_0V3 + υ_1 V4]/R

F (3)= [ς_1V5 – ς_3V6 - ς_0V7 – ς_2 V8]/R

F (4)= ψ_0(V1- V2)/R

F (5)= [ς_2V5 - ς_0V6 + ς_3V7 + ς_1V8]/R

F (6)= [υ_1V3 – υ_0V4]/R

F (7)= [ς_3V5 - ς_2V6 + ς_1V7 – ς_0V8]/R

The FDCT for One 8×8 Block

```
Algorithm (3.3) The FDCT for One 8×8 Block
Input: Gray Block (GB) 8×8
Output: DCT Block (DB) 8×8

For i=0 to 7
    • Load gray vector from the ith row of GB, ƒ(0).. ƒ(7)
    • Compute shared variables, V1..V8
    • Compute DCT coefficients σ0, ς0, ς1, ς2, ς3, υ0, υ1, ψ0
    • Compute DCT vector, F(0)..F(7) using system (3.8)
    • for r=0 to 7: TMP(r, i)=F(r): next r
Next i
For i=0 to 7
    • Load TMP vector from the ith column of TMP Block, ƒ(0).. ƒ(7)
    • Compute shared variables, V1..V8
    • Compute DCT coefficients σ0, ς0, ς1, ς2, ς3, υ0, υ1, ψ0
    • Compute DCT vector, F(0)..F(7) using system(3.8)
    • for r=0 to 7: DB(i, r)=F(r): next r
Next i
```

B. Inverse DCT (IDCT)

The equation of inverse DCT is:

$$f(u) = C(u)\sum_{x=0}^{N-1} F(x)\cos\left(\frac{u(2x+1)\pi}{2N}\right) \quad \text{..........} \text{........}(3.9)$$

$$C(u) = \begin{cases} \sqrt{\frac{1}{N}} & if\, u \neq 0 \\ \sqrt{\frac{2}{N}} & if\, u = 0 \end{cases}$$

For simplification of IFDCT we can follow same procedure like that used with FDCT, where for the case N=8, the above IDCT equation could by simplified into equations (3.10):

$f(0)=(\sigma_0 F_0 + \sigma_1 F_1 + \sigma_2 F_2 + \sigma_3 F_3 + \sigma_4 F_4 + \sigma_5 F_5 + \sigma_6 F_6 + \sigma_7 F_7)/R$

$f(1)=(\varsigma_0 F_0 + \varsigma_1 F_1 + \varsigma_2 F_2 + \varsigma_3 F_3 + \varsigma_4 F_4 + \varsigma_5 F_5 + \varsigma_6 F_6 + \varsigma_7 F_7)/R$

$f(2)=(\upsilon_0 F_0 + \upsilon_1 F_1 + \upsilon_2 F_2 + \upsilon_3 F_3 + \upsilon_4 F_4 + \upsilon_5 F_5 + \upsilon_6 F_6 + \upsilon_7 F_7)/R$

$f(3)=(\varphi_0 F_0 + \varphi_1 F_1 + \varphi_2 F_2 + \varphi_3 F_3 + \varphi_4 F_4 + \varphi_5 F_5 + \varphi_6 F_6 + \varphi_7 F_7)/R$

$f(4)= (\psi_0 F_0 + \psi_1 F_1 + \psi_2 F_2 + \psi_3 F_3 + \psi_4 F_4 + \psi_5 F_5 + \psi_6 F_6 + \psi_7 F_7)/R$

$f(5)= (\alpha_0 F_0 + \alpha_1 F_1 + \alpha_2 F_2 + \alpha_3 F_3 + \alpha_4 F_4 + \alpha_5 F_5 + \alpha_6 F_6 + \alpha_7 F_7)/R$

$f(6)= (\beta_0 F_0 + \beta_1 F_1 + \beta_2 F_2 + \beta_3 F_3 + \beta_4 F_4 + \beta_5 F_5 + \beta_6 F_6 + \beta_7 F_7)/R$

$f(7)= (\delta_0 F_0 + \delta_1 F_1 + \delta_2 F_2 + \delta_3 F_3 + \delta_4 F_4 + \delta_5 F_5 + \delta_6 F_6 + \delta_7 F_7)/R$

Where

$\sigma i = \text{round}(C(i)\times COS((u\times \pi)/16) \times R)$

$\varsigma i = \text{round}(C(i) \times COS((3\times u\times\pi)/16) \times R)$

$\upsilon i= \text{round}(C(i) \times COS((5\times u\times\pi)/16) \times R)$

$\varphi i= \text{round}(C(i) \times COS((7\times u\times\pi)/16) \times R)$

$\psi i= \text{round}(C(i) \times COS((9\times u\times\pi)/16) \times R)$

$\alpha i= \text{round}(C(i) \times COS((11\times u\times\pi)/16) \times R)$

$\beta i= \text{round}(C(i) \times COS((13\times u\times\pi)/16) \times R)$

$\delta i= \text{round}(C(i) \times COS((15\times u\times\pi)/16) \times R)$

And we obtain that:

$U1=F_0+F_7$

$U2= F_3+ F_4$

$U3=F_1+F_6$

$U4=F_2+F_5$

V1=U1+U2

V2=U3+U4

V3=U1-U2

V4=U3-U4

V5=F_0-F_7

V6=F_1-F_6

V7=F_2-F_5

V8=F_3-F_4

Where this will make the system (3.11) as the following:

$f(0) = \sigma_0 (V1 + V2)/R$

$f(1) = [\varsigma_0 V5 + \varsigma_1 V6 + \varsigma_2 V7 + \varsigma_3 V8]/R$

$f(2) = [\upsilon_0 V3 + \upsilon_1 V4]/R$

$f(3) = [\varsigma_1 V5 - \varsigma_3 V6 - \varsigma_0 V7 - \varsigma_2 V8]/R$

$f(4) = \psi_0 (V1 - V2)/R$

$f(5) = [\varsigma_2 V5 - \varsigma_0 V6 + \varsigma_3 V7 + \varsigma_1 V8]/R$

$f(6) = [\upsilon_1 V3 - \upsilon_0 V4]/R$

$f(7) = [\varsigma_3 V5 - \varsigma_2 V6 + \varsigma_1 V7 - \varsigma_0 V8]/R$

The Inverse Fast DCT (IFDCT)

```
Algorithm (3.4) The Inverse Fast DCT (IFDCT)
Input: DCT Block (DB) 8×8
Output: Gray Block (GB) 8×8
For i=0 to 7
    • Load DCT vector from the ith row of DB, F(0).. F(7)
    • Compute shared variables, V1..V8
    • Compute DCT coefficients σ0, ς0, ς1, ς2, ς3, υ0, υ1, ψ0
    • Compute DCT vector, ƒ(0).. ƒ(7) using system (3.11)
    • For r=0 to 7: TMP(r, i)= ƒ(r): next r
 Next i
For i=0 to 7
    • Load TMP vector from the ith column of TMP Block, F(0).. F(7)
    • Compute shared variables, V1..V8
    • Compute DCT coefficients σ0, ς0, ς1, ς2, ς3, υ0, υ1, ψ0
    • Compute DCT vector, ƒ(0).. ƒ(7) using system (3.11)
    • For r=0 to 7: GB(i, r)= ƒ(r): next r
 Next i
```

3.4.7 Quantization and Coding

FDCT-based image compression relies on two techniques to reduce the data required to represent the image, the first is the ***quantization*** of the image's FDCT coefficients; the second is ***coding*** the quantized coefficients.

Quantization is the process of reducing the number of possible values of a quantity, thereby reducing the number of bits needed to represent it. Coding is a technique for representing the quantized data as compactly as possible.

In this proposed work, a function to determine the quantization step used to quantize DCT coefficients. A uniform quantization was adopted. The quantization step (Q_{step}) for each coefficient C(u, v) was determined by using the function:

$$Q_{step}(u, v)=1+\varphi\times(u+v) \quad \ldots\ldots\ldots\ldots\ldots(3.12)$$

Where φ is a quality factor of range (2…8).

So, the quantization is determined by using the following equation:

$$Q_I(u,v) = round\left(\frac{C(u,v)}{Q_{step}(u,v)}\right) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(3.13)$$

In this work the Zigzag ordering process, Run length, and S-Shift coding a compressed data for the Anchor frame. The Zigzag process is the conversion of 2D quantized FDCT coefficients to one-dimensional vector but in Zigzag way, figure (3.5) illustrates the followed zigzag path.

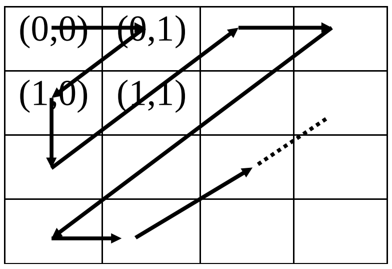


Figure (3.5) Zigzag Process

Image Quantization and Zigzag

Algorithm (3.5) Image Quantization and Zigzag

Input: Image (Img) H×W
Output: Zigzag vector (ZV)

```
Zindex=0  // Zigzag index
HB=H/8: WB=W/8 // Number of Blocks
For i=0 to HB-1
   For j=0 to WB-1
                          // Quantization
        Ib=i*8: Jb=j*8
        For r=0 to 7
              Ip=Ib+r
              For c=0 to 7
                   $Q_I$(Ip, Jb+c)= round(Img (Ip, Jb+c)/ $Q_{step}$ (r, c))
              Next c
         Next r
                          // Zigzag
       r=0: c=0: ZV(Zindex)= $Q_I$(Ib, Jb)
       For it=1 to 3
              r=0
              For k=c to 0  step -1
                    Zindex=Zindex+1
                    ZV(Zindex)= $Q_I$(Ib+r, Jb+k): r=r+1
              Next k
              c=0
              For k=r to 0  step -1
                    Zindex=Zindex+1
                    ZV(Zindex)= $Q_I$(Ib+k, Jb+c): c=c+1
              Next k
         Next it
         r=0
         For it=1 to 4
              For k=r to 7
                    Zindex=Zindex+1
                    ZV(Zindex)= $Q_I$(Ib+k, Jb+j): c=c+1
              Next k
              c=c+2: r=7
              For k=c to 7
                    Zindex=Zindex+1
                    ZV(Zindex)= $Q_I$(Ib+r, Jb+k): r=r+1
              Next k
              r=r+2: c=k-1
         Next it

    Next j
Next i
```

Run Length Encoder

Algorithm (3.6) Run Length Encoder

Input: One-dimensional zigzag array ZA of length Zmax
Output: One-dimensional array RA of length Rl

```
• Set Rl=0
• Set Zcount=0

While Zcount < Zmax

    While ZA (Zcount) <> 0
        RA (Rl) = ZA (Zcount)
        Zcount = Zcount + 1
        Rl=Rl+ 1
    Wend
    Count = 0
    While ZA (Zcount) = 0
        Count = Count + 1
        Zcount = Zcount + 1
    Wend
    RA (Rl) = 0
    RA (Rl+ 1) = Count
    Rl=Rl+ 2
Wend
```

3.4.8 Shift Key Encoder

The input data is an array of run length pairs (run and value), where the shift key encoder will encode each cell separately, and the encoded data is registered in Wb buffer and each eight bits in this Wb buffer will converted to byte and saved in the output buffer.

Shift Key Encoder algorithm

Algorithm (3.7) Shift Key Encoder

Input: One-dimensional coded array RA of length Rl
Output: Coded data of zeros and ones

```
  • Set M equal to the max codeword // Wb

For i=1 to Rl
    • Set X=RA (i)
    • If X>0 then
          S=1
          add S value to the output buffer as single bit
      else
          S=0
          add S value to the output buffer as single bit
      end if
    • X=abs(X)

  While X >M

        • Save M in the output buffer as single codeword
        • X=X-M

   Wend
   If X>0 then
          add X bits to the output buffer as single codeword
     else
          add zero bits to the output buffer as single codeword
   end if

Next i
```

3.4.9 Video Motion Estimation

As mentioned in chapter two, two standard methods (OTS and TSS) for motion estimation were implemented, also a new method for motion estimation was suggested, a hybrid method utilize both steps that followed the two standard methods (OTS, and TSS).

3.4.9.1 Hybrid Motion Estimation

The suggested Hybrid Method (HM) features are those of the two standard methods. As mentioned in chapter two, the two methods (OTS, TSS) has three stages for completing the block search, in the suggested Hybrid Method (HM), there are also three stages, the steps of first stage in the HM is the steps of 1st stage of TSS followed by the steps of the first stage of OTS, and a similar procedure is followed in the second and third stages of HM. After doing the first stage of TSS, the suggested HM will complete its stage by implementing the steps of the first stage of OTS method for searching block position in horizontal direction. While in the second stage of HM the first stage of TSS is followed by implementing the steps of OTS to do searching in the vertical direction, and finally after the final stage of TSS is completed OTS searching in the diagonal direction will performed.

3.4.9.2 Computing Motion Estimation and Coding

Figure (3.6) illustrates the general scheme of the motion estimation with the coding technique.

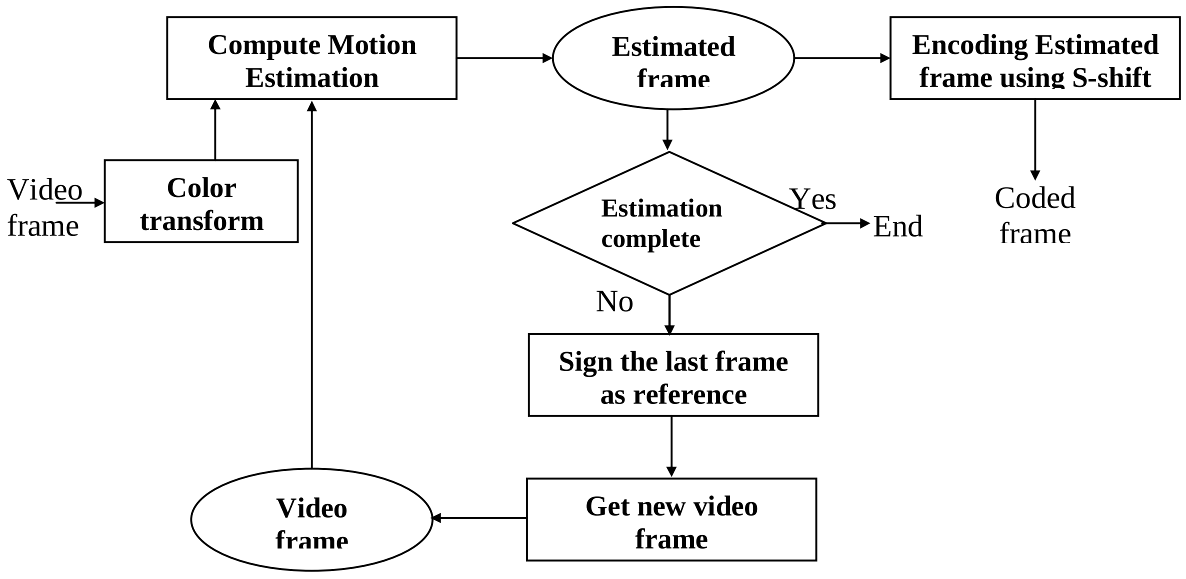


Figure (3.6) General Scheme of Motion Estimation and Coding

Where

1. At first the system will transform the video frame to its YUV components, where the estimation is implemented on Y band only because this Y band represents 90% of the whole information.
2. Then the system computes the Motion Estimation of the video frame depending on the motion estimation technique selected by the user. The first computation of motion estimation depends on the Anchor frame as a reference frame for computing the shift (motion vector) of the estimated frame.
3. For each estimated frame the S-shift encoder will applied encode the estimated frame (see algorithm 3.7).
4. If the number of estimated frames reaches five frames, the system will finish its motion for the input group of frames, and then the system will replace the reference (Anchor) frame and start the motion estimation for the next groups of frames.
5. Finally we will get a new video frame to be estimated.

OTS Motion Estimation Method

The algorithm list (3.8) illustrates the steps of implementing OTS motion estimation method.

Algorithm (3.8) OTS Motion Estimation

```
Input: Video frame of H × W
Output: Estimated frame

    • Set D equal to the dimension of the single block
    • Set NBH= H div D
    • Set NBW= W div D

  For r=0 to NBH-1  step D

    For c=0 to NBW-1  step D

      • Compute the absolute difference (AD) for block whose control coordinates
        are (r, c+1), (r, c-1) in the reference frame
          - If AD of (r, c+1) is smaller than AD of (r, c-1) then
                Continue searching for smaller AD in the right hand side of
                the original block (r,c)
            Else
                Continue searching for smaller AD in the left hand side of
                the original block (r,c)
      • Compute the absolute difference (AD) for block whose control coordinates
        are (r+1, c), (r-1, c) in the reference frame
          - If AD of (r+1, c) is smaller than AD of (r-1, c) then
                Continue searching for smaller AD in the Down ward of the
                original block (r,c)
            Else
                Continue searching for smaller AD in the Up ward of the
                original block (r,c)
      • Searching smaller AD on diagonal way

  Next c

 Next r
```

TSS Motion Estimation Method

The algorithm list (3.9) illustrates the steps of implementing TSS motion estimation method.

Algorithm (3.9) TSS Motion Estimation

```
Input: Video frame of H × W
Output: Estimated frame

        • Set D equal to the dimension of a single block
        • Set NBH= H div D
        • Set NBW= W div D

  For r=0 to NBH-1  step D

    For c=0 to NBW-1  step D

          • Loop searching smaller AD on the eight neighbors surrounding the
            center (r, c) with offset shift equal to four pixels.
          • Loop searching smaller AD on the eight neighbors for the new
            coordinate of the last small AD with offset equal to two pixels.
          • Loop searching smaller AD on the eight neighbors for the new
            coordinate of the last small AD with offset equal to one pixel.
    Next c
  Next r
```

HM Motion Estimation

The algorithm list (3.10) illustrates the steps of implementing HM motion estimation method.

Algorithm (3.10) HM Motion Estimation

```
Input: Video frame of H X W
Output: Estimated frame

    • Set D equal to the dimension of a single block
    • Set NBH= H div D
    • Set NBW= W div D

  For r=0 to NBH-1  step D
     For c=0 to NBW-1  step D
     • Loop searching smaller AD on the eight neighbors for the coordinate (r, c)
       with offset equal to four pixels.
     • Compute absolute difference (AD) for coordinates (nr, nc+1), (nr, nc-1)
                 -  If AD of (nr, nc+1) is the smaller then
                        Continue searching for smaller AD right ward
                    Else
                        Continue searching for smaller AD left ward
     • Loop searching for smaller AD on the eight neighbors for the new coordinate
       of the last AD with offset equal to two pixels.
     • Compute absolute difference (AD) for coordinates (nr+1, nc), (nr-1, nc)
                 -  If AD of (nr+1, nc) is the smaller then
                        Continue searching for smaller AD downward
                    Else
                        Continue searching for smaller AD upward
     • Loop searching for smaller AD on the eight neighbors for the new coordinate
       of the last AD with offset equal to one pixel.
     • Searching for smaller AD in diagonal direction

     Next c
  Next r
```

3.5 Video Decoding Using FDCT (Model1)

Figure (3.7) illustrates the major scheme of video decoding system.

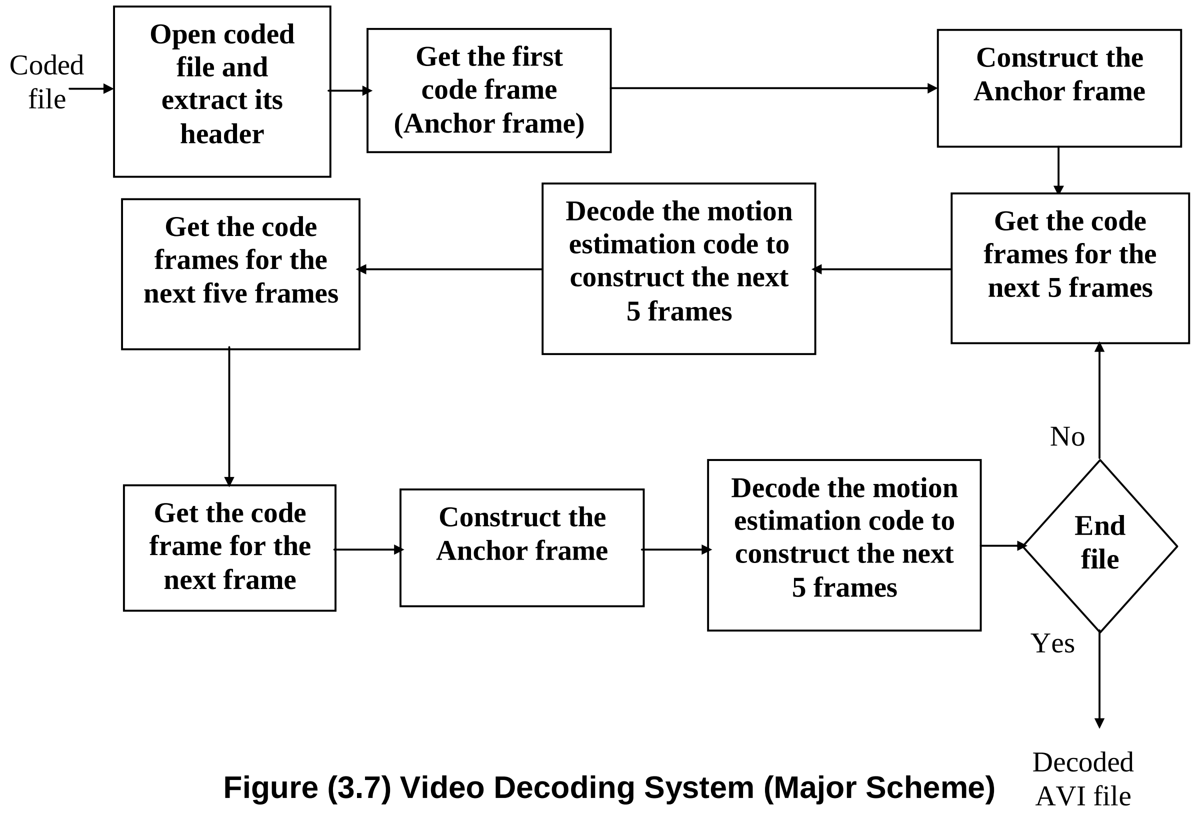

Coded file
Open coded file and extract its header
Get the first code frame (Anchor frame)
Construct the Anchor frame
Get the code frames for the next 5 frames
Decode the motion estimation code to construct the next 5 frames
Get the code frames for the next five frames
Get the code frame for the next frame
Construct the Anchor frame
Decode the motion estimation code to construct the next 5 frames
End file
No
Yes
Decoded AVI file

Figure (3.7) Video Decoding System (Major Scheme)

Video Decoding System

The algorithm list (3.11) shows the implemented major stages of the decoding modules.

Algorithm (3.11) Video Decoding System

```
Input: Coded video file
Output: Decoded AVI file

    • Open coded file and extract its header
    • Get code frame (for the Anchor frame)
    • Construct the Anchor frame
    While (Not end of file)
        • Get the code frames for the next 5 frames
        • Decode the motion estimation code to construct the next 5 frames
        • Get the code frames for the next 5 frames
        • Get code frame (for the Anchor frame)
        • Construct the Anchor frame
        • Decode the motion estimation code to construct the next 5 frames
    Wend
```

3.5.1 Decode Anchor Frame Using IFDCT

Figure (3.8) illustrates the decoding scheme for Anchors frame using FDCT.

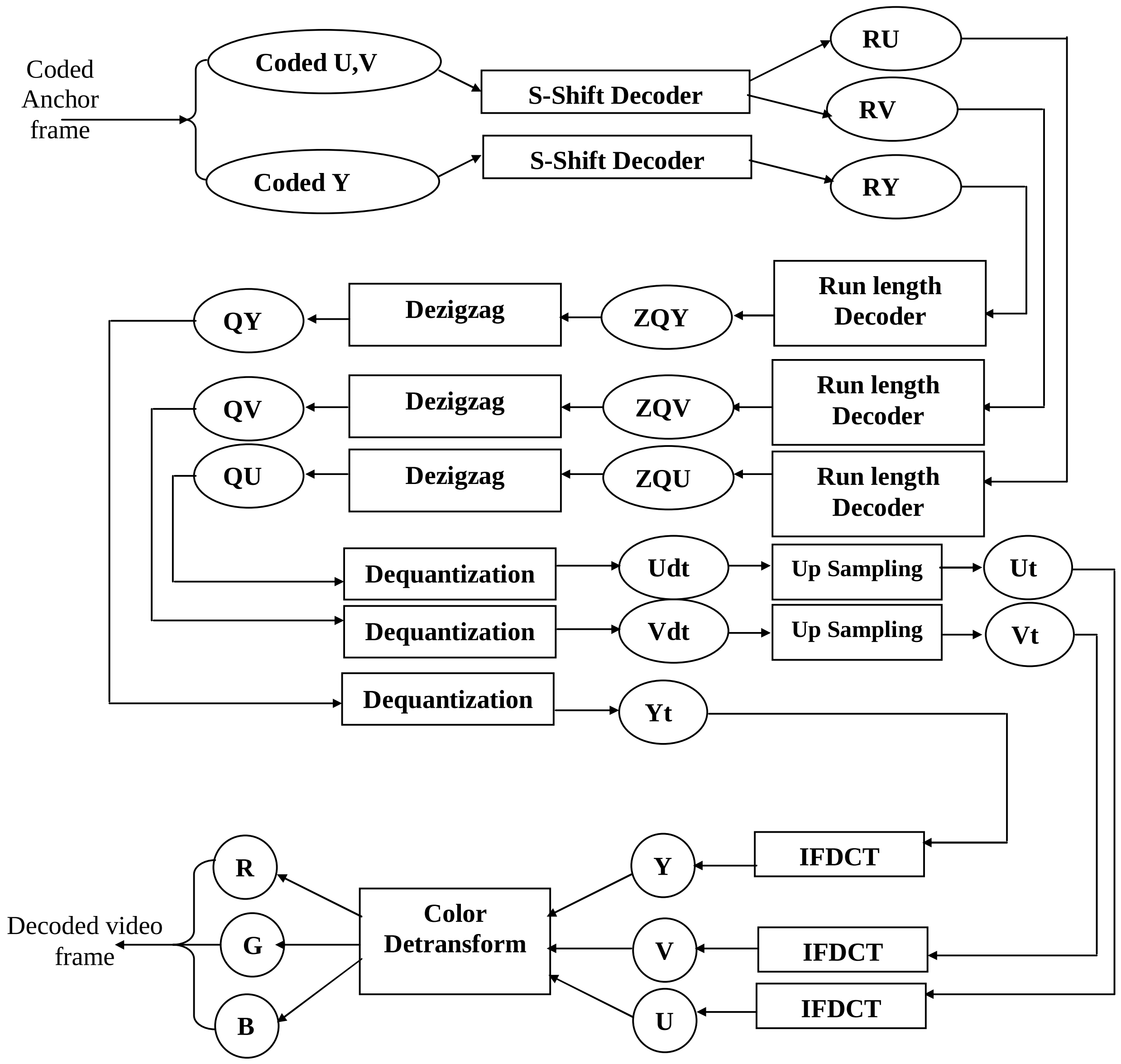


Figure (3.8): Decode Anchor Frame Using FDCT

Shift Key Decoder

The algorithm list (3.12) illustrates the implementation steps of the shift key decoding stage.

Algorithm (3.12) Shift Key Decoder

```
Input: Binary sequence of zeros and ones (the shift key codes)
Output: One-decimal constructed array RA of length Rl

  • Set M equal to the max sample (M=7 in this implementation)
  • Set Nbit equal to the number of bits for M (3 bits in this case)
  For i=1 to Rl
      • Get S= the first bit of the binary sequence // S means sign bit
      • Set R=0
      • Set V= the value of the first Nbit of binary sequence
      • Shift the binary sequence Nbit bits to the left
      • While V=M
          • R=R+V
          • Set V= the value of the first Nbit of binary sequence
          • Shift binary sequence Nbit bits to the left
      • Wend
      • If S=0 then
              RA(i)=R
          Else
              RA(i)=-R
  Next i
```

Run Length Decoder

The algorithm list (3.13) illustrates the implementation steps of the run length decoding stage.

Algorithm (3.13) Run length Decoder

Input: One-dimensional array RA of length Rl

Output: One-dimensional zigzagged array ZA of length Zmax

```
•  Set Rcount=0
•  Set Zcount=0
While Rcount < Rl
    If RA(Rcount)=0 then
        N=RA(Rcount +1)
        For i=1 to N
            ZA(Zcount)=0
            Zcount=Zcount+1
        Next i
        Rcount=Rcount+2
    Else
        ZA(Zcount)=RA(Rcount)
        Zcount=Zcount+1
        Rcount=Rcount+1
    End if
Wend
```

Image Dequantization and Dezigzag

The algorithm list (3.14) illustrates the implementation steps of the dezigzag and dequantization stage.

Algorithm (3.14) Image Dequantization and Dezigzag

Input: Zigzag vector (ZV)
Output: Image (Img) H×W

```
Zindex=0                          // Zigzag index
HB=H div 8: WB=W div 8     // Number of Blocks
For i=0 to HB-1
    For j=0 to WB-1

        Ib=i*8: Jb=j*8
                                      // Dezigzag
        r=0: c=0: $Q_I$(Ib, Jb)=ZV(Zindex)
        For it=1 to 3
            r=0
            For k=c to 0  step -1
                Zindex=Zindex+1
                $Q_I$(Ib+r, Jb+k)=ZV(Zindex): r=r+1
            Next k
            c=0
            For k=r to 0  step -1
                Zindex=Zindex+1
                $Q_I$(Ib+k, Jb+c)=ZV(Zindex): c=c+1
            Next k
        Next it
        r=0
        For it=1 to 4
            For k=r to 7
                Zindex=Zindex+1
                $Q_I$(Ib+k, Jb+j)=ZV(Zindex): c=c+1
            Next k
            c=c+2: r=7
            For k=c to 7
                Zindex=Zindex+1
                $Q_I$(Ib+r, Jb+k)=ZV(Zindex): r=r+1
            Next k
            r=r+2: c=k-1
        Next it
                                          // Dequantization

        For r=0 to 7
            Ip=Ib+r
            For c=0 to 7
                Img (Ip, Jb+c)=round($Q_I$(Ip, Jb+c)* $Q_{step}$ (r, c) )
            Next c
        Next r
    Next j
Next i
```

3.5.2 The Image Upsampling

The goal of this algorithm is to enlarge the image 4 times, and this goal was done by duplicating each pixel in the image four times.

The Upsampling

The algorithm list (3.15) illustrates the implementation steps of upsampling both U and V color components.

Algorithm (3.15) The Upsampling

```
Input: UV Image (UVimg) Dh × Dh
Output: YUV Image (img) H × W

    • H=Dh × 2
    • W=Dw × 2
 For i=0 to Dh-1: Id=i×2: Idp= Id+1
    For j=0 to Dw-1
    • Jd=j×2
    • img(Id, Jd).U=Uvimg(Id,Jd).U: img(Id, Jd).V=Uvimg(Id,Jd).V
    • img(Idp, Jd).U=Uvimg(Id,Jd).U: img(Idp, Jd).V=Uvimg(Id,Jd).V
    •  img(Id, Jd+1).U=Uvimg(Id,Jd).U: img(Id, Jd+1).V=Uvimg(Id,Jd).V
    • img(Idp, Jd+1).U=Uvimg(Id,Jd).U: img(Idp, Jd+1).V=Uvimg(Id,Jd).V

    Next j
 Next i
```

The Color Detransformation

Algorithm (3.16) The Color Detransformation

```
Input: YUV Image (Rimg) H×W
Output: RGB Image (Img) H×W
For i=0 to H-1
   For j=0 to W-1
      • Compute R band from YUVImg(i, j), $R=Y+1.140*V$
      • Compute G band from YUVImg(i, j), $G=Y–(395*U-581*V)*0.001$
      • Compute B band from YUVImg(i, j), $B=Y+2.032*U$
   Next j
Next i
```

3.5.3 Extract Motion Estimation with Decoding

Figure (3.9) illustrates the scheme of extracting motion estimation and the scheme of its decoding.

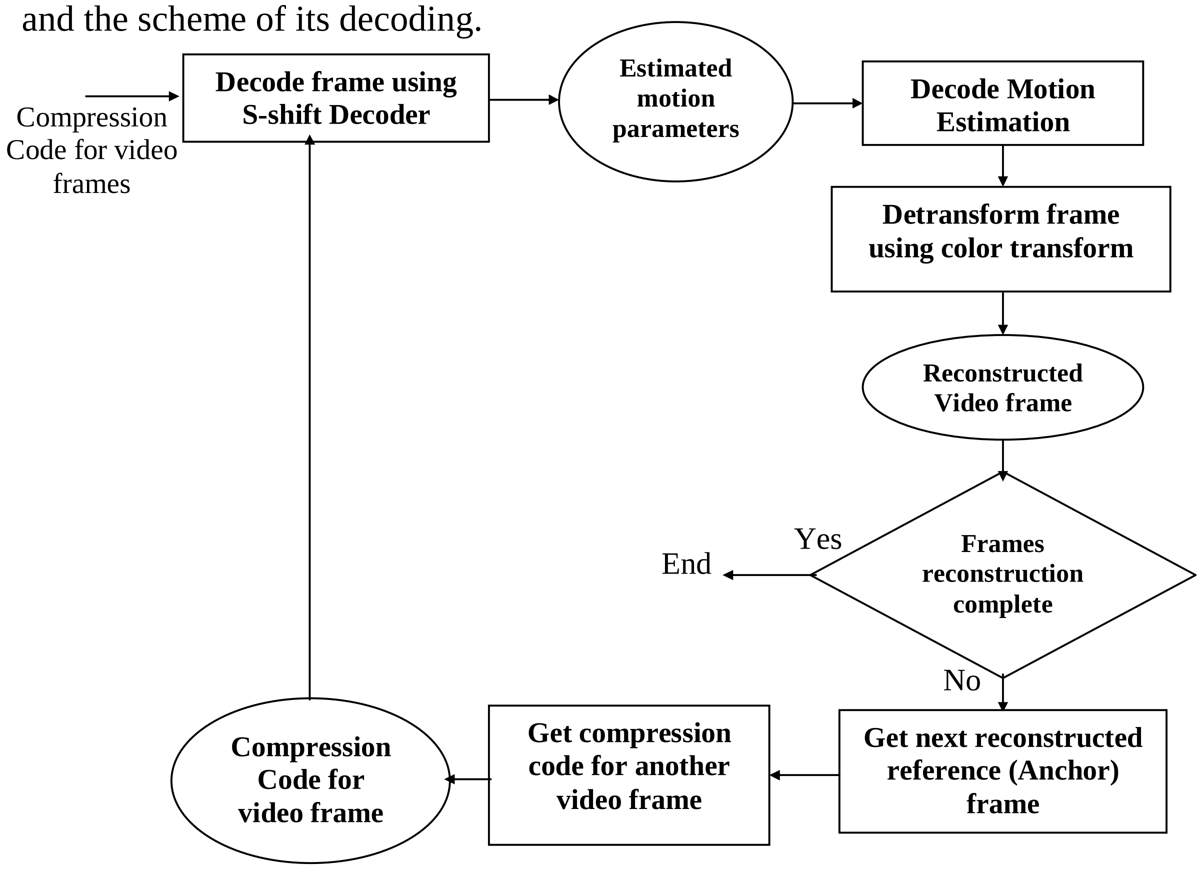

Compression Code for video frames
Decode frame using S-shift Decoder
Estimated motion parameters
Decode Motion Estimation
Detransform frame using color transform
Reconstructed Video frame
Frames reconstruction complete
Yes
End
No
Get next reconstructed reference (Anchor) frame
Get compression code for another video frame
Compression Code for video frame

Figure (3.9) Extract Motion Estimation with Decoding

Decode Motion Estimation

Algorithm (3.17) Decode Motion Estimation

Input: Estimated frames (Ef) of Dw × Dh blocks each block of Δx, Δy coordinates, and reference frame (Rf) of H × W.

Output: Transformed video frame (Tf) of H × W pixels (YUV)

```
• Set N= the Block dimension
Rdh=Dh-1: Rdw=Dw-1: No=N-1
For i = 0 To Rdh
  For j = 0 To Rdw
     Id=i*N: Jd=j*N
    For r = 0 To No
       Idp=Id+r: Rd= Id+Ef(i, j). Δx+r: Cd=Jd+ Ef(i, j). Δy
      For c = 0 To No
        • Tf(Idp, Jd + c).Y = Rf(Rd, Cd+c).Y
        • Tf(Idp, Id + c).U = Rf(Rd, Cd+c).U
        • Tf(Idp, Id + c).V = Rf(Rd, Cd+c).V
      Next c
    Next r
  Next j
Next i
```

3.6 The System Structure of Model2 Coding

Figure (3.10) illustrates the general scheme of Model-2 (Fractal).

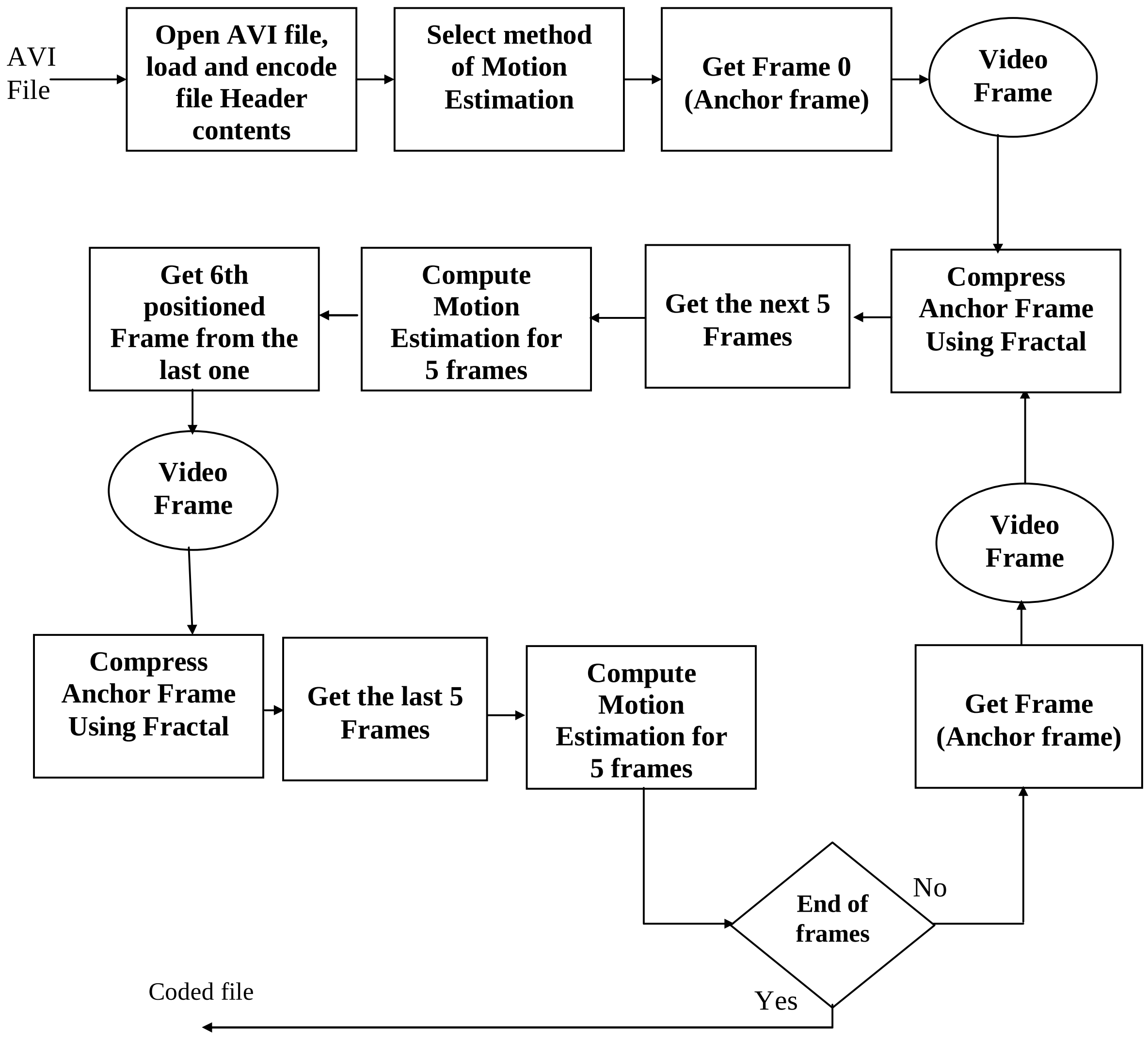

AVI File
Open AVI file, load and encode file Header contents
Select method of Motion Estimation
Get Frame 0 (Anchor frame)
Video Frame
Compress Anchor Frame Using Fractal
Get the next 5 Frames
Compute Motion Estimation for 5 frames
Get 6th positioned Frame from the last one
Video Frame
Compress Anchor Frame Using Fractal
Get the last 5 Frames
Compute Motion Estimation for 5 frames
End of frames
No
Yes
Get Frame (Anchor frame)
Video Frame
Coded file

Figure (3.10) The System Structure Model2 Coding

In this model the same strategy that followed in model1 is used, but the difference between them is that the compression system of the Anchor frame coding is implemented using the fractal coding.

3.6.1 Compress Anchor Frame Using Fractal Coding

In this work, the video frame first is transformed from RGB bands to YUV bands using Color Transform (CT) (algorithm list 3.1), then both the U and V bands are downsampled by 2 using average method. The Y band is remained without downsampling because it represents 90% from the total information of the original video frame, where any reduction on this band will affect the quality of the original video frame. Figure (3.11) illustrates the scheme of Anchor Frame compression using fractal method. Where Y, Ut, and Vt are the range blocks that represent the input bands for the Fractal compression machine, and the result represent the compressed Anchor frame.

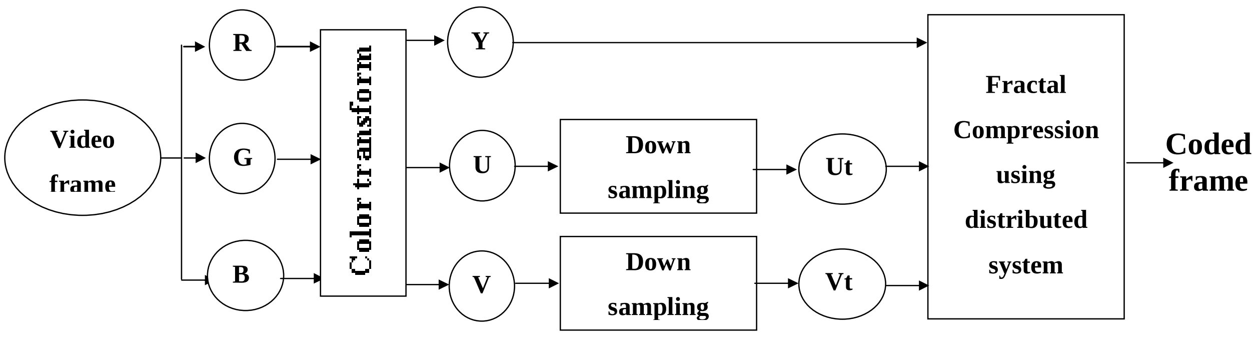


Figure (3.11) Anchor Frame Compression Using Fractal Coding

- **Down sampling:** in this process the input band will be reduced to ¼ its size (number of pixels), where each four pixels values will reduced to their one average value (see algorithm 3.2).

3.6.2 Fractal Compression Using Distributed System

In this process the input band, the range, will be partitioned into a number non-overlapped blocks, and the same range will be down sampled to create the domain that will be also partitioned to number of blocks, each block have the same size of the range block, then execute the matching technique. The matching technique will match each range block with all domain blocks to find the best domain block that approximate the range block using the affine transform with eight symmetry cases, and finally record the scale, offset, symmetry, and the x, y coordinates of the best matched domain block, (see figure (3.14)).

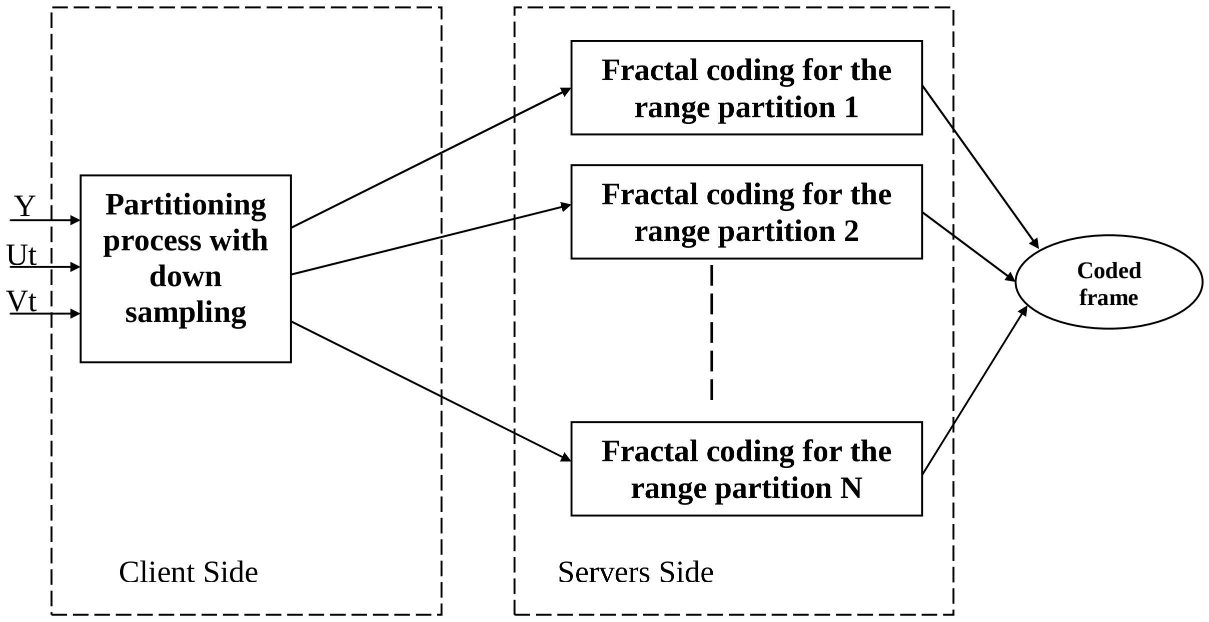

Y
Ut
Vt
Partitioning process with down sampling
Fractal coding for the range partition 1
Fractal coding for the range partition 2
Fractal coding for the range partition N
Coded frame
Client Side
Servers Side

Figure (3.12) Fractal Compression using Partitioning System

The problem with the fractal coding is the expensive time needed to complete its computation; the distributed system is one of the solutions that could be used to speed up the fractal coding. The distributed system was adopted in this work to solve this problem. One of the computers will be responsible for computing the domain blocks, divide the range blocks and send the partitions to other computers, where each of these computers will perform the required matching tasks on the blocks of the received range partition.

3.6.3 Network Communication and Network Model

The communication between the client and servers is done through the Socket Interprocess communication. In this work the developed distributed system is based on the star model as shown in figure (3.13), where single client computer exist, while other computers are utilized as servers.

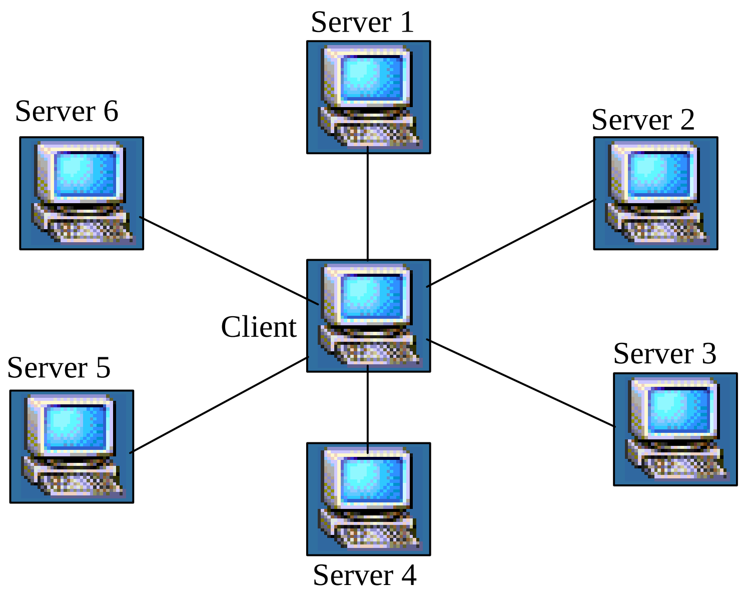


Figure (3.13) Distributed system based on star model

3.6.4 Fractal Coding for One Partition

The first step in this scheme is to partitioning the range into a number of non-overlapped blocks, the same step will be done with the domain, where the domain is also partitioned into a number of non-overlapped blocks, after the partitioning process the algorithm will start the matching stage (see figure 3.14). The output results of the matching process (i.e. S, O, Sym, x, and y coordinates) are quantized using fractal quantization (see algorithm list 3.19). Finally the quantized factors will be coded using Bitwise coding technique (see algorithm list 3.20).

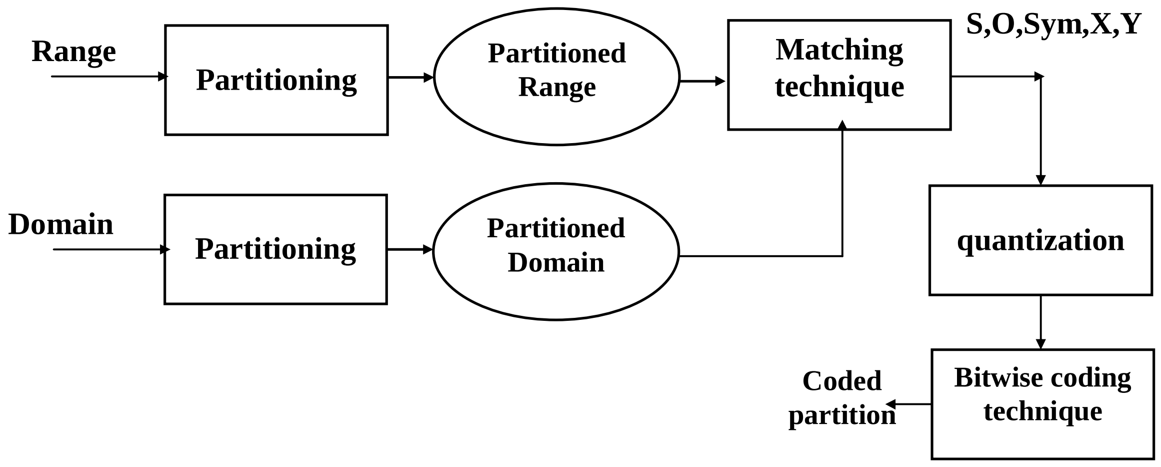


Figure (3.14) Fractal coding using one range partition

Fractal Matching Technique

The algorithm list (3.18) illustrates the implemented steps of the fractal matching stage.

Algorithm (3.18) Fractal Matching technique

```
Input: Rb is the array of range blocks, Rn is the number of range blocks, Db is
       the array of domain blocks, Dn is the number of domain blocks.
Output: S, O, x and y coordinates of each Range Block.

    • TH is a fractal threshold  // TH=6 gives an acceptable quality
    For i = 1 To Rn
        • Rb[i] is the ith range block.

        For j = 1 To Dn

            • Db[j] is the jth domain block.
            For Sym=1 to 8
                • Set D as the Sym affine transform of Db[j] using
                  equations (3.14…3.21).
                • Compute the S of (Rb[i], D) using equation (2.3).
                • Compute the O of (Rb[i], D) using equation (2.4).
                • Compute the Error between (Rb[i], D) using equation
                  (2.5).
                • If the error < TH then exit loops
    Next s
  Next j
Next i
    • Output the S, O, Sym x, and y of the Domain Block with smaller Error.
```

3.6.5 Affine Transform (Symmetry Cases)

In this work the following eight symmetry cases were implemented:

1. **Identity case:** $\Gamma 1(x, y)=R(x, y)$ ……….…………….…(3.14)
2. **Rotation 90:** $\Gamma 2(x, y)=R(S-y, x)$ …………..…………(3.15)
3. **Rotation 180:** $\Gamma 3(x, y)=R(S-x, S-y)$ ……..…..………(3.16)
4. **Rotation 270:** $\Gamma 4(x, y)=R(y, S-x)$ ………………..…(3.17)

5. **Reflection:** Γ5(x, y)=R(S-x, y) …………………..(3.18)
6. **Reflection and Rotation 90:** Γ6(x, y)=R(S-y, S-x) …(3.19)
7. **Reflection and Rotation 180:** Γ7(x, y)=R(x, S-y) ….(3.20)
8. **Reflection and Rotation 270:** Γ8(x, y)=R(y, x) …….(3.21)

Where S is the Block Size.

3.6.6 Fractal Quantization

Reducing the number of possible values of the fractal factors (i.e. scale, offset coefficients) is very important to get compression, so a uniform quantization was performed to quantized the scale and offset coefficients, also quantization was used to reduce the number of bits required to encode the coordinates of the domain blocks, where the jump step value was used as the quantization step. The scale coefficient was quantized by using the following equation.

$$S_q = round\left(\frac{S}{S_{Max}} \times \frac{2^n - 1}{2}\right), \ldots\ldots\ldots\ldots\ldots\ldots(3.22)$$

Where ***n*** is the number the bits assigned to encode the scale coefficient, and **S*Max*** is the maximum allowed value for scale. The offset coefficient was quantized using the following equation.

$$O_q = round\left(\frac{O - O_{Min}}{O_{Max} - O_{Min}} \times (2^m - 1)\right), \ldots\ldots\ldots\ldots\ldots\ldots(3.23)$$

Where ***m*** is the number of bits, **O*Max*** is the maximum allowed offset value, and **O*Min*** is the minimum allowed offset value. While the X, and Y coordinates will quantized using the following equation.

$$C_q = \frac{C}{S_j} \text{ ,..........................(3.24)}$$

Where $\boldsymbol{S_j}$ is the jump Size, and $\boldsymbol{C}$ is either the x or y coordinate, $\boldsymbol{C_q}$ is the correspond quantization index.

Fractal Quantization

The algorithm list (3.19) illustrates the steps of quantization of the fractal coefficients.

Algorithm (3.19) Fractal Quantization

```
Input: Fb[] is a single array of length Fn of fractal blocks coefficients (S, O, and (X, Y)
       coordinates).
Output: QFb[] is a single array of length Fn of fractal blocks quantized coefficients ($S_q$,
       $O_q$, and ($X_q$, $Y_q$) coordinates.
 For i = 1 To Rn
     • Fb[i] is the ith fractal Block coefficients.
     • Compute the Quantized scale ($S_q$) of QFb[i] from Fb[i] using equation (3.22).
     • Compute the Quantized offset of QFb[i] from Fb[i] using equation (3.23).
     • Compute the Quantized ($X_q$, $Y_q$) coordinates of QFb[i] from Fb[i] using equation
       (3.24).
Next i
```

Bitwise Coding Technique

The algorithm list (3.20) shows the steps of saving the fractal coefficients in the storage buffer.

Algorithm (3.20) Bitwise Coding

```
Input: Qn is the number of quantized S, O coefficients, (X, Y) coordinates, and
       Sym (symmetry) case.
Output: Rn coded buffer of 4B size.

For i = 1 To Qn
    • Q[i] is the ith Quantized Block.
    • St[i] is the ith storage buffer of 4B size.
    • Encode X, and Y coordinates in the storage buffer as a sequence of 7
      bits for each, respectively.
    • Encode Symmetry case in the storage buffer as a sequence of 3 bits.
    • Encode the S factor in the storage buffer as a sequence of 7 bits.
    • Encode the O factor in the storage buffer as a sequence of 8 bits.
Next i
```

3.7 Video Decoding Using Fractal (Model2)

The decoder system for model2 follow the same strategy of the decoder of model1 (see figure 3.7), but the difference is that the decoding of the Anchor frame uses the fractal decoding. The major scheme of fractal decoding is shown in figure (3.15).

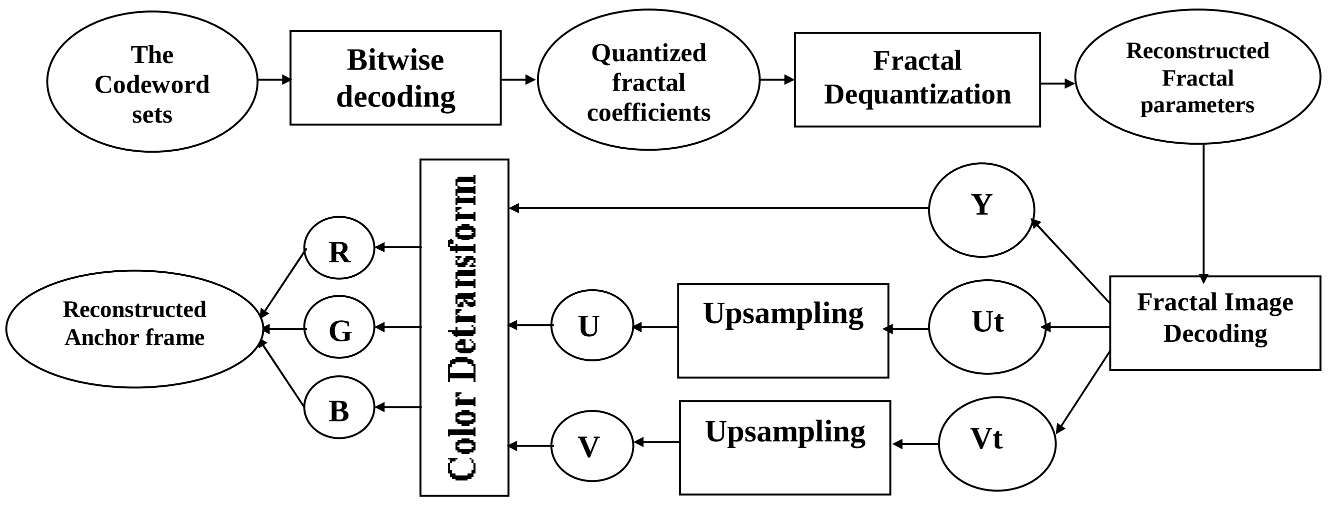


Figure (3.15) Anchor Frame Decoding Using Fractal Technique

At first read the codeword sets from the compression stream and decode them using Bitwise decoder (see algorithm list 3.21), where the results of this decoding stage are the quantization indices of the fractal coefficients (i.e. S, O, Sym, X, and Y), the reconstructed fractal coefficients are produced by using Dequantization process (see algorithm list 3.22). The results of the dequantization are the sets of reconstructed (quantized) fractal coefficients. Each set consist of the fractal coefficients for the three compressed bands Y, and downsampled Ut and Vt bands, See figure 3.15.

- **Reconstruction of Video Frame:** this process will reconstruct the image bands (Y, Ut, and Vt) by applying the affine transform using the quantized fractal coefficients, see figure 3.16.
- **Upsampling:** In this process the Ut and Vt are upsampled to produce $\overline{U}$ and $\overline{V}$ bands, where each value in Ut and Vt bands will be duplicated four times.
- **Color Detransform:** In this detransform the $\overline{Y}$, $\overline{U}$, and $\overline{V}$ bands will be detransformed to the $\overline{R}$, $\overline{G}$, and $\overline{B}$ bands to produce the video frame (see algorithm list 3.16).

Bitwise Decoder

The algorithm list (3.21) shows the steps of extracting the fractal coefficients from the storage buffer.

```
Algorithm (3.21) Bitwise Coding

Input: Qn is the number of coded storage buffer of 4B size.
Output: Qn is the number of decoded (quantized) S, O coefficients, (X, Y)
        coordinates, and Sym (symmetry) case.

For i = 1 To Qn
    • Get St[i] is the ith storage buffer.
    • Q[i] is the ith decoded (quantized) S, O coefficients, (X, Y) coordinates,
      and Sym (symmetry) case.
    • Assemble the X, and Y coordinates indices using 7 bits (for each) loaded
      from the storage buffer.
    • Assemble the Symmetry index using 3 bits.
    • Assemble the quantization scale index using 7 bits.
    • Assemble the quantization offset index using 8 bits.
Next i
```

3.7.1 Dequantization of Fractal Coefficients

The reconstructed scale coefficients could produce by using the following equation, that can be concluded form equation (3.22).

$$S_r = S_q \times S_{Max} \times \frac{2}{2^n - 1} \text{ ,...................(3.25)}$$

Where ***n*** is the number of the bits used to encode the quantization scale coefficients, and **S*Max*** is the maximum allowed value for the scale coefficient. The Offset can be dequantized using equation (3.26) that concluded from equation (3.23).

$$O_r = O_q \times \frac{O_{Max} - O_{Min}}{2^n - 1} + O_{Min} \quad ,.............(3.26)$$

Where ***n*** is the number of the bits used to encode the quantization offset coefficients, **O*Max*** is the maximum allowed value for the offset coefficient, and **O*Min*** is the minimum allowed offset coefficient. While the X, and Y coordinates can be dequantized by using equation (3.27) which is concluded from equation (3.24).

$$C_r = C_q \times S \quad ,......... ....(3.27)$$

Where ***S*** is the Block Size.

Dequantization of Fractal Coefficients

The algorithm list (3.22) illustrates the steps of the fractal coefficients dequantization stage.

Algorithm (3.22) Dequantization of Fractal Coefficients

```
Input: Qn is the number of the quantization indices of the Scale, Offset, and (X, Y)
       coordinates.
Output: Qn is the number of the dequantized (reconstructed) Scale, Offset, and (X,
       Y) coordinates.

For i = 1 To Qn
    • Get Q[i] is the ith Range Block.
    • Compute the Dequantized Scale using equation (3.25).
    • Compute the Dequantized Offset using equation (3.26).
    • Compute the Dequantized (X, Y) coordinates using equation (3.27).
Next i
```

3.7.2 Fractal Image Decoding

Figure (3.16) shows the major scheme of the video frame reconstruction process by using fractal image decoding.

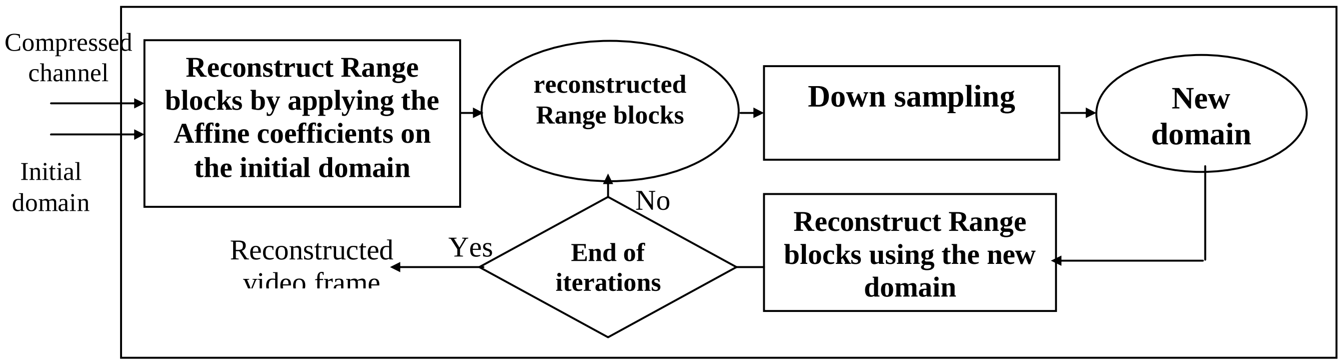

Compressed channel
Initial domain
Reconstruct Range blocks by applying the Affine coefficients on the initial domain
reconstructed Range blocks
Down sampling
New domain
Reconstruct Range blocks using the new domain
End of iterations
No
Yes
Reconstructed video frame

Figure (3.16) Video Frame Reconstruction using Fractal decoding

In this stage decode the reconstructed sets of affine (fractal) coefficients will applied on the domain pool (which will initialized arbitrarily) in an iterative manner for five iterations. At the first iteration the scheme will apply the affine transform using an initial domain which can be a 2D array of any numbers, where each pixel of x and y coordinates of the range block will be computed by using the following equation:

$$\mathbf{R}(x_r+x,\ y_r+y)=\boldsymbol{Sym}(\mathbf{D}(x+x_d,\ y+y_d))*\mathbf{S}+\mathbf{O}, \ldots\ldots\ldots\ldots\ldots\ldots (3.28)$$

Where

x_r, y_r are the coordinates of the top-left corner of the reconstructed block

x_d, y_d are the coordinates of the top-left corner of the domain block.

Sym**(D**(x+x_d, y+y_d)) is the pixel of (x+x_d, y+y_d) coordinates in the domain block after transforming it using inverse mapping for the *Sym* case, S is the scale value, and O is the offset value. The result of the whole affine reconstruction process will be downsampled to create a new domain, and then repeat the affine reconstruction again (but use the new domain) for five iterations (See figure 3.16).

3.7.3 Symmetry Decoding (Symmetry cases)

As above mentioned the fractal encoder had used eight symmetry cases, the decoder system had implemented the inverse of the symmetry cases, and as the following:

1. **Identity case:** R(x, y)= Γ(x, y). ……………….………(3.29)
2. **Rotation 90:** R(x, y)= Γ(y, S- x) ……………………….(3.30)
3. **Rotation 180:** R(x, y)= Γ(S-x, S-y) …………………….(3.31)
4. **Rotation 270:** R(x, y)= Γ(S-y, x) ……………………….(3.32)
5. **Reflection:** R(x, y)= Γ(S-x, y) ………………………..……(3.33)
6. **Reflection and Rotation 90:** R(x, y)= Γ(S-y, S-x) ……..(3.34)
7. **Reflection and Rotation 180:** R(x, y)= Γ(x, S-y) ………(3.35)
8. **Reflection and Rotation 270:** R(x, y)= Γ(y, x) …………(3.36)

Where S is the Block size.

Chapter Four
Tests and Results

4.1 Introduction

This chapter is developed to study the compression performance of the three searching methods of motion estimation (TSS, OTS, and Hybrid method) and the two different compression techniques of Anchor frames are investigated by performing set of suitable objective fidelity measures (such as MSE, PSNR, etc) on five different video sequences that are taken as testing samples. Each video sequence consists of different numbers of frames ranging from 26 to 126 frames.

The developed systems are implemented using Visual Basic language (ver 6.0) under Windows Me and Xp operating systems. The systems are executed using IBM personal computer (processor Pentium4 1.5 GigaHz), zero cash. Since A distributed system is needed to implement the fractal model, therefore LAN with switch hub and LAN-cards of speed 100 Mbs are used.

4.2 Fidelity Criteria

Generally, fidelity criteria can be divided into two classes:

<u>(1) Objective Fidelity Criteria:</u> this kind of criteria borrowed from digital signal processing and information theory, they provide equations that can be used to measure the amount of error in the reconstructed (decompressed) frame. The objective criteria, although widely used, are not necessarily correlated with the perception of frame quality. However, they are useful as a relative measure in comparing between different versions of the same frame. Commonly used objective measures are the mean-square error (MSE), the peak signal-to noise ratio (PSNR). The error between the original (uncompressed) pixel value and the reconstructed (decompressed) pixel value can be defined as:

$$\text{error}(r,\ c) = I(r, c) - \hat{I}(r, c) \quad \text{.......... .... (4.1)}$$

Where I(r, c) is the pixel value of the original frame at the (r,c) location

$\hat{I}$ (r, c) is the pixel value of the decompressed frame at the same location (r,c),

Next, the total absolute error can be defined in an (H×W) decompressed frame as:

$$\textit{Total Error} = \sum_{r=0}^{H-1}\sum_{c=0}^{W-1}\left|I(r,c) - \hat{I}(r,c)\right| \quad \text{..................(4.2)}$$

The mean-square error is found by taking the error squared divided by the total number of pixels in the frame.

$$MSE = \frac{1}{W \times H}\sum_{r=0}^{H-1}\sum_{c=0}^{W-1}(I(r,c) - \hat{I}(r,c))^2 \quad \text{.....................(4.3)}$$

From quality point of view, the smaller value of the MSE means better compressed frame (compared with the original one). Alternatively, with the Peak-signal-to-noise (PSNR), larger number implies better frame. The PSNR considers the decompressed frame to be the "signal" and the MSE as the "noise". The Peak-signal-to-noise ratio in dbcan be defined as:

$$PSNR = 10\log_{10}\frac{(L-1)^2}{\frac{1}{W\times H}\sum_{r=0}^{H-1}\sum_{c=0}^{W-1}[\hat{I}(r,c)-I(r,c)]^2} \quad ....(4.4)$$

Where L is the number of gray levels (e.g., for 8 bits per pixel L=256)

(2) Subjective Fidelity Criteria[Scott98]: this kind of criteria requires the definition of a qualitative scale to assess frame quality. This scale can then be used by human test subjects to determine frame fidelity.

Two main types of subjective measurements exist. The first is referred to as impairment tests, where the viewers score the frames in terms of how bad they are.

The second is referred to as quality tests, where the viewers score the frames in terms of how good they are.

4.3 Performance Parameters

Although many key parameters were utilized in the literature for performance evaluation of various compression methods, in this research two key parameters are utilized (***compression ratio*** and ***compression time***).

4.3.1 Compression Ratio (C_r)

It is the degree of video file size (data) reduction due to compression process. This ratio represents the size of the original uncompressed video file to the size of the overall compressed data file (using equation 4.5)**[Scott98].**

$$C_r = \frac{Uncompressed\ file\ size}{Compressed\ \ file\ size} \quad ....(4.5)$$

The (C_r) parameter is an indicator for the compactness ability of the compression process.

4.3.2 Compression Time (C_t).

It is the overall time required to perform the compression process for all the blocks of the uncompressed frames. For the suggested method, the overall required time includes the time required to obtain parameters of the blocks in addition to searching time.

The minimization of searching time is considered as the most cost criteria, which will indicate the efficiency of the matching mechanism, and compression technique.

4.4 Test Samples

To evaluate the performance of the suggested compression methods, five different video sequences are taken. The first frame of each video sequence is shown in figure (4.1). These five video samples have different number of video frames ranging from 26 to 126 video frames, at which each frame is of size 352×288. Sample 1 has 26 frames, sample 2 has 51 frames, sample 3 has 76 frames, sample 4 has 91 frames, and sample 5 has 126 frames.

First frame of video sample 1

First frame of video sample 2

First frame of video sample 3

First frame of video sample 4

First frame of video sample 5

Figure (4.1) The Video Samples Frames

4.5 Testing Strategy

The testing operation is implemented on five video samples; each one contains two types of frames, ***Anchor (A) and Predictive (P) frames***. Testing parameters for the five testing samples are illustrated in six tables (from table 4.1 to table 4.6). The first three tables show the performance parameters of the first model (FDCT) at which each table is concerned with one of the three motion estimation methods (OTS, TSS, and HM), while the other three tables show the performance parameters of the second model (fractal) considering the three motion estimation methods (OTS, TSS, and HM) respectively.

4.6 Test Results

The testing tables of FDCT model (the first three tables from table 4.1 to table 4.3) consists of three main parts, the first is ***Anchor Frames (AF)*** measurement which illustrate the ***MSE***, ***PSNR***, ***compression time***, number of Anchor frames (NO. field), in addition to the ***compression ratio*** for the tested video sample. Each field of ***MSE***, ***PSNR*** and ***compression time*** are calculated for the total anchor frames of the tested sample (All field) in addition to the average for one frame (Avg. field).

The second is ***Estimated Frames (EF)*** , this part of the table have the same fields of the first part but for the estimated frames.

The third part is ***All Frames***, this part have the total number of frames, average PSNR for one frame, average MSE for one frame, compression ratio for the overall coded video file, in addition to the ***compression time*** for the total frames (Anchor and Estimated) and the average time for one frame.

The second three tables (from table 4.4 to table 4.6) concerning fractal model have the same specifications in addition to the field which represents the number of servers used for fractal coding since fractal model is implemented using distributed system.

From the testing results of the two models (shown in tables 4.1 to 4.6), one can notice the following:

1. The testing results of the ***Anchor frames*** for the first model using FDCT technique are very good results compared with the Anchor frame method used in fractal model. This idea is obvious through the results in the testing tables, where ***PSNR*** of the ***Anchor frames*** in these tables show the high quality of the decoded frames in the FDCT tables. Although the testing results (specially the ***PSNR***) of the ***Anchor frames*** for the second model (using Fractal compression technique) show the success of the fractal compression technique (i.e. its fidelity measures are acceptable).
2. The execution time of the Anchor frames using FDCT technique is less than the time required for the fractal technique (i.e., the compression system using FDCT is faster than compression system using Fractal technique).
3. The cost of the Fractal technique is very expensive from time point of view, where in the testing results of tables 4.4, 4.5, and 4.6 the fractal technique is implemented using seven computer servers, while the compression system using FDCT technique does not need these servers, only a single machine is enough to obtain results within acceptable time.
4. All the PSNR, and MSE fidelity measures of all the motion estimation techniques (OTS, TSS, and Hybrid methods) are acceptable results noticing that:
    - The quality measures of the OTS method (shown in table 4.1) are better than the quality measures of TSS method taking in consideration that the OTS method needs more execution time (i.e. OTS slower than TSS method).

- TSS method is the best one from execution time point of view, therefore it is suitable for real time systems. From quality measures point of view, it gives acceptable but less quality than the other methods.
- Hybrid method is the best one according to its quality measure and this is clear from the ***PSNR***, and ***MSE*** fidelity measures but on the other hand, it need more execution time.

5. The quality measures of all the motion estimation techniques that depends on a decoded Anchor frames using FDCT technique are improved (i.e. better) since the quality measures of the decoded frames using FDCT technique have better quality than the Anchor frames used with fractal technique.

4.8 Subjective Samples Testing

To evaluate the suggested compression methods (for the two models) subjectively, six study frames are chosen, four of them are used for Anchor frames testing, while other two frames are used to show the testing results for motion estimation methods. The following notes summarize the subjective test:

1. Figure (4.2) shows Anchor frames subjective testing; four Anchor frames are taken in this figure, the testing shows the successful results of these Anchor frames that tested subjectively in addition to their objective tests.
2. Figures 4.3, 4.4, 4.5 shows the success of subjective testing for two video frames that compressed using TSS, OTS, and HM motion estimation method, respectively. Since the difference between the original and the decompressed frames are imperceptible.

Anchor Frame 352×288
Time 1 sec
CR=11
PSNR=41.33
FDCT

Anchor Frame 352×288
Time 1.04 sec
CR=9.428
PSNR=41.06
FDCT

Anchor Frame
Time 50.9 sec
Using 7 Servers
CR=10.673
PSNR=28.741
Fractal

Anchor Frame
Time 51.1 sec
Using 7 Servers
CR=10.673
PSNR=28.8
Fractal

Figure (4.2) Subjective and Objective Anchor Frames Testing

Estimated Frame 4×4 blocks
Three Step Method
CR=42.671
PSNR=21.71
Time =0.898 sec

Estimated Frame 4×4 blocks
Three Step Method
CR=42.671
PSNR=21.99
Time =0.857 sec

Figure (4.3) Subjective and Objective Estimated Frames Testing of TSS method

Estimated Frame 4×4 blocks
OTS Method
CR=42.671
PSNR=22.8826
Time =1.088 sec

Estimated Frame 4×4 blocks
OTS Method
CR=42.671
PSNR=22.99
Time =1.078 sec

Figure (4.4) Subjective and Objective Estimated Frames Testing of OTS method

Estimated Frame 4×4 blocks
Hybrid Method
CR=42.671
PSNR=23.812
Time =1.297 sec

Estimated Frame 4×4 blocks
Hybrid Method
CR=42.671
PSNR=23.39
Time =1.126 sec

Figure (4.5) Subjective and Objective Estimated Frames Testing of HM method

Table (4.1): Test Samples Using First Model (FDCT) With OTS Method

Samples	Anchor Frames (AF)								Estimated Frames (EF)								All Frames					
	No.	Time for AF/Sec.		PSNR of AF		MSE of AF		Cr.	NO.	Time for EF/Sec.		PSNR of EF		MSE of EF		Cr.	No.	Time/Sec.		Average PSNR	Average MSE	CR.
		All	Avg.	All	Avg.	All	Avg.			All	Avg.	All	Avg.	All	Avg.			All	Avg.			
S1	3	3	1	123.9	41.3	14.7	4.9	10.8	23	23	1	520.2	22.6	8182.8	355.8	42.7	26	26	1	24.8	315.3	32.3
S2	5	4.5	0.9	206.2	41.2	24.7	4.9	10.5	46	50	1.1	1056.9	23	15075.8	327.7	42.7	51	54.4	1.1	24.8	296.1	34.3
S3	7	6.6	0.9	288.7	41.3	34.4	4.9	11	69	82	1.2	1599.4	23.1	21574.4	312.7	42.7	76	88.6	1.2	24.8	284.3	35
S4	10	11	1.1	412.9	41.3	48.7	4.9	10.9	91	113	1.2	2045.9	22.5	33408.2	367.1	42.7	101	124	1.2	24.3	331.3	34.8
S5	12	12	1	495.8	41.3	58.1	4.8	10.4	114	142	1.3	2595	22.8	39238.1	344.2	42.7	126	154	1.2	24.5	311.9	35.2

Table (4.2): Test Samples Using First Model (FDCT) With TSS Method

Samples	Anchor Frames (AF)								Estimated Frames (EF)								All Frames					
	No.	Time for AF/Sec.		PSNR of AF		MSE of AF		Cr.	NO.	Time for EF/Sec.		PSNR of EF		MSE of EF		Cr.	No.	Time/Sec.		Average PSNR	Average MSE	CR.
		All	Avg.	All	Avg.	All	Avg.			All	Avg.	All	Avg.	All	Avg.			All	Avg.			
S1	3	3	1	123.9	41.3	14.7	4.9	10.8	23	19.3	0.8	488.6	21.2	11237.9	488.6	42.7	26	22.3	0.9	23.6	432.8	32.3
S2	5	4.5	0.9	206.2	41.2	24.7	4.9	10.5	46	36.6	0.8	995.2	21.6	20533.9	446.4	42.7	51	41.1	0.8	23.6	403.1	34.3
S3	7	6.6	0.9	288.7	41.3	34.4	4.9	11	69	55.9	0.8	1478.9	21.4	32251.4	467.4	42.7	76	62.5	0.8	23.3	424.8	35
S4	10	11	1.1	412.9	41.3	48.7	4.9	10.9	91	75.1	0.8	1924.2	21.1	45459.4	499.6	42.7	101	86.1	0.9	23.1	450.6	34.8
S5	12	12	1	495.8	41.3	58	4.8	10.4	114	91.3	0.8	2478.9	21.7	49600.6	435.1	42.7	126	103.3	0.8	23.6	394.1	35.2

Table (4.3): Test Samples Using First Model (FDCT) With Hybrid Method

Samples	Anchor Frames (AF)								Estimated Frames (EF)								All Frames					
	No.	Time for AF/Sec.		PSNR of AF		MSE of AF		Cr.	NO.	Time for EF/Sec.		PSNR of EF		MSE of EF		Cr.	No.	Time/Sec.		Average PSNR	Average MSE	CR.
		All	Avg.	All	Avg.	All	Avg.			All	Avg.	All	Avg.	All	Avg.			All	Avg.			
S1	3	3	1	123.9	41.3	14.7	4.9	10.8	23	25.3	1.1	533.5	23.2	7163.4	311.5	42.7	26	28.3	1.1	25.3	276.1	32.3
S2	5	4.5	0.9	206.2	41.2	24.7	4.9	10.5	46	54.6	1.2	1068	23.2	14256.5	309.9	42.7	51	59	1.2	25	280	34.3
S3	7	6.6	0.9	288.7	41.3	34.4	4.9	11	69	86.8	1.3	1590.6	23.1	22221	322	42.7	76	93.4	1.2	24.8	292.8	35
S4	10	11	1.1	412.9	41.3	48.7	4.9	10.9	91	114.8	1.3	2102.9	23.1	28920.9	317.8	42.7	101	125.8	1.3	24.9	286.8	34.8
S5	12	12	1	495.8	41.3	58	4.8	10.4	114	145.4	1.3	2597.9	22.8	39007.6	342.2	42.7	126	157.4	1.3	24.6	310	35.2

Table (4.4): Test Samples Using Second Model (Fractal) With OTS Method

Samples	No.of Servers	Anchor Frames (AF)									Estimated Frames (EF)									All Frames					
		No.of Frames	Time for AF/Sec.		PSNR of AF		MSE of AF		Cr.		NO.	Time for EF/Sec.		PSNR of EF		MSE of EF		Cr.		No.	Time/Sec.		Average PSNR	Average MSE	CR.
			All	Avg.	All	Avg.	All	Avg.				All	Avg.	All	Avg.	All	Avg.				All	Avg.			
S1	7	3	114	38	81.5	27.2	375	125	10.7		23	25.3	1.1	503.4	21.9	9683	421	42.7		26	139.3	5.4	22.5	386.8	34.3
S2	7	5	185	37	134.8	27	655	131	10.7		46	50.8	1.1	991.3	21.6	20930	455	42.7		51	235.8	4.6	22.1	423.2	34.8
S3	7	7	262.5	37.5	191.6	27.4	833	119	10.7		69	82.1	1.2	1483.1	21.5	31809	461	42.7		76	344.6	4.5	22	429.5	35.3
S4	7	10	383	38.3	275.2	27.5	1150	115	10.7		91	113.8	1.3	1981.6	21.8	39312	432	42.7		101	496.8	4.9	22.3	400.6	35.9
S5	7	12	460.8	38.4	320.8	26.7	1656	138	10.7		114	143.1	1.3	2463.4	21.6	51186	449	42.7		126	603.7	4.8	22.1	419.4	36.1

Table (4.5): Test Samples Using Second Model (Fractal) With TSS Method

Samples	No.of Servers	Anchor Frames (AF)								Estimated Frames (EF)								All Frames					
		No.of Frames	Time for AF/Sec.		PSNR of AF		MSE of AF		Cr.	NO.	Time for EF/Sec.		PSNR of EF		MSE of EF		Cr.	No.	Time/Sec.		Average PSNR	Average MSE	CR.
			All	Avg.	All	Avg.	All	Avg.			All	Avg.	All	Avg.	All	Avg.			All	Avg.			
S1	7	3	114	38	81.5	27.2	375	125	10.7	23	20.9	0.9	477.8	20.8	12512	544	42.7	26	134.9	5.2	21.5	495.7	34.3
S2	7	5	185	37	134.8	27	655	131	10.7	46	45.8	1	958.2	20.8	24702	537	42.7	51	230.8	4.5	21.4	497.2	34.8
S3	7	7	262.5	37.5	191.6	27.4	833	119	10.7	69	62.2	1	1423.2	20.6	38847	563	42.7	76	324.7	4.3	21.3	522.1	35.3
S4	7	10	383	38.3	275.2	27.5	1150	115	10.7	91	82.4	1	1871.4	20.6	51961	571	42.7	101	465.4	4.6	21.2	525	35.9
S5	7	12	460.8	38.4	320.8	26.7	1656	138	10.7	114	106.1	1	2346.1	20.6	64866	569	42.7	126	566.9	4.5	21.2	528	36.1

Table (4.6): Test Samples Using Second Model (Fractal) With Hybrid Method

Samples	No.of Servers	Anchor Frames (AF)								Estimated Frames (EF)								All Frames					
		No.of Frames	Time for AF/Sec.		PSNR of AF		MSE of AF		Cr.	NO.	Time for EF/Sec.		PSNR of EF		MSE of EF		Cr.	No.	Time/Sec.		Average PSNR	Average MSE	CR.
			All	Avg.	All	Avg.	All	Avg.			All	Avg.	All	Avg.	All	Avg.			All	Avg.			
S1	7	3	114	38	81.5	27.2	375	125	10.7	23	27.6	1.2	515	22.4	8625	375	42.7	26	141.6	5.4	23	346.2	34.3
S2	7	5	185	37	134.8	27	655	131	10.7	46	54.8	1.2	1026	22.3	17526	381	42.7	51	239.8	4.7	22.9	356.5	34.8
S3	7	7	262.5	37.5	191.6	27.4	833	119	10.7	69	86.9	1.3	1553.9	22.5	25116	364	42.7	76	349.4	4.6	23	341.4	35.3
S4	7	10	383	38.3	275.2	27.5	1150	115	10.7	91	115.7	1.3	2057	22.6	32487	357	42.7	101	498.7	4.9	23.1	333	35.9
S5	7	12	460.8	38.4	320.8	26.7	1656	138	10.7	114	147.7	1.3	2574.1	22.6	40926	359	42.7	126	608.5	4.8	23	338	36.1

Chapter Five
Conclusions and Future Work

5.1 Introduction

This chapter is dedicated to present some derived conclusion and a list of proposals for future work related to the research work discussed in the thesis.

5.2 Conclusions

From the results presented in the previous chapters, some remarks related to the behavior and performance of the two suggested models AFC and MEC are stimulated. A summary of some important conclusions could be presented as follows: -

1. The compression system using FDCT of the first Model is better quality than the compression system using Fractal coding, and this idea is shown from the results of SNR and PSNR that mentioned in chapter four.
2. The compression system using FDCT is faster than the compression system using Fractal coding.
3. The new developed version of FDCT is faster than the classical version of DCT, and has a very good quality (see appendix A FDCT derivation).
4. The Motion Estimation using Block Matching techniques is a suitable approach for video coding, and all the ME encoders have the same ME decoder.
5. The OTS method of ME is better quality than TSS method of ME, but the OTS method is slower than TSS method.
6. The new proposed HM of ME has very good results of Frame quality.

7. Fractal coding using distributed systems is faster than Fractal coding on a single machine, and make it more suitable for image compression.
8. Increasing Fractal threshold will decreasing Frame quality, while decreasing Fractal threshold will increase Frame quality.
9. The developed distributed system topology is a good topology for Fractal coding that divide the Fractal search on the total numbers of the shared servers.

5.3 Future Work

1. Redesign and implement the first Model of video coding but using Wavelate transform alternative to FDCT.
2. Develop the video coding systems through adding interpolated frames that will increase quality of estimated frames.
3. Develop the second Model of video coding (using Fractal coding) and fasting the Fractal coding using dual processors computers.
4. Develop a new distributed system for video coding speeding up AFC and MEF together.

List of abbreviations

Abbreviation	Meaning
AD	Absolute Difference
AF	Anchor Frame
AFC	Anchor Frame Compression
AVI	Audio/Video Interleaved
BBM	Boundary Block Matching
BM	Block Matching
BMA	Block based Matching Algorithm
C	Coordinate
CD-ROM	Compact Disk-Read Only Memory
C_q	Quantized Coordinate
C_r	Reconstructed Coordinate
CT	Color Transform
DCT	Discrete Cosine Transform
FC	Fractal Computation
FDCT	Fast Discrete Cosine Transform
FIC	Fractal Image Compression
FSA	Full Search Algorithm
HDTV	High Definition Television
HM	Hybrid Method
IFDCT	Inverse Fast Discrete Cosine Transform
IFS	Iterated Function System
IP	Internet Protocol
IT	Information Technology
JPEG	Joint Photographic Expert Group
LAN	Local Area Network
MAD	Minimum Absolute Difference
MAE	Mean Absolute Error
MB	Macro Blocks
MC	Motion Compensation

ME	Motion Estimation
MEF	Motion Estimated Frame
MPEG	Moving Picture Expert Group
MSE	Mean Squared Error
MVF	Motion Vector Field
O	Offset
O_q	Quantized Offset
O_r	Reconstructed Offset
OTS	Once Time Search
PIFS	Partitioned Iterated Function System
PSNR	Peak Signal to Noise Ratio
RGB	Red Green Blue
RIFF	Resource Interchange File Format
RLE	Run Length Encoding
RMS	Root Mean Square
S	Scale
SAD	Sum Absolute Difference
SNR	Signal to Noise Ratio
S_q	Quantized Scale
S_r	Reconstructed Scale
Sym	Symmetry
TSA	Three Step Algorithm
TSS	Three Step Search
VSBM	Variable Size Block Matching
WAN	Wide Area Network
WWW	World Wide Web

- **[Add00]** Addel J., Lossy and Lossless Image Compression, 2000, Prentice Hall PTR
- **[Ako00]** A. Kokaram, A New Global Motion Estimation Algorithm and Its Application to Retrieval in Sports Events, Msc. Thesis, 2000, University of Dublin2 (Ireland)
- **[Ali97]** Alice Yu, Motion Search Performance using the H.263 Encoder, 1997, Prentice Hall PTR
- **[Ama02]** Amal Abbas Kadhim, H.263 Image Video Compression, 2002, Msc. Thesis, University of Technology
- **[Arr97]** Array Microsystem, An Introduction to Video Compression, 1997, Prentice Hall PTR
- **[Aud00]** Auday Ali H. Al-Dulaimy, Fractal Image Compression with Fasting Approaches, 2000, Msc. Thesis, Al-Nahrain University
- **[Dde00]** D. Demirdjian, Motion Estimation from Disparity Images, Msc. Thesis, 2000, University of Cambridge
- **[Fad04]** Fadhil Salman Abed Al-Qasi, Adaptive Fractal Image Compression, 2004, Phd. Thesis, University of Technology
- **[Fcc00]** F. C. Cesbron, Multiresolution Fractal Coding of Still Images,Ph.d Thesis, 2000, The European Platform of the Georgia Institute of Technology
- **[Gle01]** Glen G. Langdon, Lossless Image Compression, Paper, 2001, University of

California

- **[Hyc¹01]** H. Y. Chunge, Adaptive Search Center Non-Linear Three Step Search, Paper, 2001, University of Hong Kong

- **[Hyc²01]** H. Y. Chung, Fast Motion Estimation With Search Center Prediction, Paper, 2001, University of Hong Kong

- **[Ibr04]** Ibraheem Nadher Ibraheem, Image Compression using Wavelate Transform, 2004, Msc. Thesis, Baghdad University

- **[Ism02]** Ismail Avcibas, A Progressive Lossless/Near-Lossless Image Compression, 2002, IEEE Signal Processing Letters

- **[Jan03]** Jan Bormans, Multimedia Image Compression, 2003, IMEC Kapeldreef Inc.

- **[Jim99]** Jim Trulove, Multimedia Networking Handbook, 1999, CRC Press LLC

- **[Jud98]** Judith Jeffcoate, Multimedia in Practice, 1998, Prentice Hall of India

- **[Kar04]** Karin R. Gastreich, OTS/Duke Undergraduate Semester Abroad Program, 2004, book, University of Costa Rica

- **[Ken02]** Ken Cabeen, Image Compression and The Discrete Cosine Transform, 2002, College of the Redwoods

- **[Mar00]** Marcin Chady, Application of The Bulk Synchronous Parallel Model in Fractal Image Compression, Paper, 2000, University of Birmingham

• **[Mic98]**	Michael Orzessek, ATM & MPEG-2 Integrating Digital Video into Broadband Networks, 1998, Prentice Hall PTR
• **[Moh99]**	Mohamed Alkanhal, Correlation Based Search Algorithms for Motion Estimation, Msc. Thesis, Book, 1999, Carnegie Mellon University
• **[Nic03]**	Nicorsin F., Video Compression Technologies, 2003, NIC Inc.
• **[Pan01]**	Panrong Xiao, Image Compression By Wavelate Transform, 2001, Msc. Thesis, East Tennessee State University
• **[Raf00]**	Rafael C. Gonzalez, Digital Image Processing, 2000, Addison-Wesley
• **[Rao01]**	Raouf Hamzaoui, Fractal Image Compression, 2001,Leipzig University
• **[Sco98]**	Scott E. Umbaugh, Computer Vision and Image Processing, 1998, Prentice Hall PTR
• **[Sfe01]**	S. Feyrer, Parallel Image Processing, 2001, Msc. Thesis, Verlag Berlin
• **[Sim01]**	Simon K. Alexander, Two-and Three Dimensional Coding Schemes for Wavelet and Fractal-Wavelet Image Compression, 2001, University of Waterloo

• **[Som01]**	Somphob Soongsathitanon, A New Orthogonal Logarithmic Search Algorithm for Fixed Block-Based Motion Estimation for Video Coding, 2001, University of Newcastle Upon Tyne
• **[Tod00]**	Todd Lammle, Cisco Certificated Network Associate, Book, 2000, SYBEX Inc.
• **[Vir99]**	Virginie RUIZ, Motion Estimation Through Approximated Densities, 1999, University of Patras (Greece)
• **[Yil02]**	Yi Liang, Phase-Correlation Motion Estimation, 2002, Stanford University

Table of Contents

Republic of Iraq
Ministry of High Education
Al-Nahrain University
College of Science

Interframe Compression using Distributed Systems

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE/COLLEGE OF SCIENCE, Al-NAHRAIN UNIVERSITY IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR
THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE

By

Dhiah Eadan Jabor
dhiaheadan@yahoo.com
(B.Sc. 2002)

SUPERVISORS

Dr. Loay A. George **Dr. Venus W. Samawi**

Januray 2005 **Thi Al-Hija 1425**

الكلمات الإفتتاحية

Video Coding
Video Compression
Image Compression
Lossy Compression
Lossless Compression
Video Frames
Intra Frame Compression
Inter Frame Compression
Fractal Coding
Fractal Compression
Fractal Image Compression FIC
Discrete Cosine Transform
Image Transform
Motion Estimation
Motion Compensation
Three Step Search TSS
Once Time Search OTS
Fractal Quantization
DCT Quantization
Anchor Frame
Estimated Frame

الخلاصة

هناك بديلان رئيسان لضغط الفيديو, الأول عادةً يدعى الأنترافريم (intraframe approach) و يعمل على أزالة تكرارات السباشيال(spatial redundancy) الموجودة في الصورة و بدون التأثير على المعلومات الهامة. هذهِ الطرق مناسبة لتطبيقات الصورة الثابتة مثل الوسائط المتعددة (multimedia)، قواعد البيانات الصورية، الترميز الفوتغرافي، ألخ. في التطبيقات التي تستخدم الصور المتتابعة، البيانات التلفازية، ألخ. من الممكن أزالة تكرارات التيمبورال (Temporal redundancy) لتحقيق الزيادة في نسبة الضغط ولهذا السبب فأن الصور الفيديوية المتتابعة عادةً مترابطة بشكل كبير. المجموعة الثانية من هذهِ الطرق تدعى الأنتيرفريم (interframe approach) وهي تعمل على حذف تكرارات التيمبورال (temporal redundancy). في المجموعة الثانية يتم حساب تخمين الحركة (motion estimation) للصور الفيديوية المتسلسلة. في النظام المعتمد تم أعتماد الأنتيرفريم (interframe approach).

في حقل تخمين الحركة (motion estimation) لضغط الفيديو هناك تقنيات كثيرة مطبقة في هذا المجال. أن تخمين الحركة الكتلي (Block based motion estimation) من أكثر طرق العمل المطبقة في مختلف الخوارزميات. خوارزمية البحث الكامل (Full Search Algorithm) توفر أفضل أنجازية ولكنها ذات كلفة حسابية كبيرة. لتقليل متطلبات الحساب، هناك خوارزميات بحث سريعة قد تم تطويرها ومنها خوارزمية الخطوات الثلاث (Three Step Algorithm). في النظام المعتمد، طريقة البحث الزمني الآني (Once Time Search)، وطريقة بحث الخطوات الثلاث (Three Step Search) قد تم تنفيذها في مجال تخمين الحركة بالأضافة ألى الطريقة الهجينة المطورة الجديدة (Hybrid Method).

مجموعة طرق (interframe approach) تنتقي عدد من الصور الفيديوية التي يتم ضغطها بأستخدام نظام ضغط يكون مختلفاً عن تقنيات تخمين الحركة، هذهِ الصور الفيديوية المنتقاة تدعى صور المرساة الفيديوية (Anchor Frames). في النظام المعتمد هناك نموذجان قد تم تطويرهما لغرض ضغط الفيديو، في النموذج الأول تم تطوير نظام ضغط يعتمد على تحويل الجيب تمام المسرع (FDCT) الذي هو عبارة عن أشتقاق جديد لتحويل الجيب تمام التقليدي (DCT)، و قد تم توثيق هذا الأشتقاق بشكل كامل في العمل المعتمد، أما النموذج الثاني فقد تم من خلالهِ تطوير نظام ضغط الترميز الجزيئي (Fractal Coding) المستخدم لضغط صور المرساة الفيديوية. من السلبيات الموجودة في الترميز الجزيئي هو الوقت المكلف الذي يحتاجه لأكمال البحث المطلوب. هذهِ المشكلة قد تم معالجتها في النظام المعتمد من خلال تسريع البحث الجزيئي بأستخدام الأنظمة الموزعة (distributed systems) التي ستقسم البحث الجزيئي على عدد الحواسيب الكلي المشترك في الشبكة المستخدمة. لقد تم تنفيذ النظام المعتمد بأستخدام فيجوال بيسك ٦,٠ كلغة برمجية. تم أستخدام معدل الخطأ التربيعي (MSE)، و نسبة الأشارة ألى التشويش (PSNR) كمعاملات حساب دقة النتائج المستخلصة من التقنيات الكلية المطورة في النظام المعتمد.

الجمهورية العراقية
وزارة التعليم العالي و البحث العلمي
جامعة النهرين

الكبس الضمني للصور المتسلسلة بإستخدام الأنظمة الموزعة

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

من قبل
ضياء عيدان جبر
(بكالوريوس جامعة النهرين ٢٠٠٢)

أشراف

د. لؤي أدور جورج	د. فينوس وزير سماوي

كانون الثاني ٢٠٠٥	ذي الحجة ١٤٢٥