New Efficient Block-Based Motion Estimation Algorithms for Video Compression and Their Hardware Implementations

by

Mohamed Mohamed Rehan
B.Sc., Cairo University, 1991
M.Sc., Cairo University, 1994

A Dissertation Submitted in Partial Fulfillment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Electrical and Computer Engineering

New Efficient Block-Based Motion Estimation Algorithms for Video Compression and Their Hardware Implementations

by

Mohamed Mohamed Rehan
B.Sc., Cairo University, 1991
M.Sc., Cairo University,1994

Supervisory Committee

Dr. Pan Agathoklis, (Department of Electrical and Computer Engineering)

Supervisor

Dr. Andreas Antoniou, (Department of Electrical and Computer Engineering)

Co-Supervisor

Dr. Fayez Gebali, (Department of Electrical and Computer Engineering)

Departmental Member

Dr. Dale Olesky, (Department of Computer Science)

Outside Member

Supervisory Committee

Dr. Pan Agathoklis
Supervisor

Dr. Andreas Antoniou
Co-Supervisor

Dr. Fayez Gebali
Departmental Member

Dr. Dale Olesky
Outside Member

ABSTRACT

Video compression technology aims at compressing large amount of video data for efficient transmission and storage without significant loss of quality. Most video compression techniques rely on removing temporal data redundancy between frames using motion estimation and motion compensation techniques which are generally very computationally expensive.

The objective of the research done in this thesis is to develop new efficient motion estimation techniques that reduce the computational complexity of motion estimation.

The thesis presents a new prediction technique referred to as weighted sum block matching (*WSBM*) which dynamically reduces the computational complexity by limiting the search to a small subset of the search area. Simulation results have shown that adding *WSBM* to some well-known search algorithms reduces their computational complexity by 6-15% without affecting the visual quality of the reconstructed video frames.

The thesis also presents two new algorithms based on the simplex optimization method, the simplex based block matching algorithm (*SMPLX*) and the flexible triangle search (*FTS*). Both techniques use a triangle that moves inside the search area and checks only positions that lie at its vertices. As a result the computational complexity of the search is reduced since it depends directly on the number of positions checked. The techniques

can change the size and orientation of the search triangle during the search. The changes make the search highly flexible and efficient and reduce the number of search positions to be checked compared to those in other search algorithms.

The *SMPLX* uses equations based on the simplex optimization method to compute the new triangle size and orientation. The *FTS*, on the other hand, was implemented to be more suitable for a digital search grid by using look-up tables and integer computations. The two algorithms were implemented as part of the H.263 and H.264 encoders. Both algorithms were compared to the state of the art motion search algorithms. Experimental results showed that both algorithms can reach sub-optimal solutions while checking fewer search positions compared to other algorithms which results in lower computational complexity as a consequence.

Additional research was done to analyze and further improve *FTS* performance. As a result, various extensions of the *FTS* have been developed such as the enhanced *FTS* (*EFTS*), the half-pixel *FTS* (*HP-FTS*), and the predictive *FTS* (*PFTS*). These extensions were also implemented as part of the H.263 and H.264 encoders.

In the *EFTS*, repeated computations are reduced by caching intermediate results. In addition, the termination condition is modified to avoid premature exit. These modifications reduce the computational complexity of the *FTS* by up to 4%.

The *HP-FTS* extended the FTS so that the search can be done at half-pixel resolution instead of full-pixel resolution. The commonly used approach for half-pixel search is based on two separate stages, i.e., full-pixel search followed by half-pixel search. By combining the two stages in *HP-FTS*, the overall computational complexity can be reduced by an average of 15% without affecting the produced quality or compression ratio.

The *PFTS* uses prediction to select the direction of the starting search triangle. Analysis results show that the proper selection of the starting search triangle has great effect on the performance of the *FTS*. Simulation results show that the *PFTS* can reduce the computational complexity of the *FTS* by 7-13%.

Finally, hardware designs for the *FTS* and the full search (*FS*) algorithms are proposed.

The *FS* was chosen due to its regularity, low control overhead, and suitability for hardware implementation. It uses a high degree of parallelism and pipelining in order to improve the computational efficiency. The *FTS* requires less computation and thus provides high processing rates. Both designs were implemented, simulated, and verified using VHDL and then synthesized with Xilinx FPGAs. Simulation results have shown that both hardware implementations are more efficient than other existing implementations in terms of performance and hardware usage.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ASM	Algorithmic state machine
BAM	Block matching algorithm
CCF	Cross correlation function
CIF	Common Intermediate Format (352×288 pixels)
CLB	Combinational logic block
DCT	discrete cosine transform
DS	Diamond search
DVD	digital versatile desk
EFTS	Enhanced FTS
FPGA	Field programmable gate array
FS	Full search
FSM	Finite state machine
FTS	Flexible triangle search
GOP	Group of pictures
GOB	group of blocks
HDL	Hardware Description Language
HDTV	high definition TV
HP-FTS	Half-pixel FTS
HS	Hexagon search
IOB	Input/output block
IUV	Luminance and chrominance color mode
ISO	International Standard Organization
ITU-T	International Telecommunication Union-Telecommunication Standardization Sector

JPEG	Joint Photography Expert Group
JVT	Joint video team
LUT	Look up table
MAD	Mean absolute difference
MC	motion compensation
ME	motion estimation
MPEG	Moving Picture Expert Group
MSE	Mean square error
PFTS	Predictive FTS
PPHPS	Parabolic prediction-based half-pixel search
PSNR	Peak signal to noise ratio
QICF	Quarter CIF (176×144 pixels)
RAM	Random access memory
RGB	Red-green-blue color model
ROM	Read-only memory
RTL	Register transfer level (or language)
SAD	Sum absolute difference
SDTV	standard definition TV
SIF	Source intermediate format (352×240 pixels)
SNR	Signal to noise ratio
SoC	System-on-a-Chip
SMPLX	Simplex-based block-matching algorithm
SQCIF	Sub QCIF
TDL	Two dimensional logarithmic search
TSS	Three step search
VHDL	Very high speed integrated circuits HDL
VLC	variable length coding
VLD	variable length decoding

WSBM	Weighted sum block matching
YUV	same as IUV

Acknowledgments

In the name of Allah, the Most Gracious, the Most Merciful. All the praises and thanks are to Allah, who gave me the strength, patience, and ability to complete this work. All prayers and peace be upon Mohammed, the prophet and messenger of Allah.

My great thanks to my parents who stood all the way behind me with their support, encouragement, prayers, and blessings until this work was done. I would like also to thank my wife and my sisters for their efforts and support.

I would like to express my appreciation to my supervisors, Dr. Agathoklis and Dr. Antoniou. Dr. Agathoklis spent a lot of his time during weekdays and weekends reviewing my work and guiding me with comments, feedback, and suggestions. Dr. Antoniou did an excellent job in reviewing my work and giving me very useful feedback about how to improve it.

Special thanks to Dr. Gebali and Dr. Watheq who provided me with very valuable advice and suggestions for the hardware section of this work. I would like also to thank Dr. Oleski for accepting to be on my supervisory committee and spending the time and effort in reviewing my thesis.

I would like to thank all my friends and colleagues who helped me in completing this work.

Dedication

To my parents: The source of life

To my family: The source of happiness

To my friends: The source of support

Chapter 1

Introduction

This chapter provides an introduction to the compression concept in general and video compression in particular. The chapter also describes the main thesis outlines.

1.1 Data Compression

Data compression has found widespread applications in recent years such as text compression, image compression, video compression, etc [1]. Compressed files are easier to store or transmit than uncompressed files since they require less storage space or transmission bandwidth. Decompression techniques are used to restore the original files when needed. The compression ratio is highly dependent on the information content, the compression technique used, and the desired reconstruction quality.

Data compression techniques are divided into two main types, lossless and lossy techniques. Each compression type targets specific applications. In lossless compression, the reconstructed data after compression is an exact copy of the original data. The main target applications of lossless compression are those applications where loss of data is not acceptable such as in text documents and medical imaging applications. Examples of lossless data compression techniques include Huffman, dynamic Huffman, arithmetic coding, Lempel-Ziv, and run-length coding. In most of these techniques, static or dynamic statistical analysis of data is performed in order to classify individual data elements and the frequency of their repetition. Then, unique symbols or sequences of bits are used to repre-

sent individual data elements. Data elements that are repeated frequently are represented by smaller symbols to achieve higher compression. The lossless compression ratio is usually in the range 1:2 - 1:10.

Lossy data compression techniques provide higher compression ratio at the expense of losing some insignificant data. Lossy compression techniques exploit certain features to increase the compression ratio without affecting the quality of the compressed data. Examples of these features include the human eye sensitivity to visual differences or human ear sensitivity to different audio frequencies. By using these features, higher compression ratio can be achieved with very limited quality degradation. Image compression algorithms can achieve compression ratios in the range 1:10 - 1:40. Several techniques are used for lossy image compression such as vector quantization, fractals, subband image coding, and transform coding. In addition, there are some very popular image compression standards such as JPEG and JPEG 2000 which rely on transform coding such as the discrete cosine transform (DCT) and wavelet transform.

1.2 Image and Video Compression Techniques

In image and video compression, lossy compression techniques are used to achieve higher compression ratio. Video and image compression are important areas of research due to the large amount of data required to represent visual information. Storage or transmission of visual information in uncompressed form is sometimes beyond the capacity of some storage devices or transmission channels. For example, for a typical true color (3 bytes per pixel) VGA resolution (640×480), the uncompressed image requires $921,600$ bytes or almost 1 Megabyte. The amount of storage required for one second of uncompressed video clip or 30 frames with VGA resolution would be $27,648,000$ or around 27 MBs. For a two-hour movie, this would take approximately 200 GBs of storage. This huge amount of video data cannot be easily stored, on a hand-held device, for example, or transmitted in real time over existing networks especially wireless networks. Typical applications that

require video and image compression are video or image storage, video conferencing, video streaming, HDTV, and facsimile transmission.

1.2.1 Image Compression techniques

Image compression standards such as JPEG [2, 3] and JPEG 2000 [4–6] rely on transform coding such as the DCT and wavelet. In the DCT transform coding, the image content is transformed from the spatial domain to the DCT or the frequency domain in which fewer symbols can be used to represent image information. Quantization can later be applied to improve the compression efficiency.

In vector quantization, the image is divided into blocks. An indexed block-set, codebook, of frequently repeated image blocks is defined and used as a base to represent the whole image. Compression is achieved by storing this codebook as well as a limited amount of information for each image block. This information includes the index number of the best matching block in the codebook for each image block.

In subband image coding, the image is divided into several non-overlapping frequency bands. Each frequency band is coded with a different target quality or compression ratio.

1.2.2 Video Compression Techniques

Video compression is considered a superset of image compression [7–11]. In video compression, there are two types of data redundancy that can be removed or compressed; spatial (intra-frame) and temporal (inter-frame). Spatial data represent data in the same video frame and can be compressed using image compression algorithms. Temporal data represent data between video frames and can be compressed using different techniques including motion estimation. By using both intra-frame and inter-frame compression, video compression ratio in the range 1:50 - 1:200 can be achieved.

Several general video compression approaches have been applied for inter-frame coding such as three-dimensional transform coding, hybrid coding, and frame replenishment [12].

Three-dimensional transform coding is an extension of the available two-dimensional coding techniques used for still images with the temporal direction added as a third dimension [13]. Hybrid transform coding uses a combination of two-dimensional image compression techniques for intra-frame compression in addition to other techniques such as motion estimation and motion compensation for inter-frame compression. Frame replenishment techniques are based on coding frame differences only. These techniques can be used in encoding motion sequences that involve slow motion such as in video telephony. Examples of existing video compression techniques are, three-dimensional wavelet transforms [13], fractal image/video compression [14, 15], vector quantization [16], neural networks [17], etc.

Motion estimation techniques have been used to improve the compression efficiency over frame replenishment techniques. Unlike frame replenishment techniques which encode the difference between blocks that lie in the same position in two successive frames, motion estimation techniques achieve better compression by searching and identifying the best matching blocks between frames. The resulting block differences and the motion vectors that describe the positions of the matching blocks are then encoded.

There are several types of motion estimation such as block-based, pixel-based, and region-based. Pixel-based motion estimation performs the search for each pixel in the frame, block-based motion estimation uses a block of pixels, and region-based motion estimation uses a pre-selected region or group of pixels. Block-based motion estimation is the most popular among motion estimation techniques due to its simplicity and suitability for hardware implementation.

In block-based motion estimation, each video frame is divided into a group of blocks referred to as macroblocks. Individual macroblocks of one frame are compressed by firstly locating the best matching macroblock in the previous frame, reference frame, and secondly, the difference between these two macroblocks as well as the relative displacement information, motion vector, are coded. This process is repeated for each macroblock in the frame to be coded.

Motion estimation is an essential part of any video compression standard due to the high compression ratio it can achieve. However, the computational complexity of motion-estimation algorithms can be 40 - 60% of the overall computational requirements for a video encoder. Consequently, there are several research directions associated with motion-estimation algorithms. The important ones are as follows:

- Using efficient techniques to check only a limited subset of the existing positions without affecting the search efficiency.
- Using prediction techniques to select a proper set of starting search parameters. These techniques rely on results from previous searches.
- Providing efficient hardware implementations for existing motion estimation techniques.

1.3 Video Compression Standards

The increased demand on video compression led to the emergence of several international standards and recommendations for video compression. Two main international organizations took part in the development of these standards and recommendations, namely, the International Standard Organization (ISO) which introduced Motion Picture Expert Group (MPEG) standards and the International Telecommunication Union-Telecommunication Standardization Sector (ITU-T) which introduced several video compression recommendations (e.g., the H.26x). Each committee has its own recommendations and standards. In addition, the two committees have produced joint work in the past and are currently working together on the H.264 standard.

Table 1.1 lists the most common standards proposed by the ISO and ITU-T committees. ISO has introduced MPEG-1 [18–20] for compressing/retrieving data to/from CD-ROM while MPEG-2 [21–24] was introduced for coding video signals for HDTV applications. Finally MPEG-4 [25, 26] is intended for a variety of video applications including applications for low bit-rate video transmission. On the other hand, ITU-T has introduced several

Table 1.1. *Video compression standards.*

Standard	Typical Bit-Rate	Organization	Typical applications
H.261	P x 64 kbits/s, p =130	ITU-T	ISDN Video phone
MPEG-1 (ISO 1172-2)	1.2 Mbits/s	ISO/IEC JTC1	CD-ROM
MPEG-2 (ISO 13818-2)	4-80 Mbits/s	ITU-TISO/IEC JTC1	SDTV, HDTV
H.263	64 kbit/s or below	ITU-T PSTN	Video Phone
MPEG-4 (IS 14496-2)	24-1024 kbits/s	ISO/IEC JTC1	A variety of video applications
H.264	64 kbits/s	ITU-TISO/IEC JTC1	A variety of video applications

video compression recommendations such as the H.261 [27], H.262 [28], H.263 [29, 30], and H.264 [31, 32]. Recommendations H.261, H.262, and H.263 were intended for video communication applications such as video conferencing and video telephony. H.262 is similar to the MPEG-2 standard.

H.264 is the most recent video standard and it targets a variety of video applications. It is meant to provide a substantial improvement above all other video compression standards and recommendations [33, 34]. The H.264 work was done jointly by ISO and ITU-T committees.

1.4 Thesis Outline

This thesis proposes efficient motion-estimation techniques that can be implemented in software and hardware multimedia applications. The thesis includes an analysis of the existing motion estimation algorithms, a discussion of their limitations, and a proposal for new approaches. Several algorithms are proposed and compared to other state-of-the-art algorithms.

Chapter 2 provides an overview of video compression standards. The chapter provides a brief introduction to the main building components of both video encoder and video

decoder systems, describes in detail the MPEG-1 standard, and provides highlights of other standards.

Chapter 3 explains the motion estimation problem in general and block-based motion estimation in particular. The main parameters of motion estimation are discussed. A detailed description of each parameter is provided. Finally, some well-known motion estimation algorithms are described in more detail. The chapter also includes a description of weighted sum block matching (*WSBM*) which uses prediction techniques to improve computational efficiency.

Chapter 4 describes the simplex optimization method [35–37] and the modifications introduced in order to use it for finding the best matching blocks between two frames. Results obtained with the proposed algorithm and comparisons between its performance and those of other algorithms are also provided.

Chapter 5 describes the *FTS* algorithm which was developed also based on the simplex optimization method but was modified to be more suitable for motion estimation problem. Comparison results between the *FTS* and other algorithms have shown that the *FTS* has superior performance in terms of the computational efficiency, accuracy of search results, and quality of the reconstructed video frames.

Chapter 6 introduces more improvements to the *FTS* algorithm based on an analysis of the *FTS* behavior. These improvements are the enhanced *FTS*, the half-pixel *FTS*, and the predictive *FTS*.

Chapter 7 presents two hardware implementations for the full search (*FS*) and the *FTS* techniques. Both implementations aim at using a high degree of parallelism and pipelining whenever possible as well as attempt to achieve high utilization of the hardware. The *FS* implementation uses a hierarchal decomposition approach and converts the 2-dimensional motion estimation problem into a series of 1-dimensional problems. The *FTS* implementation uses separate units for the regular and irregular parts of the *FTS*. In addition, pipelining is incorporated to improve hardware utilization. These implementations were developed, simulated and verified using the VHDL language and were synthesized in terms of Xilinx

FPGAs. Simulation results show that both implementations offer very good performance in terms of the number of cycles required to complete a motion estimation search per macroblock and the number of hardware units required to represent these implementations on FPGA.

Chapter 8 provides conclusions and suggestions for future research and development.

Chapter 2

Generic Video Compression Standard

This chapter provides an introduction to current video compression standards and recommendations. These standards have been developed by different committees and at different times to target different applications. However, the basic video compression concepts of these standards are similar. For the sake of simplicity, this chapter provides a detailed description of MPEG-1. Its features and comparison with other standards will be briefly discussed.

The MPEG standard consists of three layers; system, audio, and video [18–26] as shown in Figure 2.1. The video and audio layers specify the coded representation of video and audio compressed streams. The system layer specifies the layer that wraps the compressed audio and video together. The system layer defines how multiple channels of audio and video streams can be multiplexed, and other important synchronization and transmission information.

Sometimes, the video and audio streams are referred to as elementary streams. Each elementary stream has its own encoder and decoder pair. As an example, the MPEG video encoder is used to generate the compressed video bitstream according to the video layer specifications defined in the standard. On the other hand, the MPEG video decoder is used to convert the compressed video bitstream into uncompressed video.

The standard defines the bitstream syntax of the compressed video or audio. Technical implementation details were left open for research and competition. Different but compatible encoder implementations exist in commercial applications which have differ-

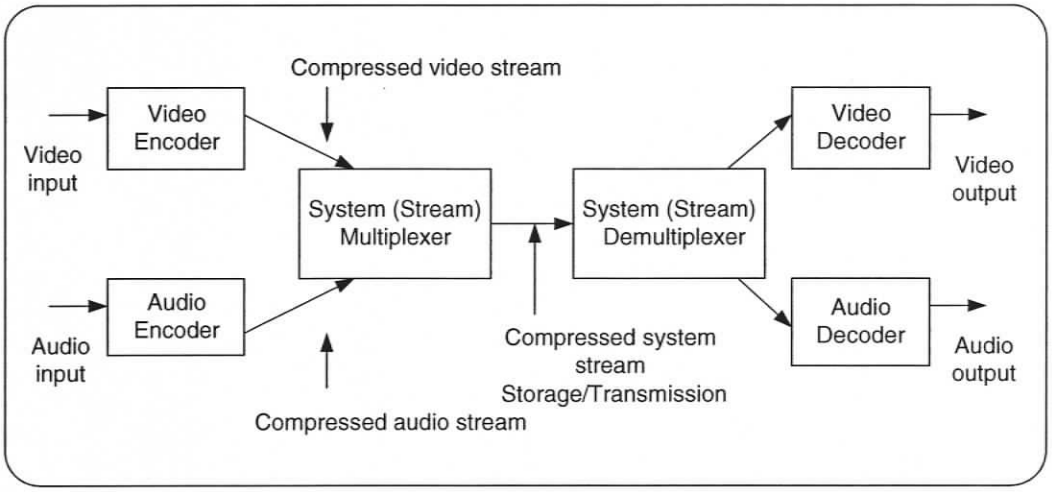


Figure 2.1. *Video compression layers.*

ent computational performance and support different features. As video compression is the main focus of this research, this chapter will discuss only the video compression layer.

2.1 Structure of MPEG Video Stream

The existing MPEG standards target a wide range of applications which include digital video storage on CD-ROM (MPEG-1), digital versatile disk (DVD) and HDTV applications (MPEG-2), and video conferencing and other applications (MPEG-4).

2.1.1 Input Video Format

The input video signal to a typical MPEG encoder is digitized and converted to the IUV or YC_bC_r color model instead of the RGB color model. In the YC_bC_r color model, each frame is represented by a luminance component (Y) containing brightness information and two chrominance components (C_b,C_r) containing color information. The YC_bC_r color model has low cross-correlation between its spectral components and has less spectral re-

dundancy. The choice of the YC_bC_r color model was based on the fact that the human eye is more sensitive to the Y component than to the chrominance components. Since most of the intensity information is present in the Y component, different coding precisions can be specified for each of the three components during the coding process and thus better compression can be achieved.

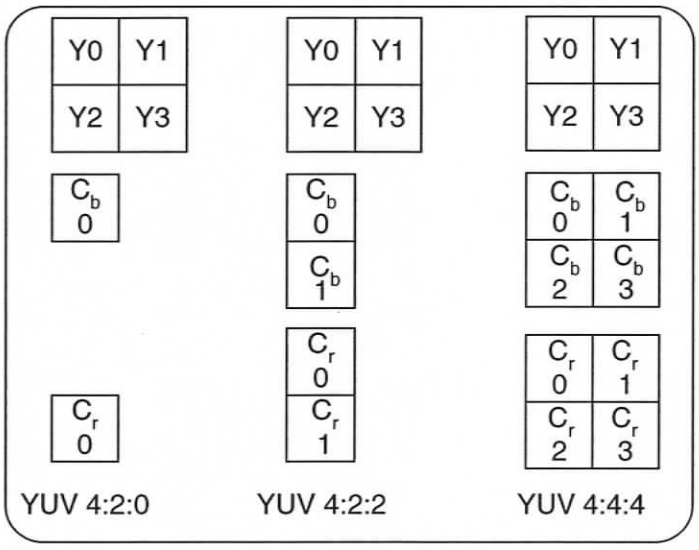


Figure 2.2. *YUV color subsampling.*

In most of the video compression standards, the two chrominance components are subsampled in either horizontal or vertical direction or both to improve compression efficiency. Figure 2.2 displays the different available sampling modes, namely, YUV 4:2:0, YUV 4:2:2, and YUV 4:4:4. In YUV 4:2:0, both C_b and C_r, are subsampled with respect to both the vertical and horizontal directions. In the YUV 4:2:2 mode, C_b and C_r are subsampled only in one direction. In the YUV 4:4:4 mode, no subsampling is done.

YUV 4:2:0 is the default subsampling mode is most standards. Other modes are supported in some profiles of MPEG-2, MPEG-4, and in H.264.

2.1.2 MPEG Video Frames

MPEG-1 has three basic video frame types: Intra-frames (I), predicted frames (P), and bidirectional predicted frames (B) as shown in Figure 2.3.

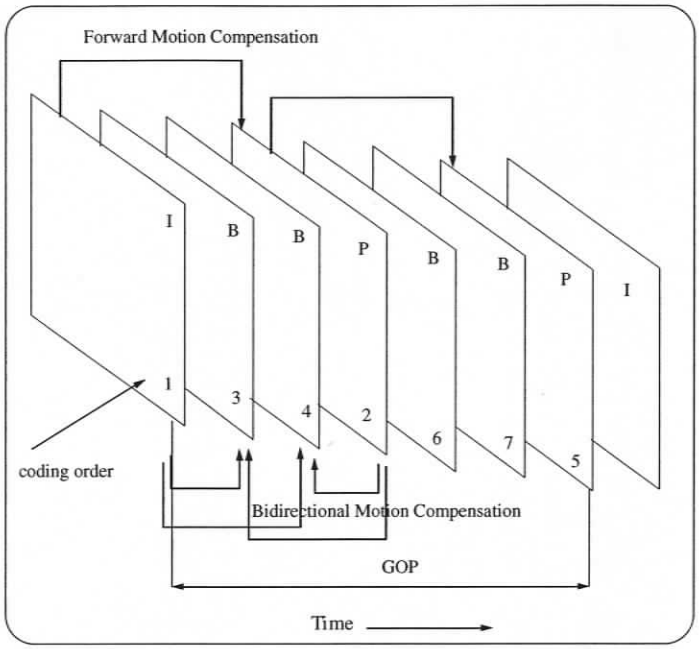


Figure 2.3. *Types of frames in MPEG.*

I frames are coded independently as still images using a similar approach to the JPEG standard [2]. P and B frames are inter-frame coded using motion estimation and motion compensation (ME/MC) techniques. In P frames, motion-estimation is done with respect to the immediate previously coded I or P frame. On the other hand, motion estimation for B frames is done with respect to the previous I or P frame as well as the next I or P frame. Since B frame coding requires pre-coding of frames in future time, the frame coding order in the compressed stream is different from the frame display order. Due to this fact, B frames coding requires frame buffering and reordering mechanisms at both the encoder and decoder.

2.1.3 MPEG Video Bitstream Layers

The MPEG video stream consists of six layers as follows:

1. Sequence layer
2. Group of pictures layer
3. Picture layer
4. Slice or group of blocks layer
5. Macroblock layer
6. Block layer

The sequence layer consists of a sequence header followed by sequence data and a sequence end symbol. The sequence header contains sequence specific information such as frame resolution, aspect ratio, picture rate, bit rate, supported optional modes, etc. The sequence data is simply a collection of compressed frames and the sequence end symbol is a symbol used to identify the end of the sequence data.

A sequence layer consists of one or more groups of pictures ($GOPs$) as shown in Figure 2.3. A GOP is a set of pictures in successive display order. A picture layer defines the coding information for each picture or video frame. The structure details of each picture is shown in Figure 2.4. As can be seen, each video frame is divided into slices. A slice itself is divided into macroblocks. Based on the color subsampling mode, each macroblock consists of 6, 8, or 12 blocks for the YUV 4:2:0, YUV 4:2:2, and YUV 4:4:4 modes respectively.

A slice can be as big as the whole picture or as small as a single macroblock. In some other standards, the term group of blocks ($GOBs$) is used instead of slice. Slices are very useful for error resilience and are recommended for applications where higher probabilities of data loss exist as in video transmission in noisy channels. In such an application, data received at the decoder side may be partially lost. The use of slices allows for a partial recovery and processing of the remaining received data.

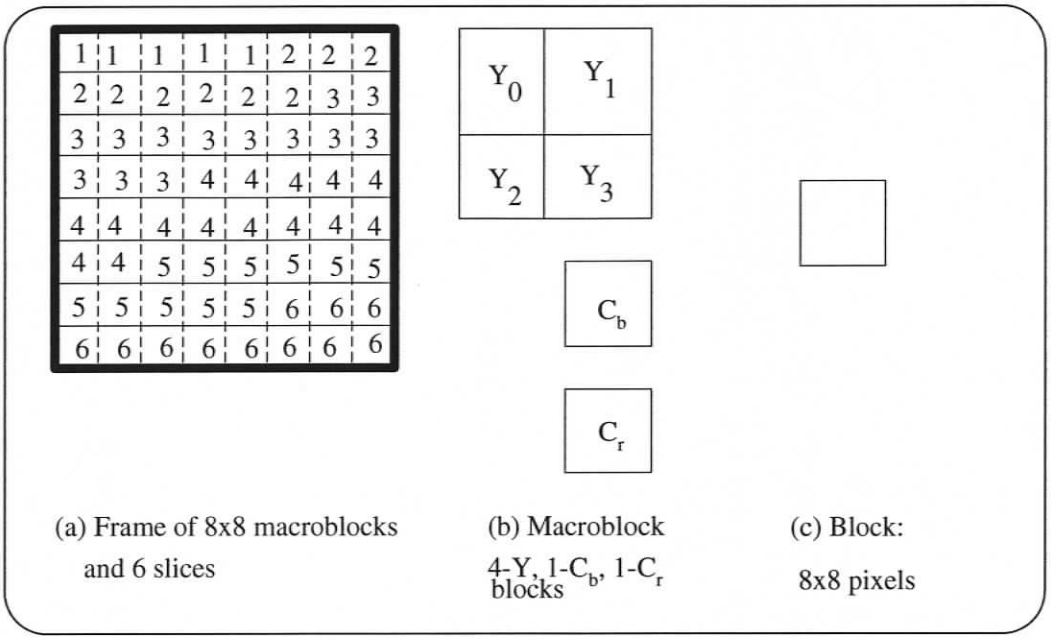


Figure 2.4. *Frame representation in MPEG.*

2.2 Basic Coding Process

This section describes the general techniques used in I, P, and B frame coding.

2.2.1 Coding of I Frames

Coding of I frames is based on the DCT algorithm. First, the DCT is applied to each 8×8 block as given by the equation

$$F(u,v) = \frac{1}{4}C(u)C(v)\sum_{m=0}^{7}\sum_{n=0}^{7}\cos\left[\frac{\pi(2m+1)u}{16}\right]\cos\left[\frac{\pi(2n+1)v}{16}\right], \quad (2.1)$$

where $u, v, m, n = 0, 1, \ldots, 7$, and

$$c(\omega) = \begin{cases} \frac{1}{\sqrt{2}} & \omega = 0 \\ 1 & O.W. \end{cases}$$

The resulting DCT coefficients are then uniformly quantized as follows.

The coefficient $F(0,0)$ is called the DC coefficient while all other coefficients are called AC coefficients. The DC and AC coefficients are quantized as

$$QF(0,0) = NINT\left[\frac{F(0,0)}{8}\right], \; u = v = 0, \tag{2.2}$$

where $NINT$ is the nearest integer value and

$$QF(u,v) = \frac{16F(u,v)}{MQUANT\, Q(u,v)}, \; u \neq 0, \; v \neq 0, \tag{2.3}$$

where Q(u, v) is a weighted quantization matrix and $MQUNAT$ is a quantizer scale factor. The quantization matrix sets the relative quantization step for each coefficient in the block. $MQUNAT$ is used to control the generated bit rate. Higher $MQUNAT$ means coarser quantization and a smaller number of output bits. $MQUNAT$ together with the quantization matrix determine the actual quantization factor. The quantization matrix can be altered for each sequence in MPEG-1 as well as each picture in MPEG-2. On the other hand, $MQUNAT$ can be changed for each macroblock.

The quantized coefficients are then zigzag scanned as shown in Figure 2.5 and ordered into a group of [run, level] pairs. The level value indicates the value of nonzero coefficient while the run value indicates the number of preceding zeros to that level. The [run, level] pairs are then variable-length coded (VLC) using the Huffman coding technique.

2.2.2 Coding of P and B Frames

P and B frames are inter-coded using ME/MC techniques. In ME/MC, the frame being compressed is called the current frame. The nearest I or P frame is called the reference frame.

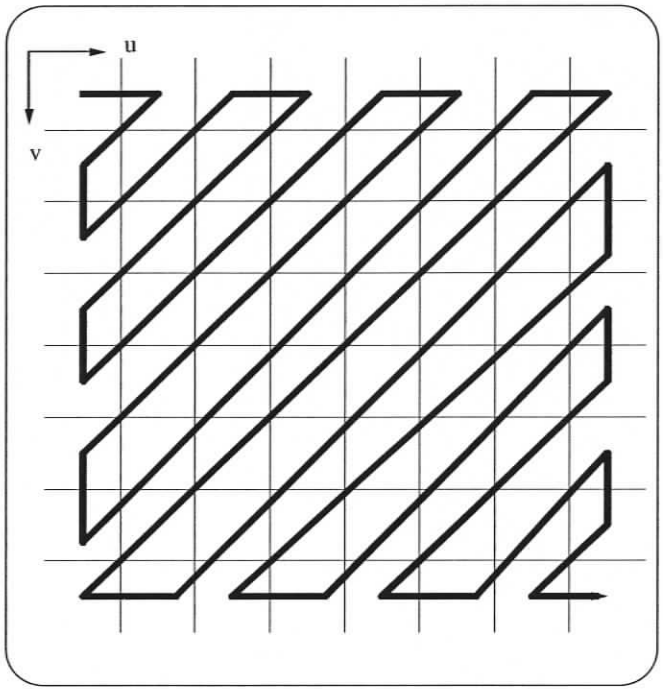


Figure 2.5. *Zigzag scan.*

Block-matching algorithms (BMAs) are used for motion estimation work at the macroblock level. BMAs are used to find the best matching macroblock between two frames i.e., the macroblock in the reference frame that has minimum difference from the macroblock being coded in the current frame. Once the best match between two macroblocks is found, the displacement vector between those macroblocks and the difference between them, which is referred to as the prediction error, are computed. The prediction error is then DCT processed, quantized, and VLC coded in a similar way to that for I frames.

Since the prediction error is usually smaller than the individual macroblock data, a much lower number of bits is needed for encoding and thus higher compression ratio is achieved. Due to the importance of finding the best matching macroblock, motion estimation is very computationally expensive and thus many block-matching algorithms with reduced computational complexity have been proposed. These algorithms aim at finding

the optimal or sub-optimal matching macroblocks by checking only a small subset of the potential best matching macroblocks. BMAs will be discussed in more detail in chapter 3.

2.2.3 Video Encoder and Decoder Block Diagram

The complete encoding and decoding processes are illustrated in Figures 2.6 and 2.7, respectively. At the encoder, frames are first reordered to follow prediction rules. Each frame is then divided into slices. Slices are divided into macroblocks which, in turn, are divided into blocks. Motion estimation is applied at the macroblock level while DCT and quantization are applied at the block level. The coefficients of each block are VLC coded into the output bitstream.

During encoding, the compressed stream bit rate is controlled by means of a rate control regulator which monitors the produced bit rate and compares it with the desired or target bit rate. The $MQUANT$ quantization factor is modified to increase or decrease the output bit rate within the target bit rate.

Since the decoder does not have access to the original frames at the encoder side, the encoder reconstructs the compressed frames to match the decoder output. Those reconstructed frames are used in the motion estimation process.

The decoding operation is exactly the reverse of the encoding operation. Block coefficients are variable-length decoded. Then inverse quantization followed by inverse DCT is applied to each block to reconstruct the prediction error. Motion compensation is then applied to reconstruct each macroblock. When all macroblocks in a frame are reconstructed, the frame is either displayed right away or stored to be displayed later according to its display order.

2.2.4 Encoder Performance Measure

The performance of any video encoder can be measured using different criteria such as the quality of the produced bitstream, the encoder computational complexity, and the resulting

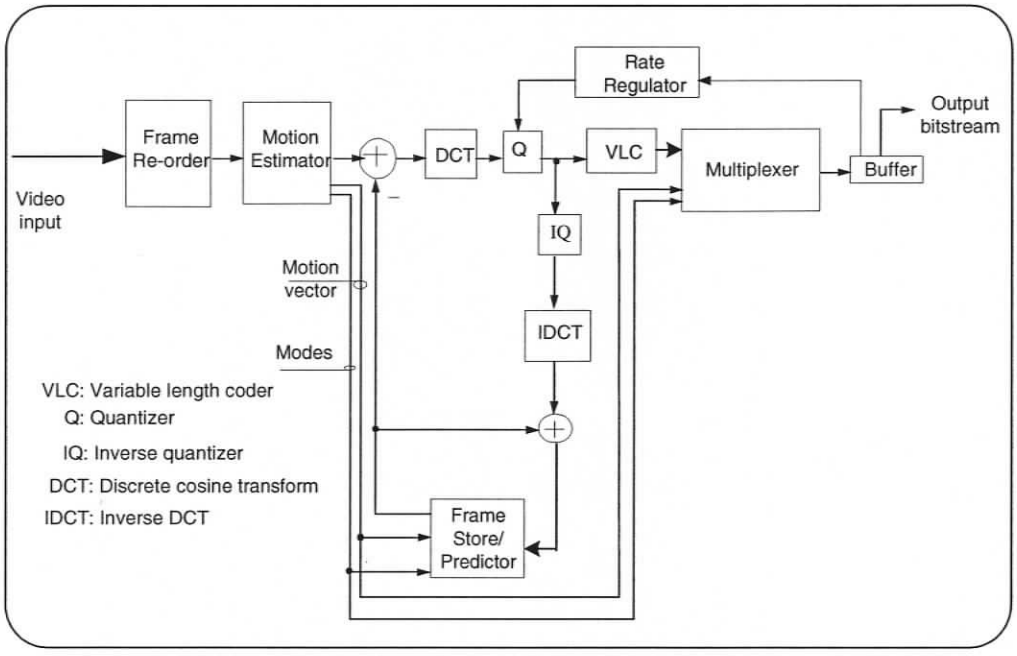


Figure 2.6. *MPEG video encoder.*

compression ratio.

The quality of the produced bitstream can be measured in terms of both quantitative and qualitative measures. The average peak-signal-to-noise ratio ($PSNR$) is often used to quantitatively measure the quality of the compressed bitstream and it is given by

$$PSNR = 10 \log \frac{255^2}{MSE} \tag{2.4}$$

where

$$MSE = \frac{1}{NM} \sum_{k=1}^{N} \sum_{l=1}^{M} [o_{i,j}(k,l) - r_{i,j}(k,l)]^2$$

In the above equation, $o_{i,j}$ and $r_{i,j}$ are the pixel value at location (i,j) in the original frame and the reconstructed frame, respectively. N, M are the numbers of frame pixels

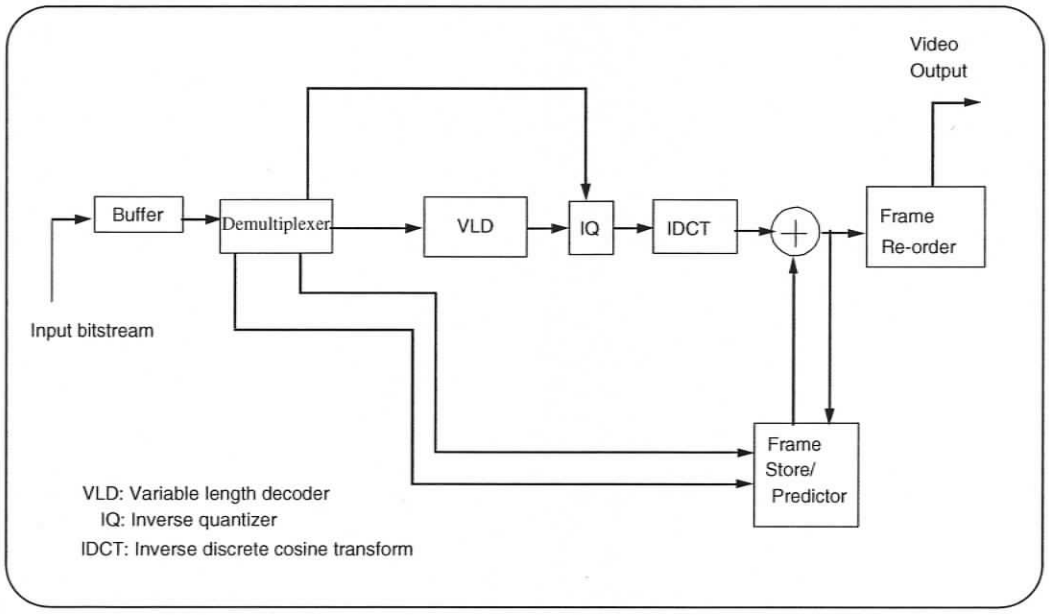


Figure 2.7. *MPEG video decoder.*

in the horizontal and vertical directions. In general, the higher the $PSNR$, the better the quality. The visual quality of the reconstructed frames is used as a qualitative or subjective measurement of the encoder performance. The compression ratio is highly dependent on the video sequence and it can range from 1:50 to 1:200.

The computational complexity of the encoding process is expressed in mega-instructions per second (MIPS). Motion estimation computational complexity is measured in terms of the number of search operations performed since the computation involved at each search position is the dominant factor as will be explained in the next chapter.

There are other comparison criteria that are used to compare the performance of BMAs. Examples of these criteria include the first-order entropy of the prediction error, the deviation of the obtained motion vectors from full-search results and the compression ratio. The first-order entropy of the prediction error is given by

$$V = -\sum_{i=0}^{N} p_i \log_2 p_i \tag{2.5}$$

where N is the number of all possible values of the prediction error.

2.3 Advanced Coding Process

The previous section described the basic video encoder behavior. In addition to the basic encoder, the standard allows optional features that can be specified by the target application. These features are used to improve the coding quality, efficiency, and robustness to bitstream errors. These features will be described in this section.

2.3.1 Sub-Pixel Accurate Motion Estimation

Research indicates that sub-pixel accuracy improves the $PSNR$ and reduces the noise level and blocking effect in the reconstructed bitstream. Examples of current sub-pixel motion search include half-pixel and quarter-pixel. In most standards, half-pixel motion estimation is used after full-pixel motion estimation is completed whereby the area around the best-matching position obtained from full-pixel search is interpolated and half-pixel search is applied to the surrounding pixels.

Half-pixel increases the computational complexity due to the pixel interpolation and additional search positions. However, a noticeable improvement in visual quality and compression ratio can be obtained. Due to the success of using half-pixel, quarter-pixel motion estimation was later added in MPEG4 and H.264.

2.3.2 Macroblock Coding Modes

Macroblocks can have different coding types or modes within a single I, P, or B picture. In I picture, macroblocks can be coded without ME. This type of macroblock coding is referred to as intra-macroblock mode. In a P picture, a macroblock can be coded

as inter-macroblock or intra-macorblock. Inter-macroblock is coded using ME/MC while intra-macorblock is coded without ME similar to I frame macroblock. The use of intra-macrblock coding mode in P picture is preferred when the prediction error is high and thus the intra-macroblock mode becomes more efficient than the inter-macroblock mode. B picture can include macroblock types similar to those of P picture as well as additional types due to the use of two reference frames in ME, forward and backward reference. These modes are forward, backward, and bidirectional where ME is performed using forward , backward, or both reference frames, respectively. Sometimes after quantization of a macroblock, all coefficients become zeros, so there is no need to code that macroblock and this type is called a skipped macroblock.

There are other modes where ME is performed at different macroblock resolutions such as 16×8, 8×16, 8×8, or 4×4. Each of these modes uses a different number of motion vectors and thus requires more bits to store motion vector information. These modes are not available in every standard. The choice of the proper coding mode is a trade-off among computational complexity, picture quality, and available coding bits [38, 39].

2.3.3 Rate Control Techniques

The quantization factor plays an important role in the quality and bit rate of the compressed video stream. The higher this factor, the higher the compression ratio and the lower the quality. The selection of this factor is not specified by the standard. As a result, there are several rate control techniques that have been introduced to optimally select the quantization factor such that the best quality is achieved at the specified target bit rate.

There are several rate control algorithms [40–42]. Most rate-control algorithms can be classified as frame-based or macroblock-based [43] techniques. In the frame-based rate control, the quantization factor is selected for the entire frame while in the macroblock-based rate control techniques, the quantization factor is selected for each macroblock. Frame-based rate control techniques are simpler but less accurate than macroblock-based ones.

2.4 Highlights of other Compression Standards

Other compression standards such as H.263, MPEG-4, and H.264 have the same basic features as MPEG-1 and MPEG-2. i.e., they include motion estimation, DCT transform coding, and VLC coding. However, each of these standard targets different applications and includes additional features that differentiate it from other standards. These features are generally added on top of the basic standard and some of these features can be turned on or off based on the application. Examples of these features are:

- Motion vector over picture boundary
- Deblocking filter
- Data partitioning and synchronization marks
- Reversible $VLCs$
- Slices
- Intra prediction
- Overlapped block motion estimation
- Multiple reference frames

In motion vector over picture boundary, boundary pixels are extended and the motion search can be done in the extended area. This technique improves the quality of the produced bitstream since some of the best matching macroblocks can lie on this area.

In block-based coding, blocking effects can occur between blocks due to the use of different quantization values for each block. The deblocking filter is a low-pass filter that is used to reduce these effects.

Slices are coded separately and can be recovered even if the rest of the picture is not recoverable.

Data partitioning is used to code higher priority bitstream elements separately from low priority elements. For example, the macroblock header, motion vector information, and DC coefficients of a macroblock are coded into a separate stream from the AC coefficients

for the same macroblock. The two streams can then be transmitted with different priority with higher priority given to the stream that contains the header information. This way, the high priority stream can be used to partially reconstruct the video frames if the low priority stream is lost.

Reversible $VLCs$ are used to recover coefficients of a corrupted block from either forward or backward direction.

The use of data partitioning, slices, and reversible $VLCs$ increases the size of the compressed bitstream due to the insertion of additional control bits.

Intra prediction is used to improve the compression ratio in I frames by coding the difference between the quantized coefficients in the same block instead of their absolute value. In general, this technique reduces the number of bits used for macroblock coding. The saved bits can be used to improve the quality of the compressed bitstream.

In multiple reference frames, motion estimation is performed with respect to one or more frames. The objective is to find the best matching across several frames and thus improve the quality and the compression ratio of the compressed bitstream.

Profile and Levels

In order to support a variety of applications, several optional features and specific coding constraints are defined in each standard in terms of profiles and levels. A profile defines an additional set of features over the basic implementation. These additional features enable the encoder to be more application specific. For example, these profiles can be added to improve the quality, error resilience, or compression ratio. On the other hand, levels define some of the restrictions on encoder behavior for certain profiles. As an example, maximum supported resolution, frequency, bit rate, etc.

The following subsections briefly present the main features of some other standards.

2.4.1 MPEG-2

MPEG-2 [22] was proposed to support interlaced video and HDTV or broadcasting applications. The main features of MPEG-2 are as follows:

- Support for interlaced video. As a result, motion estimation can be done on field or frame level. Macroblock size can be 16×8. An additional scan pattern for VLC codes that is more efficient for interlaced video was added.
- Support for different chrominance subsampling modes, i.e., YUV 4:2:0, YUV 4:2:2, and YUV 4:4:4.
- Support for scalable coding. Spatial, SNR, and temporal scalability. In spatial scalability, video sequence can be encoded and displayed at different resolutions. SNR scalability provides the encoded sequence at different quality levels. The client application can select the required quality level based on its own specifications such as available bandwidth, for example. Temporal scalability allows video sequence to be decoded and displayed at different frame rates.
- The optional features of MPEG-2 are specified in different profiles. Encoders need only to implement all the features defined in the same profile. This way, the complexity of the implementation is reduced compared to the case where all features have to be implemented in all encoders.

A detailed comparison between MPEG-1 and MPEG-2 can be found in [24].

2.4.2 MPEG-4

MPEG-4 has been introduced for the coding of audio-visual objects. One of the main features of MPEG-4 is the support of arbitrary shaped objects and the inclusion of different objects such as text, graphics, etc. Like MPEG-2, MPEG-4 has several profiles and levels. Other features of MPEG-4 are as follows:

- Improved coding efficiency by using DC prediction and AC prediction
- Additional scan mode (alternate horizontal scan)
- Includes new motion vector coding capabilities such as four motion vectors, unrestricted motion vectors, and motion vector over picture boundary

- Supports quarter-pixel motion estimation which enhances the motion search resolution and thus reduces the resulting prediction error
- Error resilience capabilities such as data partitioning, synchronization marks. reversible $VLCs$
- Object based video coding
- Still texture coding
- Mesh animation, face and body animation

2.4.3 H.261

The H.261 standard [27] is the first modern standard proposed for video conferencing on the ISDN network. It does not have many of the capabilities of MPEG-1 such as half-pixel, and B frames, for example.

2.4.4 H.263

The H.263 standard [29] was proposed as an enhancement of H.261. It includes several features that are similar to those of MPEG-1 and MPEG-4. The baseline H.263 does not have B frames or GOP header. Following the success of H.263, H.263+ and H.263++ recommendations were proposed as enhanced extensions to H.263 [30]. Each recommendation includes a set of additional annexes or features that can be added on top of the baseline H.263 to support more capabilities. Examples of H.263+ additional features include four motion vectors, slices mode for error resilience, intra-prediction, deblocking filter, and overlapped block motion compensation.

2.4.5 H.264

The H.264 standard [31] is the most recent proposed standard and it is considered the most efficient video coding standard so far. H.264 is intended to produce the best coding

efficiency over other standards. As a result, most of the optional features proposed in other standards are included in H.264. Due to its advanced features, the ITU-T and ISO committees have joined forces in the development of the H.264 standard. ISO has added H.264 as a separate profile in MPEG-4. H.264 defines three profiles, i.e., baseline, main, and extended. Some of the main features of H.264 [32] are as follows:

- Variable block-size motion compensation with small block sizes
- Quarter-pixel motion compensation
- Multiple reference picture motion compensation
- In-loop deblocking filter
- Context adaptive variable length coding
- Context-based adaptive binary arithmetic coding [44]
- Redundant pictures
- Data partitioning
- Weighted prediction

A full comparison between H.264 and other standards is found in [33, 34].

2.5 Standard Test Sequences

Standard test sequences are used to test the performance of the algorithms presented in the thesis. These are classified into three categories based on the level of motion involved in these sequences as follows:

- Low motion: Miss America, Akiyo, News, and Susie
- Medium motion: Flower garden, Tennis, Coastguard, and Silent
- High motion: Stefan, Foreman, Football, and Carphone

Table 2.1 lists the standard video formats and Figures 2.8, 2.9, and 2.10 display some of these standard test frames.

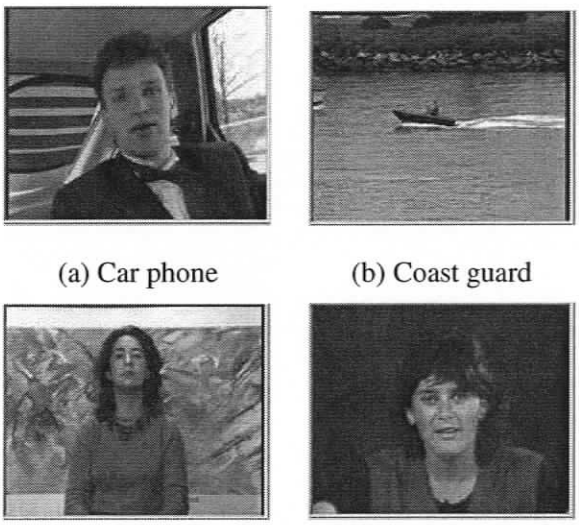
Table 2.1. *Standard video format.*

Format	TV System	Resolution	Sampling	frequency frame/sec
SQCIF (Sub QCIF)	NTSC/PAL	128 × 96	Prog.	30
QCIF (Quarter CIF)	NTSC/PAL	176 × 144	Prog.	30
SIF Source	NTSC	360 × 240	Prog.	30
Intermediate Format	PAL	360 × 288	Prog.	25
CIF	NTSC	352 × 288	Prog.	30
(Common Intermediate format)	PAL	352 × 288	Prog.	25
4CIF	NTSC/PAL	704 × 576	Prog.	30
ITU-R 601	NTSC	720 × 480	Inter.	30
	PAL	720 × 576	Inter.	25
SDTV	NTSC/PAL	720 × 480	Prog.	30
HDTV	NTSC/PAL	1920 × 1152	Prog.	50
		1920 × 1080	Prog.	60

2.6 Simulation Tools

Several video standard implementations were used for this research, namely, MPEG-2, H.263, and H.264 based on the availability of their public code. Since these standards have different settings and features, the results obtained by running the same search algorithm may differ from one standard to the other. However, the relative performance results obtained for the different search algorithms using the same standard were consistent. i.e., any algorithm that is faster in one standard implementation is also faster in the other implementations. Some of the noticeable differences among the three standards are as follows:

- MPEG-2 uses B frames which increase the number of search positions per macroblock in motion search.
- H.264 uses motion vector over the picture boundary which also results in a different value for the average number of search position.



(a) Car phone (b) Coast guard

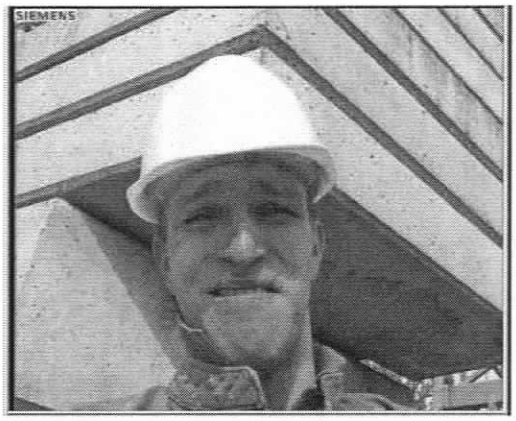
(c) Silent (d) Miss america

Figure 2.8. *Standard QCIF Sequences.*

- Each standard uses a different rate control algorithm and target bit rate.
- Each standard uses a different motion prediction technique.
- Each standard has a different VLCs for bit coding.
- Some standard use early termination techniques in motion search.

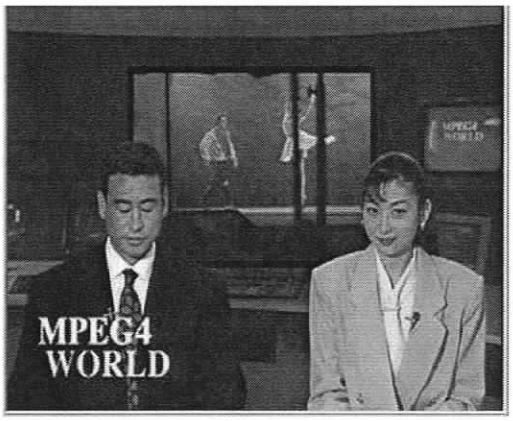
2.7 Conclusions

The MPEG-1 video coding standard has been described. The structure of the MPEG-1 bitstream syntax as well as the individual building blocks of both the video encoder and decoder have been explained. The main encoding parameters and performance measure criteria have been presented. Highlights of other video standards such as MPEG-2, MPEG-4, H.261, H.263, and H.264 have been covered briefly and the additional features that each standard includes have been described.



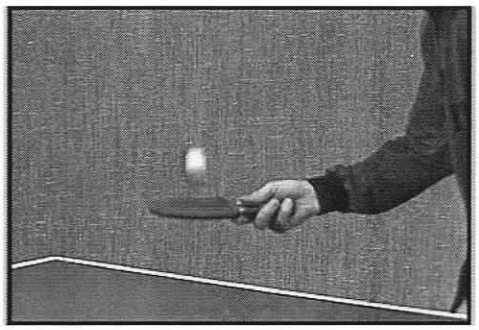
(a) Foreman

(b) Akiyo



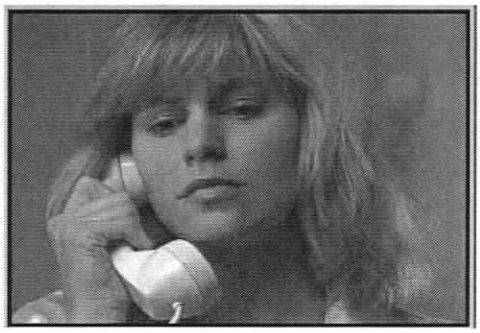
(c) News

Figure 2.9. *Standard CIF Sequences.*



(a) Tennis

(b) Flower garden



(c) Susie

Figure 2.10. *Standard SIF sequences.*

Chapter 3

Block-based Motion Estimation

The previous chapter provided an overview of video compression standards. In this chapter, motion estimation is described in more detail.

3.1 Introduction

Motion estimation is a key component in most of the video compression standards [18–31]. The main objective of motion estimation is to reduce the temporal data redundancy in a sequence of video frames. This is achieved by identifying the best matching blocks between two frames and encoding only the difference, referred to as the prediction error, between the two frames instead of encoding the entire frame. In general, efficient motion estimation produces a small prediction error and thus a coding of a block would require a small number of bits which results in a better compression efficiency as a consequence.

There are three types of motion estimation algorithms, pixel-based, block-based, and region-based. In pixel-based motion estimation, search for the best match is done on a pixel by pixel basis, in block-based motion estimation, search is done on a block of $n \times n$ pixels, while in region-based motion estimation, search is done on a region of an arbitrary size.

Block-based motion estimation algorithms are used in video compression standards due to their simplicity and ease of hardware implementation. In block-based motion estimation, each frame is divided into a group of equally sized macroblocks and each macroblock is handled as one entity during the motion estimation process. Usually, a block-matching

algorithm searches a pre-specified search area (search window) to find the best matching block in the reference frame to a specific block in the current frame to be coded.

Different evaluation criteria can be used to decide the optimal match. Some of the most commonly used criteria include the mean absolute difference (MAD), the mean-square error (MSE), and the cross-correlation function (CCF).

Figure 3.1 illustrates the motion estimation process. Generally at least two frames are involved; the reference frame and the current frame. The current frame is the frame that is currently been encoded. The reference frame is a frame that has been previously encoded and is used as a reference for the coding of the current frame. The current frame is divided into macroblocks and motion estimation is done on each macroblock individually. A motion estimation algorithm locates the best matching macroblock in the reference frame to the macroblock been encoded in the current frame. Once the best matching macroblock is found, the difference or the prediction error between the best-matching macroblock and the current macroblock is computed, DCT transformed, quantized, and run-length encoded. In addition to coding the difference between the two macroblocks, the motion vector which represents the relative displacement between the two macroblocks is also coded.

3.2 Block-Based Motion Estimation Parameters

The motion estimation problem can be formulated as follows: Given a block of $N \times M$ pixels in the current frame, it is required to locate another block in the reference frame that best matches the block in the current frame. Typical values for the block size in pixels are 16, 8, and 4. The current frame is defined as the frame at time t. The reference frame for forward motion prediction is defined as the frame at time $t - n$ and the reference frame for backward motion prediction is defined as the frame at time $t + n$.

As shown in Figure 3.2, the current block location is described by the x, y coordinates of its top-left corner or origin. The search area is the region of $[-P, P]$ pixels around the x, y origin in the reference frame. The parameter P is the search area width in one direction

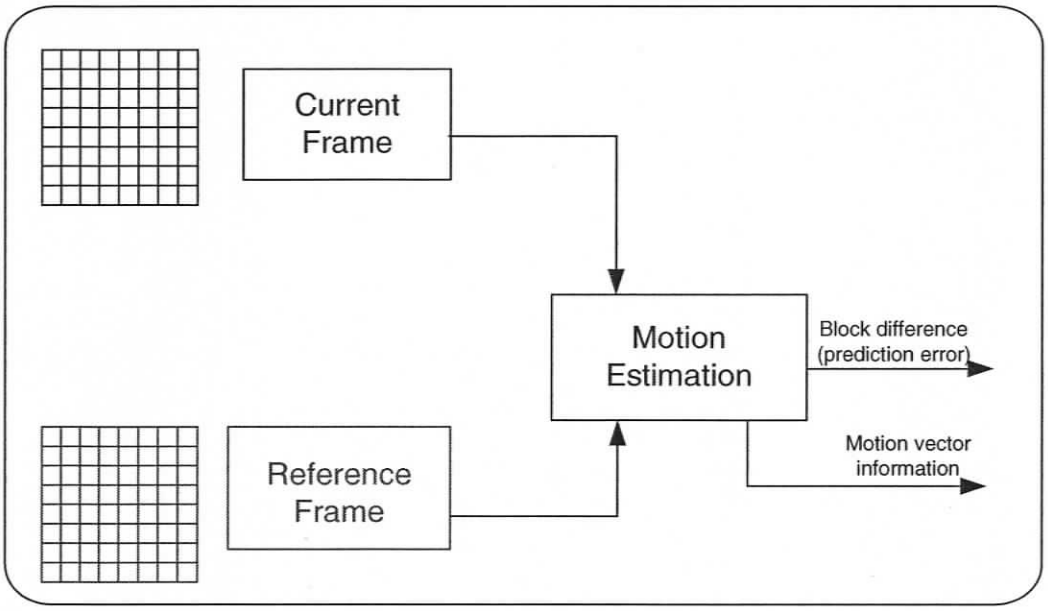


Figure 3.1. *Motion estimation process.*

and the corresponding size of the search area is $(2P + 1)^2$. Typical values of P are 8, 16, or 32. The value of P used can be supplied as an external parameter to the encoder.

Given a block in the current frame, it is required to find the best matching block whose origin is located inside the search area. This will lead to a block with a motion vector (u, v).

There are three main parameters associated with the motion estimation process: the search area, the evaluation criterion, and the search algorithm.

3.2.1 Search Area

The search for the best matching block includes all the blocks whose origins lie inside the search area as shown in Figure 3.2. If the search area lies over a frame boundary, it can be either trimmed or extended over the frame boundary. In a trimmed search area, the search is done on the parts on the search area that lie inside the frame boundary. On the extended search area, the boundary pixels are replicated to fit the search area outside

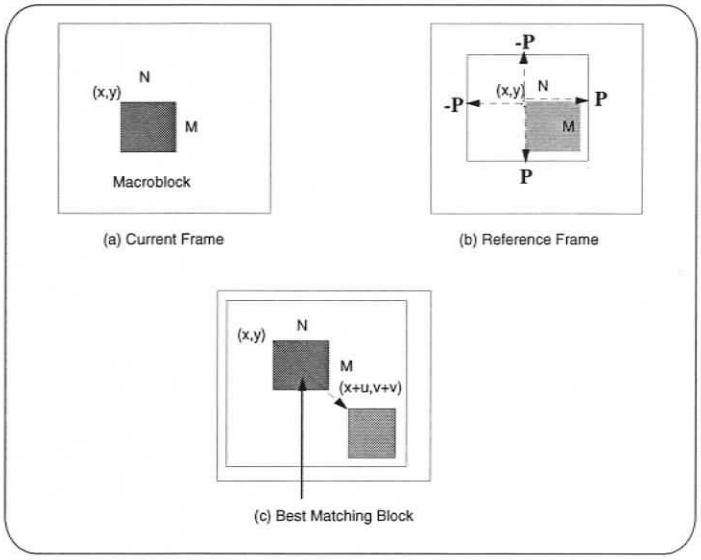


Figure 3.2. *Motion estimation parameters.*

the frame boundary. The later case is referred to as motion vector over picture boundary feature. This feature is not supported by all encoders/decoders.

3.2.2 Matching Criteria

A matching criterion is an evaluation function used to measure the degree of matching between two macroblocks. Based on the matching criterion, the macroblock that produces the best matching is chosen for encoding and the corresponding motion vector is used. As the matching criterion is used extensively in the search algorithm, it is essential to choose a computational inexpensive matching criterion. There are different matching criteria. The following subsections describe some of the popular ones.

3.2.2.1 Mean Absolute Difference

The mean absolute difference (MAD) is the most frequently used matching criterion for block matching. The MAD between the block in the current frame and the block in the

reference frame is given by

$$MAD(dx, dy) = \frac{1}{MN} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} |S_k(x, y) - S_{K-1}(x + dx, y + dy)| \quad (3.1)$$

where dx and dy are the displacements between the two blocks and $M \times N$ is the block size. MAD is preferred because it provides reasonable performance with reduced computational complexity as it does not involve multiplications or divisions. For each pixel, a subtraction, absolute value calculation, and an addition are required, i.e., 3 operations/pixel. If the division by $M \times N$ is removed, then the term sum of absolute difference (SAD) is used instead of MAD, that is

$$SAD(dx, dy) = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} |S_k(x, y) - S_{K-1}(x + dx, y + dy)| \quad (3.2)$$

3.2.2.2 Mean-Square Error

The mean-square error (MSE) which is the Euclidean distance between the two macroblocks is known to produce better results. This makes the MSE criterion more consistent with the human visual system. The MSE is given by

$$MSE(dx, dy) = \frac{1}{MN} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} [S_k(x, y) - S_{K-1}(x + dx, y + dy)]^2 \quad (3.3)$$

The main drawback of using the MSE is the additional computational complexity due to the extra multiplication step.

3.2.2.3 Cross Correlation Function

The cross correlation function (CCF) is derived from the correlation between two random variables and is defined as

$$CCF(dx,dy) = \frac{\sum_{x=0}^{N-1}\sum_{y=0}^{M-1} S_k(x,y)\, S_{K-1}(x+dx,y+dy)}{\sqrt{\sum_{x=0}^{N-1}\sum_{y=0}^{M-1} S_k^2(x,y)}\sqrt{\sum_{x=0}^{N-1}\sum_{y=0}^{M-1} S_{K-1}^2(x+dx,y+dy)}} \quad (3.4)$$

In contrast to using the minimum MAD or MSE as an indication for the best match, the maximum cross-correlation value indicates the best match. The CCF provides more accurate results than other matching criteria such as MAD and MSE. However, its computational complexity is much higher.

3.3 Block Matching Algorithms

Many of the available block matching algorithms have been proposed for the motion estimation problem [45–78]. This section provides a brief introduction to these algorithms. The most common algorithms, which are used in performance comparison in the next chapters, are described in more detail.

3.3.1 Full Search Algorithm

The basic search algorithm is the full or exhaustive search (*FS*) which checks each location in a specified search area to find the best matching block. The *FS* algorithm is very computationally expensive. For example, in the case where MAD is used as a matching criterion, the number of search locations is $(2P+1)^2$, each search requires 3 operations per pixel, and for $N \times M$ macrblocks, the total number of operations per macroblock required is $N \times M \times 3 \times (2P+1)^2$.

The *FS* algorithm is very regular and thus more suitable for hardware implementation than other algorithms. However, its computational complexity is very high and thus several other less computationally expensive algorithms have been proposed. These algorithms include fast full-search algorithms as well as fast search algorithms based on different approaches [47–50]. These algorithms achieve sub-optimal solutions with less computation through the selection of a small subset of the available search positions.

3.3.2 Fast Search Algorithms

This section describes some of the existing fast search algorithms. Examples include the cross-search algorithm [53], the 2-D logarithmic search [51,52], the three-step search (*TSS*) [54–56], the four step search [57], the hierarchical search [58], the diamond search (*DS*) [59–62], zonal search [62,63], the hexagon search (*HS*) [64], etc.

These algorithms achieve reduced computational complexity by

- using a simple search pattern such as the logarithmic search, three-step search, diamond search, hexagon search, etc.
- performing the search at different frame resolutions such as hierarchical search and multi-resolution search [65,66].
- using block features or partial data in evaluating the matching criterion such as pixel subsampling or integer project [67,68].

The following subsections explain in more detail the logarithmic, the diamond, and the hexagon search algorithms since they are among the most popular algorithms and, in addition, they have been used in our simulations.

3.3.2.1 Two-Dimensional Logarithmic Search

The two-dimensional logarithmic search (*TDL*) [52] is one of the early proposed block-matching algorithms. The technique is very simple and similar in nature to the binary search tree algorithm. The steps involved in the *TDL* search are as follows:

Step 1: The search area $[P, P]$ is divided equally into two areas, one area inside $[-P/2, P/2]$ and the other area outside it.

Step 2: Instead of performing full search in the $[-P/2, P/2]$ area, only nine search positions are evaluated. These nine points are the center point at $(0, 0)$ and the major eight points on the perimeter of the rectangular $(-d_i, d_i)$. The coordinates of the nine points are $(0, 0)$, $(0, d_i)$, $(0, -d_i)$,$(d_i, 0)$,$(d_i, -d_i)$, (d_i, d_i), $(-d_i, 0)$, $(-d_i, d_i)$, and $(-d_i, -d_i)$ where d_i is given by

$$d_i = 2^{k-1}, \tag{3.5}$$

$$k = \log_2 p$$

Step 3: The position with the best match is used as the origin of the next search iteration using distance $d_2 = d_i/2$ and the second step is repeated. This process is repeated k times and the minimum point obtained is the best matching point.

Based on the above description, the *TDL* search calculates $8k+1$ positions. The special case where $k = 3$ is illustrated in Figure 3.3. This search is referred to as the three step search (*TSS*).

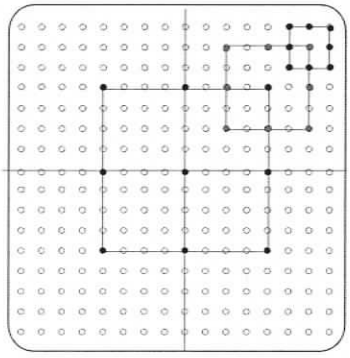


Figure 3.3. *Three step search.*

The *TSS* [53–56] provides good results for video conferencing scenes, which involve limited or slow motion. There are also some other algorithms that are related to the *TSS* algorithm such as the modified *TSS* ($MTSS$) [53], the new *TSS* (*NTSS*) [56], and the four step search (4SS) [57].

Experimental results have shown that the average number of search positions per macroblock is approximately 21 for the *TSS*, and 17 for the *NTSS*.

3.3.2.2 Diamond Search Algorithm

The diamond search algorithm (*DS*) [59–62] uses two diamond search patterns as shown in Figure 3.4; a large diamond shape for coarse search and a small diamond shape for fine search. The search starts using the large diamond and when the minimum position is found at the center of the diamond, the small diamond is used.

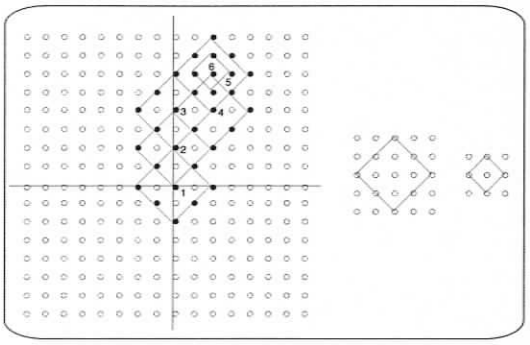


Figure 3.4. *Diamond search.*

The diamond search steps are as follows:

Step 1: Start the search with the large diamond. Select the diamond center as the origin of the search area. Calculate the error criterion at each point of the diamond as well as at the center. If the minimum is found at the center move to step 3. Otherwise, move to step 2.

Step 2: Use the minimum found at the previous step as the origin of the diamond and calculate the errors at the new diamond points. If the minimum is found at the diamond center, go to step 3. Otherwise, repeat step 2 with the diamond center as the minimum-error point.

Step 3: Switch to the small diamond with its center at the most recent minimum point. Calculate the error at each diamond point and exit.

Unlike the *TSS* which checks 9 positions at each iteration, the diamond search checks 9 positions in the first iteration and 4 positions in the last iteration. In intermediate iterations,

the *DS* checks either 3 or 5 positions. If the previous minimum is found at the diamond edge, 3 more positions are evaluated. If the minimum is found at a diamond vertex, 5 more additional positions are evaluated. The total number of checked positions is thus $(9+N) \times M + 4$ where N is the number of iterations and M can be either 3 or 5 based on the location of the minimum point in each iteration.

Several versions of the *DS* have been proposed in [59, 60, 62]. Experimental results have shown that the *DS* requires an average of 14 search per macroblock.

3.3.2.3 Hexagon Search Algorithm

The hexagon search algorithm (*HS*) is similar to the *DS* except that it uses a hexagon as search pattern instead of a diamond-shaped one. The choice of a hexagon pattern is based on research that indicates that the shape and direction of the search pattern can affect the search speed [64].

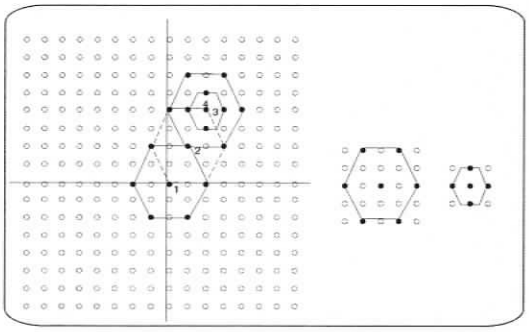


Figure 3.5. *Hexagon search.*

The steps used in the hexagon search are similar to those for the *DS* described in the previous section. The total number of search points per block is $(7+N) \times 3 + 4$ where N is number of iterations [64].

3.3.3 Advanced Search Techniques

In order to improve the speed and the quality of the motion search algorithm, various methods have been proposed and applied to some of the described search algorithms. These methods are general and can be applied to most search algorithms. An example of these methods is the thresholding technique where the most recent minimum is checked against a certain minimum value [69, 70]. If the current minimum is less than this minimum value, the search is stopped. The same approach can be used in the computation of the SAD, MAD, or any matching criterion. If during the computation of the matching criterion, the partial result exceeds the current minimum value, then the computation stops and the algorithm proceeds with the partial value obtained. This technique is referred as early termination and it is very useful in saving a lot of computation.

3.3.4 Motion Prediction Techniques

Motion prediction techniques [71–76] are used to further increase the search efficiency of block matching algorithms by identifying a good starting search position based on motion search results from the surrounding blocks in the same frame or in previous frames. The predicted starting position is used as the origin of a fast search algorithm to be executed later.

There are three predictions techniques: inter-frame [75], intra-frame [76], or both. In inter-frame prediction, a block location can be estimated using the information about the motion vector in the previous frame or frames. In intra-frame prediction, a block motion vector can be estimated based on the motion vectors of the surrounding blocks in the same frame. Inter-frame prediction requires more memory to store previous frame data and thus this technique involves additional implementation complexity.

One of the commonly used prediction techniques involves using the median of the surrounding blocks to predict the motion of the current block.

An approach which will be discussed in the next section, the weighted sum block

matching technique (*WSBM*), uses prediction to modify the origin of the search area and its size.

3.4 Weighted Sum Block Matching Technique

The aim of the *WSBM* is to benefit from motion vector information obtained from neighboring blocks since those blocks probably move in the same direction. If the search starts near the best matching location, the number of block matching evaluations can then be reduced and thus the computational complexity of the search can be reduced. The *WSBM* modifies also the size of the search window in addition to modifying the origin of the search window according to the calculated prediction information.

3.4.1 *WSBM* Steps

The *WSBM* technique is illustrated in Figure 3.6 and can be described in terms of following steps:

Step 1: The three motion vectors V_1, V_2, and V_3 from the previously calculated neighboring blocks are used to get MAD_1, MAD_2, and MAD_3 between the current block and the blocks whose origins are given by these motion vectors. Each origin consists of the current block origin shifted by one of these three motion vectors.

Step 2: A new motion vector V is calculated based on the weighted sum of the motion vectors V_1, V_2, and V_3. The weights of V_1, V_2, and V_3 are inversely proportional to MAD_1, MAD_2, and MAD_3, respectively. V is given by

$$V = \left[\frac{1}{2}\sum_{i=1}^{3}\left(1-\frac{MAD_i}{MAD_t}V_i\right)\right] \tag{3.6}$$

where

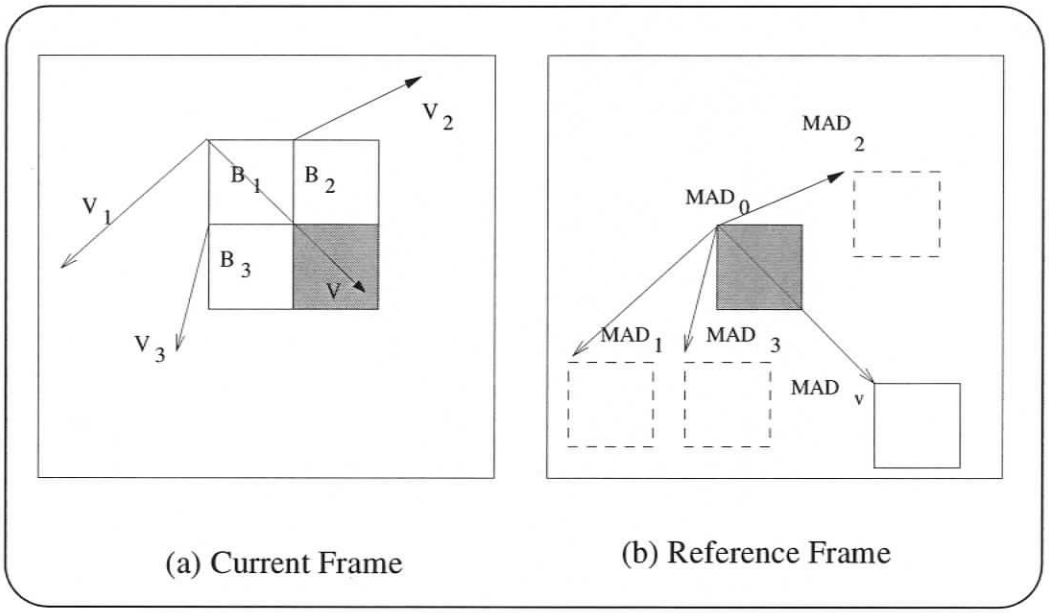


Figure 3.6. *WSBM prediction motion vectors.*

$$MAD_t = \sum_{i=1}^{3} MAD_i$$

The factor of $1/2$ is used for normalization purposes.

Step 3: MAD_V and MAD_0, the MAD value at $V_0 = 0$, are calculated. MAD_0 is checked to avoid the case where vector V leads to a worse matching than V_0 which can occur if slow motion is involved.

Step 4: MAD_V is compared with MAD_0. A value of MAD_V that is less than that of MAD_0 means that the V direction may lead to better matching. In this case, the new search origin is chosen by shifting the current block origin by vector V. Also, the search window size is reduced by half since it is more likely that the starting position is in the neighborhood of the best match. On the other hand, if MAD_0 is less than MAD_V, then the search origin remains V_0 and the search window size remains the same.

Step 5: A non-predictive search technique is applied to the modified search window origin

and size, which can be full search or any other search technique.

3.4.2 *WSBM* Results

The *WSBM* technique was implemented as part of the MPEG-2 encoder. Two sets of results were obtained. The first is a comparison between *WSBM* and the adaptive search window method (*ASWM*) [78]. The second comparison is with two different fast BMAs with and without the addition of *WSBM*. These algorithms are the *MTSS* search algorithm [74] and the orthogonal step search (*OSS*) algorithm [71].

In both simulations, three different standard sequences were used: Tennis, Flower, and Susie. The Tennis sequence contains fast movement in a small area of the scene, the Susie sequence presents a slow motion case, and the Flower sequence contains fast motion. all the other encoder parameters were fixed for all simulations. The sequence information and search window parameters were chosen as:

- GOP sequence IBBPBBPBBPBBPBBI
- SIF resolution 352×240 pixels/frame, 8 bits/pixel
- Initial search window range -16 to 16

The comparison criteria were chosen to be the number of block matching evaluations, compression ratio, and the average of the first-order entropy of the prediction error described in section 2.2.4. The number of block matching evaluations is important as an estimation of the computation complexity of the algorithm. The compression ratio and the first-order entropy give an estimate of the algorithm's success to locate the best match.

Table 3.1 shows the achieved simulation results. As can be seen, the required number of block matching evaluations has been reduced dramatically down to 30% when using *WSBM* with the *FS* algorithm although the compression ratio is the same. The average entropy of the errors is almost the same in all cases.

Table 3.2 shows the percentage improvement obtained by adding *WSBM* to the OSS and the $MTSS$ algorithms. As can be seen, the addition of *WSBM* has reduced the com-

Table 3.1. *Comparison between WSBM and ASWM.*

Sequence	Criterion	Full search	*WSBM*	*ASWM*
Tennis	Total BM evaluations (normalized)	1.00	0.30	0.62
	Average first-order entropy	4.1371	4.1595	4.1401
	Compression ratio	53:1	53:1	53:1
Flower	Total BM evaluations (normalized)	1.00	0.31	0.54
	Average first-order entropy	4.6809	4.7427	4.6717
	Compression ratio	22:1	22:1	22:1
Susie	Total BM evaluations (normalized)	1.00	0.31	0.58
	Average first-order entropy	2.4054	2.4054	2.4051
	Compression ratio	54:1	54:1	54:1

putational complexity of both algorithms. In both simulations, the visual quality of the produced sequences was almost identical.

3.5 Conclusions

Block-based motion estimation was discussed. Its formulation, parameters, existing solutions, etc., were explained. Different matching criteria as well as some of the most common block matching algorithms were presented. The fast algorithms used for performance comparisons such as the *TSS*, *DS*, and *HS* were explained in more detail. The chapter also dealt with the *WSBM* algorithm [79], which is one of the thesis contributions. Simulation results have shown that incorporating the *WSBM* to other search algorithms reduces their computational complexity without affecting the quality or the compression ratio of the reconstructed frames.

Table 3.2. *Improvements achieved by incorporating WSBM in MTSS and OSS*

Sequence	**Criterion**	**Improvement %**	
		MTSS	***OSS***
Tennis	Total BM evaluations	15.0	8.3
	Average first-order entropy	0.25	0.32
	Compressed file size	0.62	0.17
Flower	Total BM evaluations	13.8	6.1
	Average first-order entropy	2.48	1.62
	Compressed file size	5.6	3.24
Susie	Total BM evaluations	13.1	6.5
	Average first-order entropy	1.58	1.57
	Compressed file size	0.02	-0.05

Chapter 4

Simplex-Based Motion Estimation

The previous chapter described the motion estimation problem and its main parameters. In this chapter, a technique for block-based motion estimation using the simplex optimization method [35–37] is presented.

4.1 Introduction

In an n-dimensional optimization problem, a performance index is minimized in the n-dimensional space of the independent variables. This performance index is formulated based on the problem to be optimized. The simplex optimization is a method for solving any n-dimensional non-constrained optimization problem if the gradient of the performance index is not available. Since motion estimation can be formulated as a two-dimensional optimization problem, the simplex optimization method can be used to solve it.

This chapter provides an overview of the simplex optimization method and introduces a simplex-based block matching (*SMPLX*) algorithm. The performance and simulation results obtained are discussed and compared with results obtained using other block matching techniques.

4.2 Simplex Optimization Method

For an n-dimensional optimization problem, the simplex optimization method specifies a polyhedron of $n+1$ vertices and by using a set of operations, the polyhedron is moved away from higher-error to lower-error positions. These operations are; reflection, expansion, contraction, and reduction. The movement is achieved by locating the vertex that is associated with the highest-error value and then replacing it with another vertex of lower-error value through successive iterations. In each iteration, only one operation is performed. The simplex algorithm stops when no further improvement can be achieved (the highest-error value vertex cannot be replaced with lower-error value vertex) or no further improvement is required (the search reaches a specific maximum number of iterations or a specified minimum error value).

4.3 Simplex Optimization Method Operations

The simplex algorithm defines an initial search polyhedron of $n+1$ vertices, $X_1, X_2, \ldots, X_{n+1}$ where X is an n-dimensional vector. Using an error function , $F(X)$, to evaluate the error value at each vertex, the simplex algorithm searches for the value of X that minimizes $F(X)$ by performing successive iterations in which the reflection, expansion, contraction, or reduction operations are used. The simplex operations, initialization, and termination are as follows:

- **Initialization**

  Given an initial point X_1, the remaining points can be evaluated as follows

  $X_2 = X_1 + Te_1$

  $X_3 = X_1 + Te_2$

  .....

  $X_{n+1} = X_1 + Te_{n+1}$

  where e_i is a unit vector in the i^{th} direction and T is an arbitrary step length.

Once the initial vertices are defined, the error value at each vertex is computed, the vertex with the highest-error value and the vertex with lowest-error value are labeled X_h and X_l, respectively. In the case where $n = 2$ and the number of vertices is 3, the remaining vertex is labeled X_m.

- **Reflection**

  In the reflection operation, X_h is reflected around the other vertices. The resulting vertex is labeled X_r and is given by

$$X_r = X_c + \alpha(X_c - X_h) \tag{4.1}$$

where X_c is the centroid of all vertices in the except X_h and is given by

$$X_c = \frac{1}{N} \sum_{i=1}^{N+1} (X_i - X_h),\ i \neq h \tag{4.2}$$

and α is a reflection coefficient, $\alpha \geq 1$. The recommended value for α is 1 [35]. Figure 4.1 illustrates the reflection operation for $n = 2$.

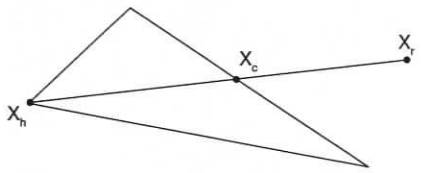


Figure 4.1. *Simplex reflection operation.*

The error value at X_r , $F(X_r)$, is evaluated and

- If $F(X_r) \leq F(X_l)$, then the expansion operation is performed next.
- If $F(X_r) > F(X_i), i \neq h$, then contraction operation is performed next.
- Otherwise, replace X_h by X_r and start next iteration with the reflection operation.

- **Expansion**

  The expansion operation follows reflection when $F(X_r) \leq F(X_l)$. In the expansion operation, the reflection vertex X_r is stretched further to an expansion vertex X_e as shown in Figure 4.2. X_e is given by

$$X_e = X_c + \gamma(X_r - X_c) \tag{4.3}$$

  where γ is an expansion coefficient. The recommended value for γ is 2 [35].

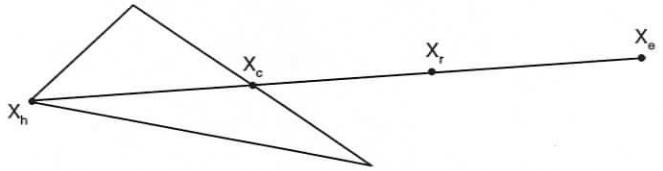


Figure 4.2. *Simplex expansion operation.*

  The error value at the expansion vertex X_e, $F(X_e)$, is evaluated and

  - If $F(X_e) \leq F(X_l)$, then replace X_h by X_e.
  - Otherwise, replace X_h by X_r. Then start the next iteration with a new reflection operation.

- **Contraction**

  The contraction operation is used when $F(X_r) > F(X_i)$, $i \neq h$. In this case, the search has moved too far from X_c so the algorithm readjusts the search near X_c by contracting X_h to X_t as shown in Figure 4.3. Before computing X_t, $F(X_r)$ is compared to $F(X_h)$. If $F(X_r) \leq F(X_h)$, then X_h is replaced by X_r. X_t is then evaluated as

$$X_t = X_c + \beta(X_h - X_c) \tag{4.4}$$

where β $(0 < \beta < 1)$ is a contraction coefficient. The recommended value for β is 0.5.

- If the resulting error value, $F(X_t) < F(X_h)$, then X_h is replaced by X_t and a new iteration with a reflection operation is started.
- Otherwise, a reduction operation will be performed in the next iteration.

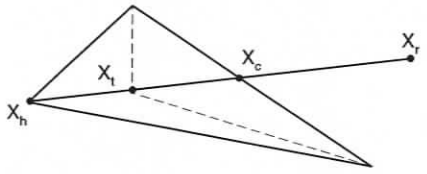


Figure 4.3. *Simplex contraction operation.*

- **Reduction**

The reduction operation is used when $F(X_r) > F(X_h)$ or $F(X_t) > F(X_h)$. In this case, the polyhedron is shrunk in the neighborhood of X_l. All the vectors $(X_i - X_l), i = 1, ..., n+1$, are reduced by one-half from X_l as given by

$$X_i = X_l + 0.5(X_i - X_l) \tag{4.5}$$

Following reduction, a new iteration is started with a reflection operation.

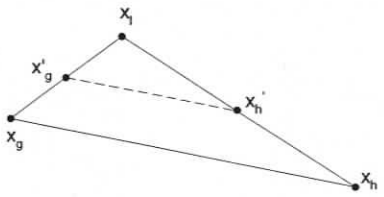


Figure 4.4. *Simplex reduction operation.*

- **Termination**

  After each search iteration, the simplex method performs a convergence test to check if the exit condition is met. The convergence test evaluates the standard deviation σ of the $(n+1)$ error values for each vertex given by

$$\sigma^2 = \frac{1}{n+1}\sum_{i=1}^{n+1}(F(X_i) - F')^2 \; where \; F' = \frac{1}{n+1}\sum_{i=1}^{n+1}F(X_i) \qquad (4.6)$$

  If σ is less than a predefined small value ε, the simplex algorithm stops based on the assumption that the last set of vertices is near the minimum point, and the search can be concluded. The complete flowchart of the simplex optimization method is shown in Figure 4.5.

4.4 Simplex-Based Block Matching Algorithm

As motion estimation can be formulated as a 2-D optimization problem, the simplex optimization method can be applied to find the best matching blocks. In the *SMPLX* algorithm, the polyhedron specified by the simplex method becomes a triangle. The algorithm uses this triangle to locate the best matching block. The error function representing the mismatch between two blocks is used as the simplex performance index. In addition, the algorithm modifies the following simplex parameters to be more suitable for motion estimation:

- Error evaluation function
- Boundary condition
- Initial search points
- Exit condition

Error Evaluation Function The error evaluation function can be any common block matching evaluation function such as MAD (equation 3.1), SAD (equation 3.2), MSE (equation 3.3), or CCF (equation 3.4). In our simulations, MAD was used as an evaluation criterion.

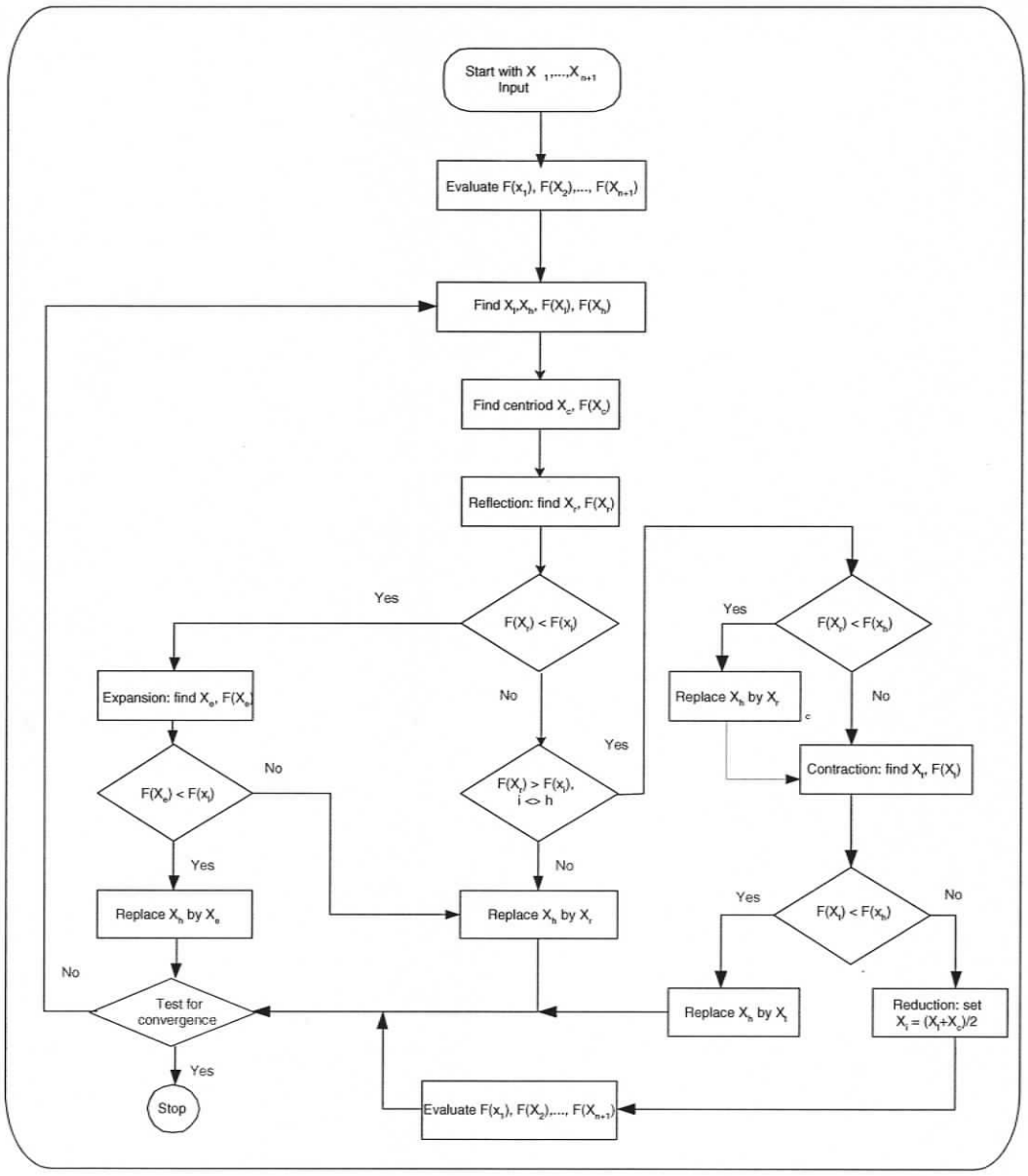


Figure 4.5. *Simplex algorithm flow chart.*

Initial Search Points The initial search points are chosen around the origin of the current macroblock or around the origin shifted by a prediction motion vector if one is available.

Boundary Condition In the *SMPLX* algorithm, a boundary condition is necessary as motion search is carried out in a limited search area. Due to this fact, if one of the search triangle vertices ends up outside the specified search area, it will be reflected back into the search area.

Exit Condition The standard-deviation-based exit condition is not very practical in motion estimation since it is very computationally expensive. An alternative approach is to check if the minimum error value obtained is less than a prespecified value. In addition, an upper limit for the maximum number of iterations per macroblock is added to make sure that the algorithm stops.

The values used for the reflection, expansion, and contraction coefficients i.e., α, γ, and β, are as recommended in [35]. These values are based on experimental results and they are as follows: $\alpha = 1$, $\gamma = 2$, and $\beta = 0.5$. The following two subsections describe the *SMPLX* algorithm and its performance based on simulation results.

In the *SMPLX* algorithm, three vertices are needed to construct the simplex triangle and each vertex is represented by a 2-D vector $X_i = (x, y)$. The MAD at each vertex is referred to by $MAD(X_i)$. The *SMPLX* algorithm is as follows:

Step 1: Select three initial vertices X_1, X_2, and X_3. Each vertex represents a best-match candidate for the current block. Those initial vertices are computed as follows:

X_1 is the search area origin

$X_2 = X_1 + e_1$

$X_3 = X_1 + e_2$

where e_j is a unit vector in the j^{th} direction and $T = 1$.

Step 2: Compute the MAD at each vertex. Identify the vertex with the highest MAD as X_h, the vertex with the lowest MAD as X_l, and the middle vertex as X_m. Initialize the iteration counter $k = 0$.

Step 3: Calculate the centroid vertex, X_c, of all the vertices excluding X_h as

$$X_c = \frac{1}{N} \sum_{i=1}^{N+1} (X_i - X_h),\ i \neq h \tag{4.7}$$

Step 4 Reflection: Compute reflection vertex, X_r, by reflecting X_h through X_c using equation 4.1. Calculate $MAD(X_r)$ as well.

Step 5 Expansion: If $MAD(X_r) \leq MAD(X_l)$, then reflection was successful. In this case, expand reflection vertex X_r to vertex X_e using equation 4.3.

If $MAD(X_e) < MAD(X_l)$, replace X_h by X_e, and go to step 3 with $k = k + 1$. Otherwise, replace X_h by X_r, and go to step 3 with $k = k + 1$. In both cases, a vertex that has a lower MAD value than X_l is found and used to replace X_h.

Step 6 Contraction: If $MAD(X_r) > MAD(X_i)$ and $i \neq h$, then the algorithm did not have successful reflections and the search triangle is likely in the neighborhood of a minimum position so the search triangle is contracted to approach that minimum position. Contract vector $(X_h - X_c)$ using equation 4.4, replace X_h by X_t, and continue from step 3 with $k = k + 1$.

Step 7 Reduction: If $MAD(X_r) > MAD(X_h)$, reduce all the vectors $(X_i - X_l)$ for $i = 1, 2, 3$, by one-half from X_l using equation 4.5, and return to step 3 with $k = k + 1$.

Termination Condition: The search is terminated when no change occurs in the vertices for two successive iterations, when the number of iterations exceeds a specified limit, or when the minimum value obtained is equal to or less than a specified error value.

4.5 Simulation Results

The *SMPLX* algorithm was implemented as part of an MPEG-2 encoder and was used to encode standard video sequences. The performance of the *SMPLX* algorithm was compared to that of the *MTSS* [74] and the *FS* algorithms. The *MTSS* was chosen for its low computational requirements and the *FS* was chosen for its capability to find the minimum search position without being trapped in a local minimum. Different video sequences with

Table 4.1. *Average number of block matching per macroblock.*

Sequence	*FS*	*MTSS*	*SMPLX*
Tennis	973.70	26.21	24.61
Flower	973.70	26.21	25.08
Susie	973.70	26.21	25.15
Football	973.70	26.21	24.07

different types of motion were used. The encoder parameters were fixed for all simulations. The sequence information and search window parameters were chosen as follows:

- GOP sequence IBBPBBPBBPBBPBB
- Frame resolution: 352×240 pixels/frame, 8 bits/pixel
- Search window range -16 to 16

The comparison criteria used were the average number of block matching evaluations, the compression ratio, and the MSE between the original frames and the reconstructed frames. The number of block matching evaluations provides an estimation of the computation complexity of the algorithm. The compression ratio and MSE give an approximation of the algorithm's capability to locate the best match.

Table 4.1 lists the average number of block matching comparisons for the different algorithms. As can be seen, the average number of block matching comparisons required by the *SMPLX* algorithm is lower than that of the *MTSS* and much less than that of the *FS*. This indicates that the *SMPLX* algorithm is less computationally expensive than the other two algorithms. The compression ratio, on the other hand, did not change much between the three algorithms as can be seen in Table 4.2.

Table 4.3 presents the first-order entropy (equation 2.5) of the prediction errors. The MSE is plotted in Figure 4.6 for the different video sequences. The figure indicates that the *SMPLX* algorithm provided better performance than the *MTSS* especially in areas that

Table 4.2. *Compression ratio results.*

Sequence	*FS*	*MTSS*	*SMPLX*
Tennis	44.91	44.22	44.44
Flower	17.27	14.95	15.14
Susie	54.96	54.93	54.95
Football	18.24	18.15	18.17

Table 4.3. *Average first-order entropy comparison results.*

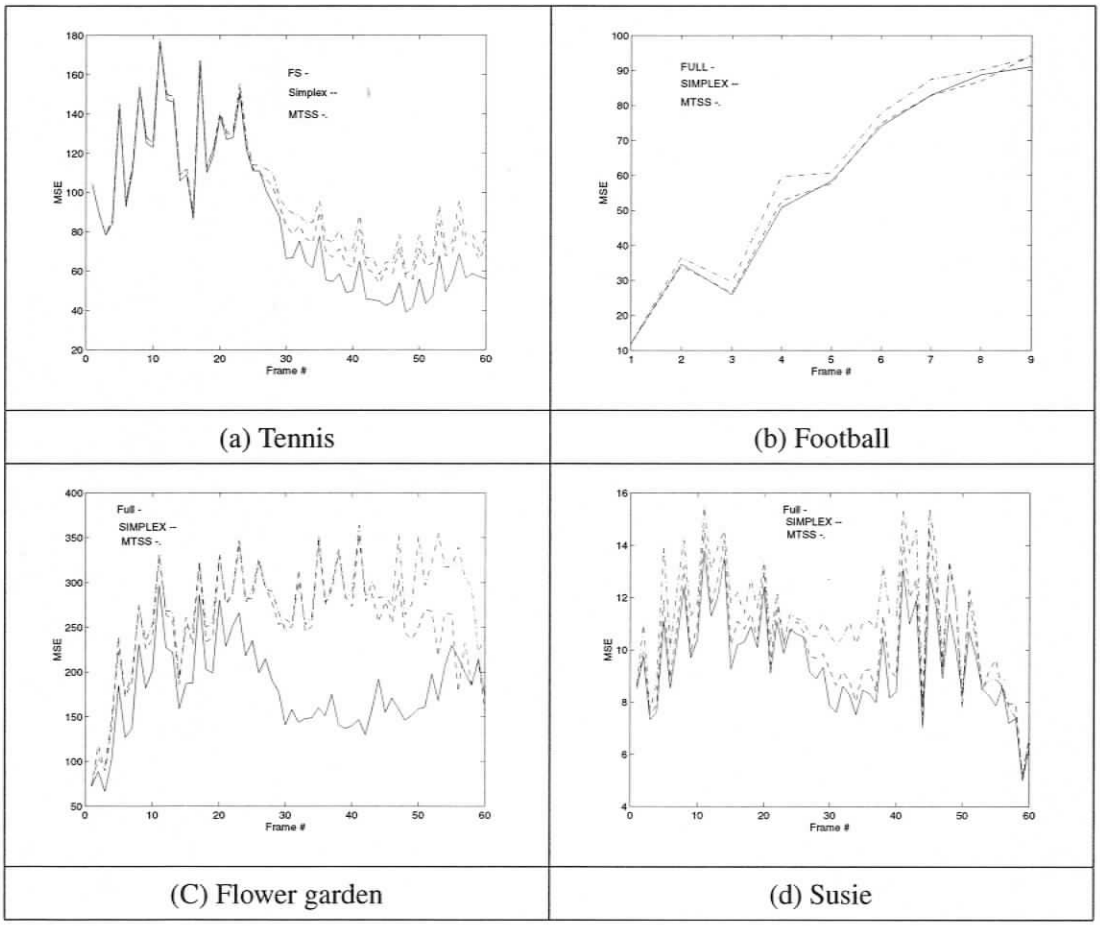
Sequence	*FS*	*MTSS*	*SMPLX*
Tennis	4.239251	4.301513	4.290848
Flower	4.745513	5.313476	5.153831
Susie	2.723500	2.820975	2.774492
Football	3.863005	3.959666	3.955665

contain fast motion as in the football sequence. The *SMPLX* algorithm also achieved comparable results to those achieved with the *FS* in terms of the visual quality of the reconstructed frames.

The *SMPLX* algorithm was published in [80]. A similar approach based on the simplex optimization method was also presented in [81–84]. The *SMPLX* and the other algorithms employ different boundary conditions and were simulated in different applications which makes them difficult to compare.

4.6 Performance of the *SMPLX* Algorithm

Simulation results indicate that the *SMPLX* algorithm is less computationally expensive than other fast BMAs without introducing any significant deterioration in the quality of the reconstructed video or compression ratio. However, a detailed analysis of the performance



(a) Tennis (b) Football

(C) Flower garden (d) Susie

Figure 4.6. *MSE for different Sequences.*

of the *SMPLX* algorithm suggests that although it offers a very good performance, it has certain disadvantages, as follows, that can be improved:

- The search triangle can collapse to a point or line
- Uses floating-point calculations
- Offers limited control over the search triangle size and exit condition

The search triangle can collapse to a point or line

The simplex optimization method was originally proposed for the continuous-parameter space. The *SMPLX* algorithm, however, is run in a discrete-parameter space. This some-

times leads to the collapse of the search triangle to a point or a line due to the digital grid approximation after vertex calculations.

Uses floating-point calculations

The *SMPLX* algorithm uses floating-point calculations to compute reflection, expansion, contraction, and reduction operations which are relatively slow compared to integer-based operations. Floating-point computations are not as significant as MAD computations. However, replacing floating-point calculations with integer-based ones would reduce the overall computational complexity of the algorithm and make it easier to implement in hardware.

Offers limited control over the search triangle size and exit condition

The movement of the search triangle in the *SMPLX* algorithm is not fully controllable. As an example, the triangle may increase dramatically in size, shrink too much, collapse to a point or a line, etc. Consequently, several search iterations may be performed without any new outcome.

4.7 Conclusions

A new block-based motion estimation algorithm based on the simplex optimization method was presented. The *SMPLX* algorithm defines a search triangle and by using various operations such as reflection, expansion, contraction, and reduction in successive iterations, the search triangle is moved, redirected, or resized until the best matching block is found. These operations provide the algorithm with a high flexibility and efficiency to locate the best-matching block by checking a small number of search positions compared to other algorithms. In addition, the flexible behavior of the *SMPLX* algorithm improves its ability to avoid being trapped in local minimum. Simulation results indicated that the *SMPLX* algorithm is less computationally expensive compared to some other block matching algorithms such as the *FS* and the *MTSS* without degrading other compression factors such as quality of the reconstructed frames or the compression ratio.

Chapter 5

Flexible Triangle Search for Block-Based Motion Estimation

The previous chapter described the simplex optimization method and introduced the *SMPLX* algorithm. In this chapter, a new algorithm for block-based motion estimation, the flexible triangle search (*FTS*) algorithm is presented.

5.1 Introduction

The *FTS* algorithm is a simplex-based block-based motion estimation that uses a search triangle to locate the minimum error. In the *FTS* algorithm, the use of the search triangle was motivated by the simplex optimization method used in the previous chapter and the *FTS* algorithm was further adapted to the discrete grid used in motion estimation.

The *FTS* algorithm was compared to other fast algorithms such as the diamond search (*DS*) and the hexagon search (*HS*) as well as the *SMPLX* algorithm and it was found to be much more efficient than the other algorithms. Considering that the *DS* and the *HS* are two of the most efficient motion search algorithms, it can be concluded that the *FTS* algorithm is one of the top motion search algorithms available.

5.2 The *FTS* Algorithm

The objective of the *FTS* algorithm is to find the best matching block using a specified matching criterion such as the MAD or the SAD described in chapter 3. It uses sets of triangles of different sizes to perform the search. Like the *SMPLX* algorithm, the *FTS* algorithm uses reflection, expansion, and contraction operations to move or resize the triangle. In addition, it uses a new operation, translation, to move the triangle to another position. The *FTS* defines several triangle levels. Each level includes triangles of the same size but different orientations. Triangles in the same level reflect to each other to simulate the simplex reflection operation. The *FTS* switch between different levels to simulate the expansion and contraction operations. Figure 5.1 displays the triangles for levels 0 to 2. More than 3 levels could be used, however, experimental results have shown that 3-4 levels are entirely satisfactory for the commonly used search window sizes. The vertices of the triangles in these levels are always on the integer grid. Each triangle is defined by its identification id and its level, i.e., $T21$ stands for triangle T, level 2, and id 1.

The triangle ids for the three levels are:

$Level\ 0 = \{T00, T01, T02, T03\}$

$Level\ 1 = \{T10, T11, T12, T13, T14, T15\}$

$Level\ 2 = \{T20, T21, T22, T23, T24, T25\}$

To facilitate the representation of the search triangles in look-up tables (LUTs), the vertices of these triangles are denoted as V_0, V_A, V_B where V_0 is the center point and V_A, V_B are the vertices in counterclockwise sense from V_0. Thus, the coordinates of the three vertices of a triangle can be obtained from the triangle id and the coordinates of V_0.

Based on the above notation, the basic operations of the *FTS* algorithm, reflection, expansion, contraction, and translation can be represented using LUTs and can be computed without floating point operations. Table 5.1 lists level 0 triangles, and their possible reflections and expansions which are also illustrated in Figures 5.2 and 5.3 respectively. Tables 5.2 and 5.3 list level 1 and level 2 triangles respectively. Similar tables for reflec-

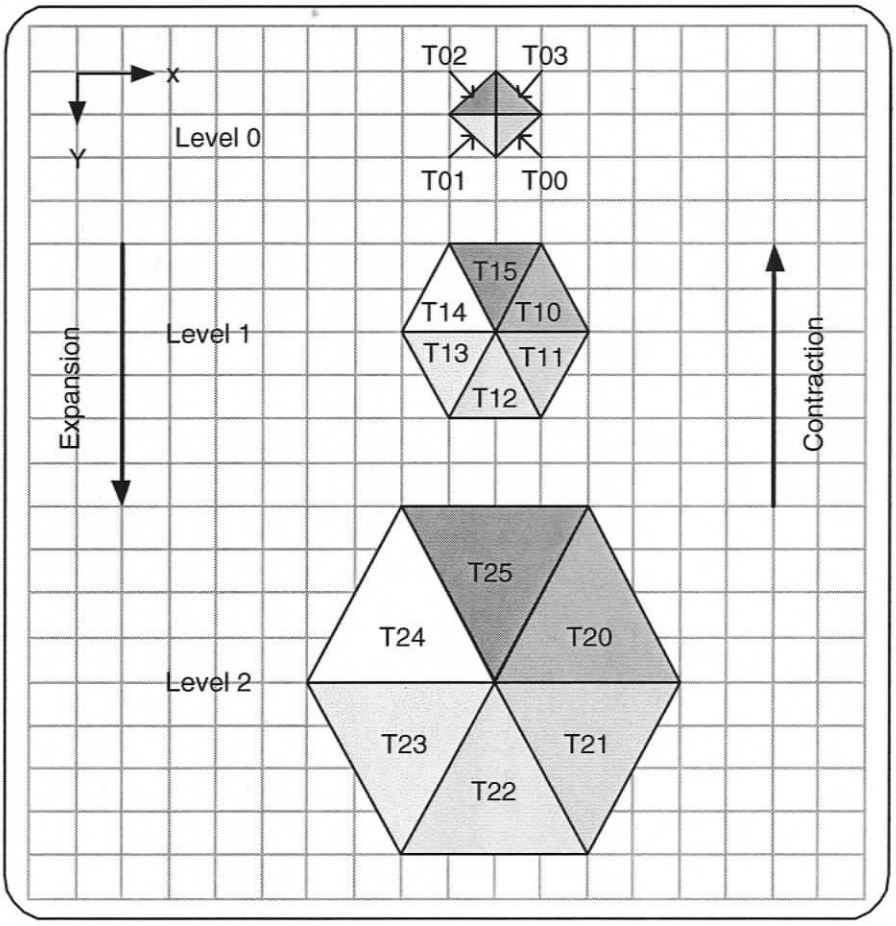


Figure 5.1. *Triangle sets for levels 0 to 2.*

tion and expansion can be constructed for the other levels. The important implication of these tables is that the *FTS* algorithm can be implemented using these tables and all search operations can then be done using integer calculations. Thus the computational efficiency can be greatly increased.

Contraction from level 2 to 1 is straightforward since the triangle orientation does not change. However, contraction from level 1 to level 0 requires some modifications since the number of triangles in level 0 is less than the number of triangles in level 1 as shown in

Table 5.1. *Group 0 look-up table of the FTS algorithm.*

		V_0 reflection around V_A and V_B		Expansion of V_0 reflection-vertex		V_A reflection around V_0 and V_B		Expansion of V_A reflection-vertex		V_B reflection around V_0 and V_A		Expansion of V_B reflection-vertex	
Current Triangle Level 0		New Triangle Level 0	Origin Shift V0	Test Point Ve	New Triangle Level 1	New Triangle Level 0	Origin Shift V0	Test Point Ve	New Triangle Level 1	New Triangle Level 0	Origin Shift V0	Test Point Ve	New Triangle Level 1
T00		T02	(1,1)	(2,2)	T14	T03	(0,0)	(0,-2)	T12	T01	(0,0)	(-2,0)	T11
T01		T03	(-1,1)	(-2,2)	T10	T00	(0,0)	(2,0)	T13	T02	(0,0)	(0,-2)	T12
T02		T00	(-1,-1)	(-2,-2)	T11	T01	(0,0)	(0,2)	T15	T03	(0,0)	(2,0)	T14
T03		T01	(1,-1)	(2,-2)	T13	T02	(0,0)	(-2,0)	T10	T00	(0,0)	(0,2)	T15

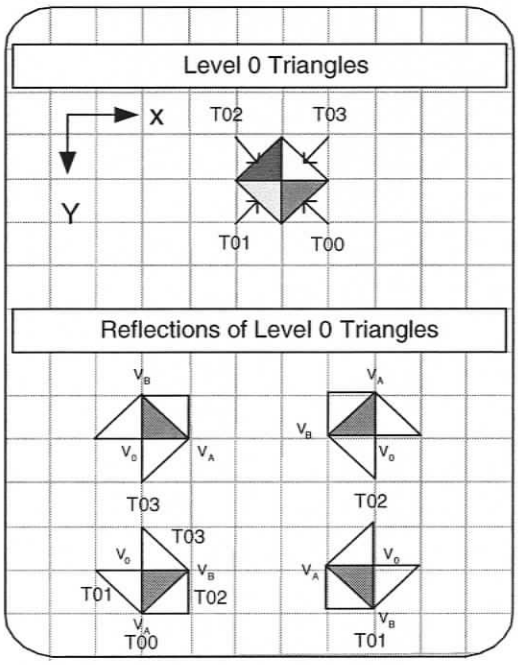


Figure 5.2. *Possible reflections for level 0 triangles. The original triangle is the dark one.*

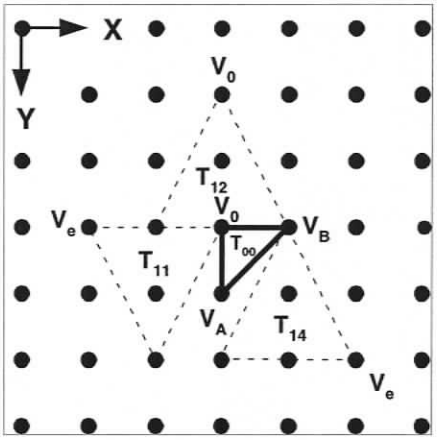


Figure 5.3. *Result of reflection followed by expansion of triangle T00. T00 is shown using solid lines and the resulting level 1 triangles are shown using dotted lines.*

Table 5.2. *Group 1 look-up table of the FTS algorithm.*

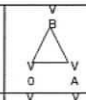
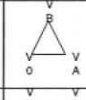
		V_0 reflection around V_A and V_B		Expansion of V_0 reflection-vertex		V_A reflection around V_0 and V_B		Expansion of V_A reflection-vertex		V_B reflection around V_0 and V_A		Expansion of V_B reflection-vertex	
Current Triangle Level 1		New Triangle Level 1	Origin Shift V0	Test Point Ve	New Triangle Level 2	New Triangle Level 1	Origin Shift V0	Test Point Ve	New Triangle Level 2	New Triangle Level 1	Origin Shift V0	Test Point Ve	New Triangle Level 1
T10		T13	(3,-2)	(5,-3)	T23	T15	(0,0)	(-3,-3)	T25	T11	(0,0)	(1,4)	T21
T11		T14	(3,2)	(5,3)	T24	T10	(0,0)	(1,-4)	T20	T12	(0,0)	(-3,3)	T22
T12		T15	(0,4)	(0,6)	T25	T11	(0,0)	(4,-1)	T21	T13	(0,0)	(-4,-1)	T23
T13		T10	(-3,2)	(-5,3)	T20	T12	(0,0)	(3,3)	T22	T14	(0,0)	(-1,-4)	T24
T14		T11	(-3,-2)	(-5,-3)	T21	T13	(0,0)	(-1,4)	T23	T15	(0,0)	(3,-3)	T25
T15		T12	(0,-4)	(0,-6)	T22	T14	(0,0)	(-4,1)	T24	T10	(0,0)	(4,1)	T20

Table 5.3. *Group 2 look-up table of the FTS algorithm.*

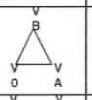
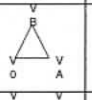
		V_0 reflection around V_A and V_B		Expansion of V_0 reflection-vertex		V_A reflection around V_0 and V_B		Expansion of V_A reflection-vertex		V_B reflection around V_0 and V_A		Expansion of V_B reflection-vertex	
Current Triangle Level 2		New Triangle Level 2	Origin Shift V0	Test Point Ve	New Triangle Level 3	New Triangle Level 2	Origin Shift V0	Test Point Ve	New Triangle Level 3	New Triangle Level 2	Origin Shift V0	Test Point Ve	New Triangle Level 3
T20		T23	(6,-4)	(8,-5)	T33	T25	(0,0)	(-4,-4)	T35	T21	(0,0)	(2,5)	T31
T21		T24	(6,4)	(8,5)	T34	T20	(0,0)	(2,-5)	T30	T22	(0,0)	(-4,4)	T32
T22		T25	(0,8)	(0,9)	T35	T21	(0,0)	(6,2)	T31	T23	(0,0)	(-6,-2)	T33
T23		T20	(-6,4)	(-8,5)	T30	T22	(0,0)	(4,4)	T32	T24	(0,0)	(-2,-5)	T34
T24		T21	(-6,-4)	(-8,-5)	T31	T23	(0,0)	(-2,5)	T33	T25	(0,0)	(4,-4)	T35
T25		T22	(0,-8)	(-8,5)	T32	T24	(0,0)	(-6,2)	T34	T20	(0,0)	(6,-2)	T30

Table 5.4. *Contraction from level 1 to level 0 Triangles.*

Original triangle (level 1)	New triangle (level 0)
T10	T03
T11	T00
T12	T00
T13	T01
T14	T02
T15	T02

Figure 5.1. The contractions from level 1 to 0 are listed in Table 5.4.

Like most other algorithms, the *FTS* algorithm is easily integrated with early termination and motion vector prediction techniques to improve its computational performance. When motion prediction is used, the predictive motion vector is used as the center of the starting triangle group. In addition, an early termination condition based on the SAD value is added.

The *FTS* algorithm can now be described as follows:

Given a reference frame $S_{l-1}(x, y)$, an $M \times N$ macroblock in the current frame $S_l(x, y)$, find the displacement vector $V_{min} = (dx, dy)$ so that $SAD(V_{min})$ in equation 5.1 is minimized.

$$SAD(dx, dy) = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} |S_l(x, y) - S_{l-1}(x + dx, y + dy)| \tag{5.1}$$

The details of the *FTS* algorithm are as follows:

Step 1: Initialization

Initialize the current triangle level, current triangle within that set, and initial triangle vertices V_0, V_A, and V_B in the search area. Choose V_0 as the origin of the search window. If motion vector prediction is used, shift V_0 by the predictive motion vector. Initialize the

iteration counter $K = 0$ and set translation vector V_d to 0 and displacement vector V_{min} to V_0.

Step 2: Check termination

- Check termination conditions. If any termination condition is satisfied, then terminate the search.
- Determine the SAD for each new vertex in the current triangle. Identify the vertex with the highest SAD value as V_h, the vertex with the lowest SAD value as V_l, and the vertex with the middle SAD value as V_{mid}.
- If the previous step was a successful expansion or translation operation, go to step 6; otherwise continue to step 3.

Step 3: Reflection

- Get a new vertex V_r by reflecting V_h of the current triangle using the table corresponding to the current level and calculate $SAD(V_r)$.
- If $SAD(V_r) < SAD(V_h)$, go to step 4; otherwise go to step 5.

Step 4: Expansion

- Locate the expansion vertex V_e for the current triangle using the appropriate triangle level table.
- If $SAD(V_e) < SAD(V_r)$, then expansion was successful; increase the triangle level and update the current triangle. Calculate the translation vector between the reflection and expansion vertices, V_d, using $V_d = V_e - V_r$.
- If $SAD(V_e) < SAD(V_{min})$, set $V_{min} = V_e$ and go to step 2 with $K = K + 1$.
- If $SAD(V_e) \geq SAD(V_r)$, then expansion was not successful. Update the current triangle by replacing V_h by V_r. If $SAD(V_r) < SAD(V_{min})$ set $V_{min} = V_r$ and go to step 2 with $K = K + 1$.

Step 5: Contraction

- If the current level is 0, then no more contractions can be done. In this case, terminate the search. Otherwise, contract the triangle by reducing the triangle level. Update the current triangle and set $K = K + 1$; then go to step 2.

Step 6: Translation

- Find a new vertex, V_t, by translating V_l using $V_t = V_l + V_d$ and calculate $SAD(V_t)$.
- If $SAD(V_t) < SAD(V_l)$, then translation was successful; replace V_l by V_t, set $K = K + 1$. If $SAD(V_l) < SAD(V_{min})$, set $V_{min} = V_l$ and go to step 2.
- If $SAD(V_t) \geq SAD(V_l)$, then translation was not successful; set V_l as the origin of the next search triangle and $K = K + 1$ and go to step 2.

Termination Conditions: The search is terminated if

- no more successful contraction operations are possible
- the number of search iterations reaches a pre-specified limit (Kmax)
- the value of SAD becomes less than a pre-specified threshold ExitSAD

The relationship between the *FTS* algorithm operations and triangle levels is illustrated in Figure 5.4 and the complete flow chart of the algorithm is shown in Figure 5.5.

Based on the above description of the *FTS* algorithm, the effect of the basic operations is as follows:

1. Each reflection operation moves the triangle away from positions of large error using only one SAD calculation while most other fast algorithms require several SAD calculations.
2. Expansion operation speeds up the search by increasing the step and thus avoiding unnecessary intermediate SAD calculations. The contraction operation reduces the search step to achieve a higher resolution.
3. The translation operation is useful when a translational motion is detected during the search. The *FTS* algorithm uses the translation operation if a successful expansion

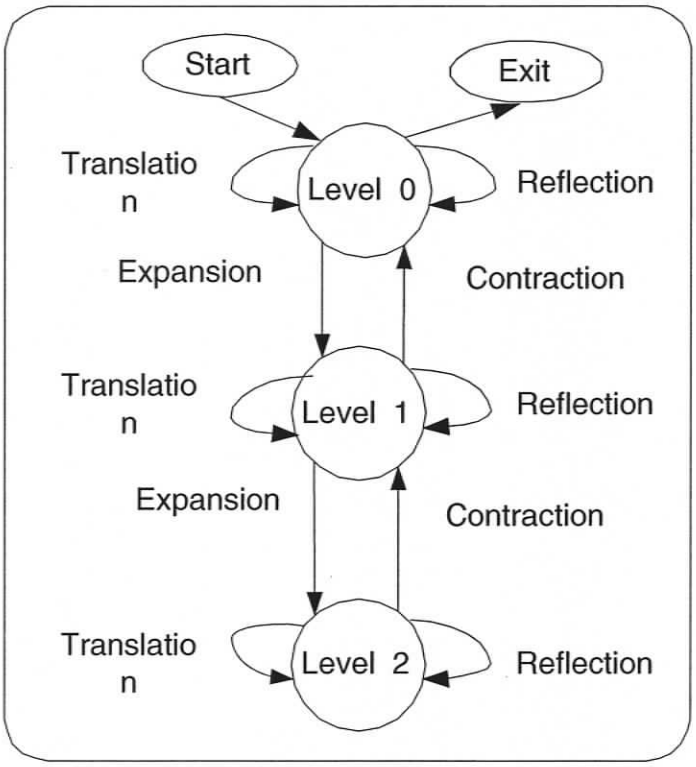


Figure 5.4. *Relation between the FTS operations reflection, expansion, contraction and triangle levels.*

follows a successful reflection in which case the chosen direction of expansion may yield a better minimum point. These operations provide the algorithm with excellent search flexibility. Further, the reflection operation can help to avoid inferior local minima.

The *FTS* algorithm has the following useful features

- It is optimized for an integer grid and for performing all operations using integer calculations.
- Most of the calculations are predefined in look-up tables, which can easily be accessed during the search process.

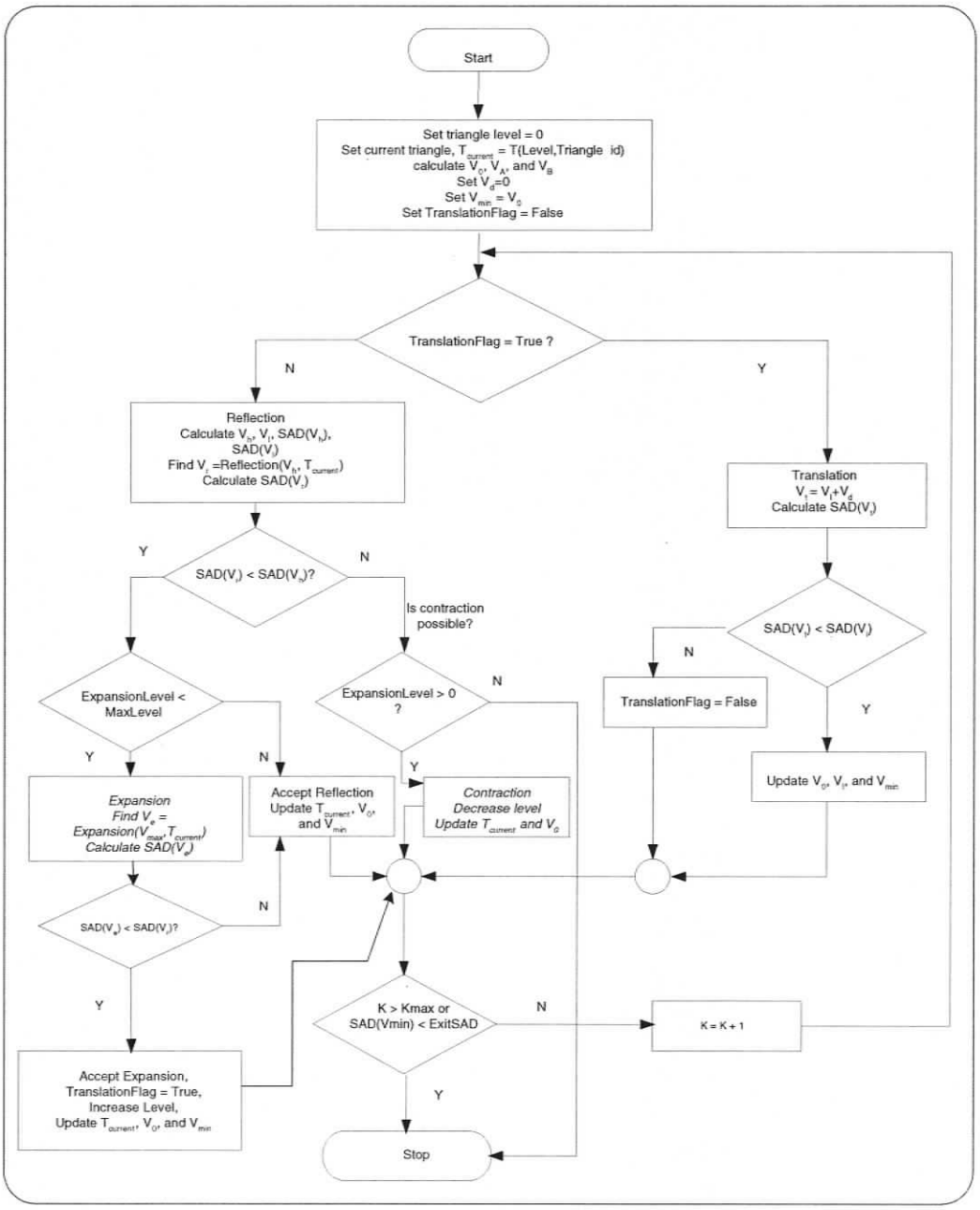


Figure 5.5. *Flowgraph of the FTS algorithm.*

- Search triangles are predefined and, consequently, it is not possible for two or three vertices to coincide and thus the triangle cannot collapse to a line or point.
- The flow control and exit conditions are robust.

5.3 Illustrative Example

An example of the search pattern using the *FTS* algorithm is illustrated in Figure 5.6. The search starts at the origin of the search window and concludes by finding V_{min} which represents the location with the minimum SAD. The steps involved are as follows:

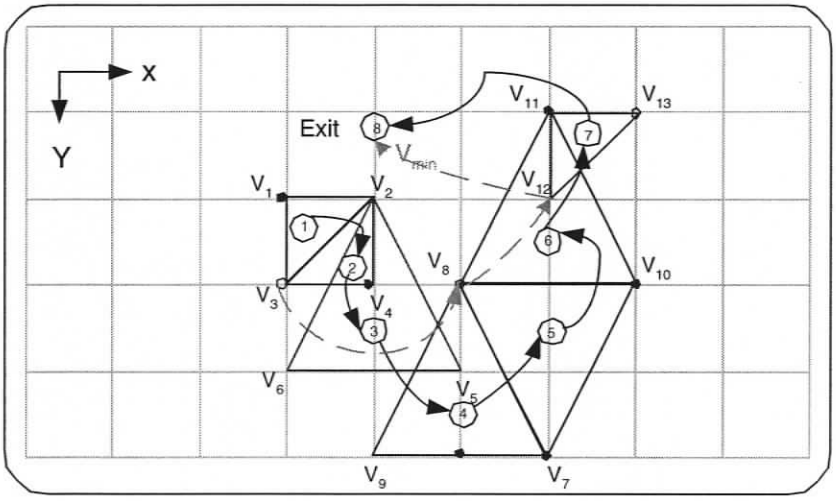


Figure 5.6. *Example of a search pattern using the FTS algorithm.*

1. Start: The triangle search starts at level 0; the current triangle is $T00$ with initial vertices V_1, V_3, and V_2. In this case $SAD(V_1)$ is the maximum and $SAD(V_3)$ is the minimum. Thus, V_1 is set equal to V_h, V_3 to V_l and V_{min} to V_3.
2. Reflection: The triangle vertex V_1 is reflected to V_4. Since $SAD(V_4) < SAD(V_1)$, reflection is successful and should be followed by expansion. The new triangle becomes $T02$.

3. Expansion: An expansion operation is performed at V_5 and since $SAD(V_5) < SAD(V_4)$, expansion is considered successful. The current triangle is then expanded to $T14$ (based on Table 5.1) with vertices V_2, V_5, and V_6. V_d is calculated from $V_d = V_e - V_r = (1, 1)$. Since in this case, $SAD(V_5) > SAD(V_{min})$, V_{min} will not be updated.

4. Translation: Since the last operation was successful, translation is attempted. Using the translation vector $V_d = (1, 1)$ from the expansion step, a translation of the current triangle is attempted to V_7, V_8, and V_9. In this case, $SAD(V_9)$ is the maximum error, $SAD(V_8)$ is the minimum error and it is less then $SAD(V_{min})$. As a result V_{min} is updated to V_8. The triangle id remains $T14$.

5. Reflection: Since the last operation was a successful translation and the second translation attempted did not lead to a vertex with a lower error than SAD(V8), a reflection is attempted by reflecting V_9 to V_{10}. Since $SAD(V_{10}) < SAD(V_9)$, this operation is a successful reflection. In the reflected triangle, $SAD(V_7)$ is the maximum error. Further, $SAD(V_{10}) > SAD(V_8)$ so V_8 remains the minimum point and V_{min} is not updated. The new triangle becomes $T15$.

6. Reflection: Expansion is not successful, so reflection is attempted by reflecting V_7 to V_{11}. Since $SAD(V_{11}) < SAD(V_8) < SAD(V_7)$, the reflection was successful and also V_{min} is updated to V_{11}. The new triangle becomes $T12$.

7. Contraction: Expansion and reflection are not successful and thus contraction is attempted. Based on Table 5.4, $T12$ is contacted to $T00$. In the new triangle, $SAD(V_{12})$ is the lowest and is also lower than $SAD(V_{min})$. Thus V_{min} is updated to V_{12}.

8. Exit: An additional reflection does not lead to lower values for SAD. In addition, it is not possible to contract to a lower level. The algorithm exits with the location of the minimum SAD value in V_{min}.

5.4 Performance Analysis

The *FTS* algorithm was integrated as part of the JVT/H.264 reference encoder. In the earlier version of the algorithm, which was published in [85], the H.263 was used in the simulation. The *FTS* algorithm was compared to the *NTSS* [55], the *FS*, the *DS* [59], and the *HS* [64] algorithms. The *NTSS* is well known for its simplicity while the *DS* and the *HS* are well known for their low computational requirements. For purposes of comparison, scenes with different kinds of movement have been used. In addition, both QCIF and CIF resolutions were used.

Except for the search algorithm, all other encoding parameters were kept fixed. These parameters include:

- Macroblock size (16×16)
- Search area size (16×16)
- Rate control algorithm
- Motion vector prediction is included
- Early exit condition when SAD value become less than a specified value(ExitSAD)
- Number of I and P frames per each test sequence

Table 5.5 lists the average number of block matching comparisons per frame obtained. As can be seen, the average number of block matching comparisons required by the *FTS* algorithm is less than that of the *NTSS*, the *FS*, the *DS*, and the *HS*. These results indicate that the *FTS* algorithm is more computationally efficient than any of the other three techniques.

The compression ratio results in Table 5.6 indicate that the *FTS* algorithm produced slightly less compression ratio than the *FS* and comparable results to the *DS*, the *HS*, and the *NTSS*. In all cases, the difference in compression ratio was within 1%, which is almost negligible given the reduction achieved in the amount of computation.

The average $PSNR(Y)$ results with and without using rate control are presented in Table 5.8 and Table 5.7, respectively. For a qualitative comparison, Figure 5.7 shows the

Table 5.5. *Average number of block matching per macroblock.*

Sequence	***FS***	***NTSS***	***DS***	***HS***	***FTS***
QCIF resolution (176x144)					
Miss America	1089	20.00	14.76	12.65	9.04
Akyio	1089	17.34	12.24	11.26	6.51
News	1089	17.49	12.42	11.35	6.83
Silent	1089	17.67	12.59	11.46	7.23
Coastguard	1089	18.15	12.70	11.73	7.37
Foreman	1089	19.11	13.62	12.26	8.20
Carphone	1089	19.33	14.11	12.60	8.44
Stefan	1089	19.03	13.6	12.36	8.33
CIF resolution 352×288					
Coastguard	1089	18.38	12.9	11.86	7.37
Container	1089	18.69	13.55	12.02	7.32
Foreman	1089	20.37	15.09	13.36	9.32
Paris	1089	17.73	12.59	11.52	7.06
Stefan	1089	21.07	15.60	13.69	9.30

reconstructed frames using the different algorithms. The visual difference is hardly noticeable. It can be inferred from Tables 5.8 and 5.7 and Figures 5.7 and 5.8 that the $PSNR(Y)$ values obtained using the *FTS* algorithm are comparable to these of the *NTSS*, the *DS*, and the *HS*. In addition, these values are also very close to those of the *FS*.

Figure 5.8 displays the $PSNR(Y)$ values for each frame while Figure 5.9 shows the changes of $PSNR(Y)$ at different bit rates. As shown, the *FTS* algorithm is comparable to that of other algorithms except for the *FS*.

From the above comparison, it is clear that the compression ratios, as well as the average $PSNR$ and visual quality of the reconstructed frames for all algorithms used in the

Table 5.6. *Compression ratios.*

Sequence	*FS*	*NTSS*	*DS*	*HS*	*FTS*
QCIF resolution 176 × 144					
Miss America	279.67	279.19	276.33	278.67	280.37
Akyio	312.43	312.50	312.91	313.16	313.00
News	105.21	105.14	105.17	105.34	104.85
Silent	91.64	91.39	91.60	91.40	91.38
Coastguard	38.54	38.50	38.50	38.51	38.42
Foreman	54.42	54.82	54.60	54.49	54.70
Carphone	49.90	49.91	49.86	49.75	49.89
Stefan	13.84	13.80	13.80	13.77	13.70
CIF resolution 352 × 288					
Coastguard	413.36	413.69	414.13	413.40	413.63
hline Container	31.94	31.97	31.96	31.97	31.96
Foreman	174.38	173.47	173.88	173.54	173.34
Paris	68.61	68.20	67.95	67.03	67.32
Stefan	64.37	64.24	64.22	64.15	64.04

comparison are not significantly different. This indicates that the significant reduction of the computational complexity obtained using the *FTS* algorithm did not come at the expense of visual quality deterioration or compression efficiency reduction.

The estimated reduction in the number of block matching computations when the *FTS* algorithm is used is around 30-60% depending on the video sequence.

In order to verify the stability and consistency of the *FTS* algorithm for different sequences, the performance of the *FTS* and other algorithms for different sequences is plotted in Figure 5.10. The use of LUTs to store the vertex data improved the computational efficiency of the algorithm. The required memory to store these LUTs is approximately

Table 5.7. *Average $PSNR(Y)$ (with rate control disabled)*

Sequence	***FS***	***NTSS***	***DS***	***HS***	***FTS***
QCIF resolution 176×144					
Miss America	39.62	39.63	39.63	39.64	39.61
Akyio	37.8	37.79	37.77	37.77	37.77
News	36.26	36.26	36.23	36.24	36.25
Silent	35.45	35.46	35.47	35.47	35.47
Coastguard	33.85	33.86	33.86	33.85	33.86
Foreman	34.93	34.91	34.9	34.9	34.9
Carphone	36.03	36.02	36.01	35.99	35.98
Stefan	33.76	33.76	33.76	33.75	33.76
CIF resolution 352×288					
Coastguard	39.31	39.30	39.31	39.31	39.32
Container	34.32	34.30	34.3	34.30	34.29
Foreman	35.54	35.53	35.53	35.53	35.52
Paris	35.89	35.89	35.87	35.86	35.87
Stefan	35.11	35.12	35.11	35.12	35.12

200 bytes, which is very insignificant given current advances in memory sizes and the total memory allocated by the encoder.

Table 5.9 compares the number of block matching evaluations required by the *FTS* algorithm relative to those required by the *NTSS*, *DS*, and *HS* algorithms.

5.5 Conclusions

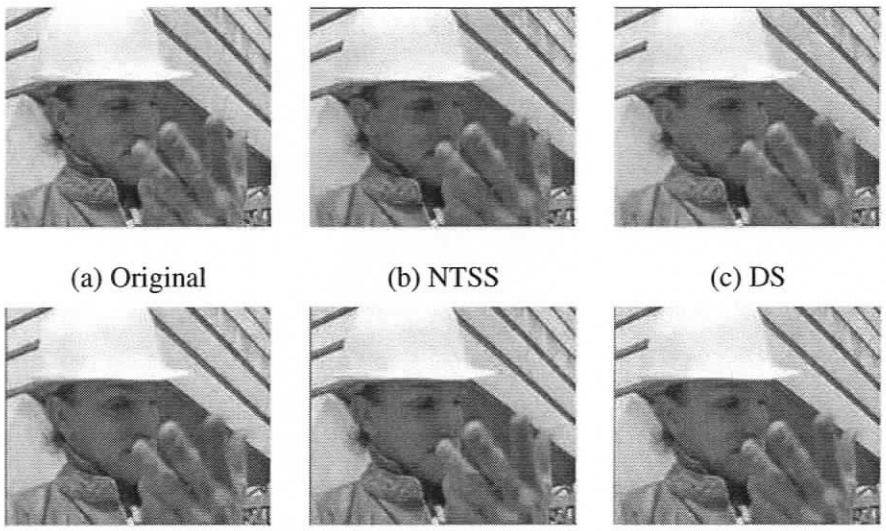
The *FTS* algorithm for block-matching motion estimation was introduced. The algorithm is based on the simplex optimization method and is adapted for a discrete grid. The *FTS*

Table 5.8. *Average $PSNR(Y)$ (with rate control enabled)*

Sequence	*FS*	*NTSS*	*DS*	*HS*	*FTS*
QCIF resolution 176×144					
Miss America	44.16	44.14	44.12	44.14	44.15
Akyio	45.39	45.4	45.41	45.38	45.44
News	39.14	39.14	39.13	39.14	39.21
Silent	37.96	37.98	37.95	37.96	37.94
Coastguard	31.51	31.49	31.48	31.49	31.47
Foreman	36.84	36.85	36.84	36.83	36.85
Carphone	38.16	38.13	38.14	38.13	38.13
Stefan	28.55	28.5	28.51	28.5	28.46
CIF resolution 352×288					
Coastguard	43.36	43.35	43.36	43.35	43.33
Container	32.61	32.61	32.60	32.60	32.59
Foreman	36.46	36.46	36.45	36.46	36.44
Paris	35.82	35.77	35.74	35.68	35.72
Stefan	35.07	35.08	35.01	35.04	34.97

algorithm uses a set of triangles of different sizes to perform the reflection, expansion, contraction, and translation operations. These operations enable the *FTS* algorithm to quickly change the search direction and switch between coarser and finer searches.

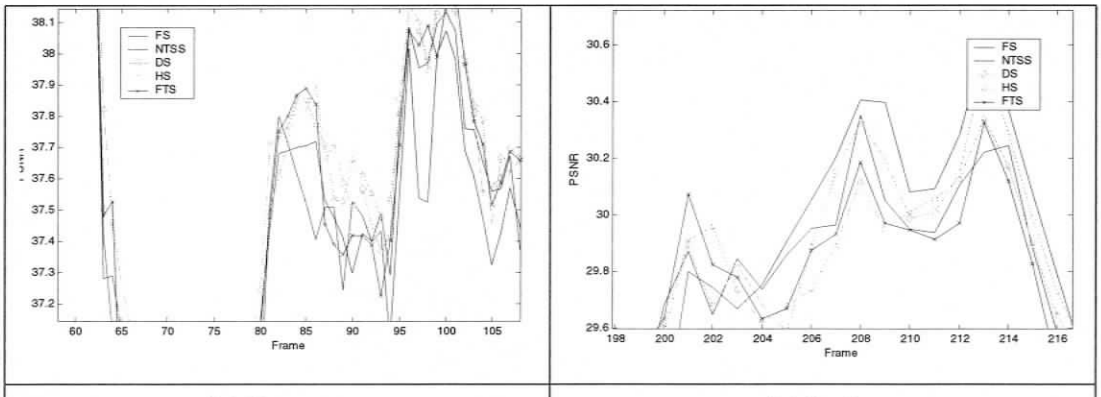
The proposed technique was implemented as part of H.264 encoder and compared with some other popular block-matching algorithms. Results indicated that the proposed technique requires significantly less block matches than other fast algorithms without any significant reduction in the compression ratio or deterioration of the visual quality of the reconstructed frames. The *FTS* algorithm results were published in [85, 86] and a patent application for the algorithm has been submitted by the University of Victoria Innovation



(a) Original (b) NTSS (c) DS

(d) FS (e) FTS (f) HS

Figure 5.7. *Reconstructed frame number 255 for Foreman QCIF.*



(a) Foreman (b) Stefan

Figure 5.8. *$PSNR$ value per each frame.*

and Development Corporation [87].

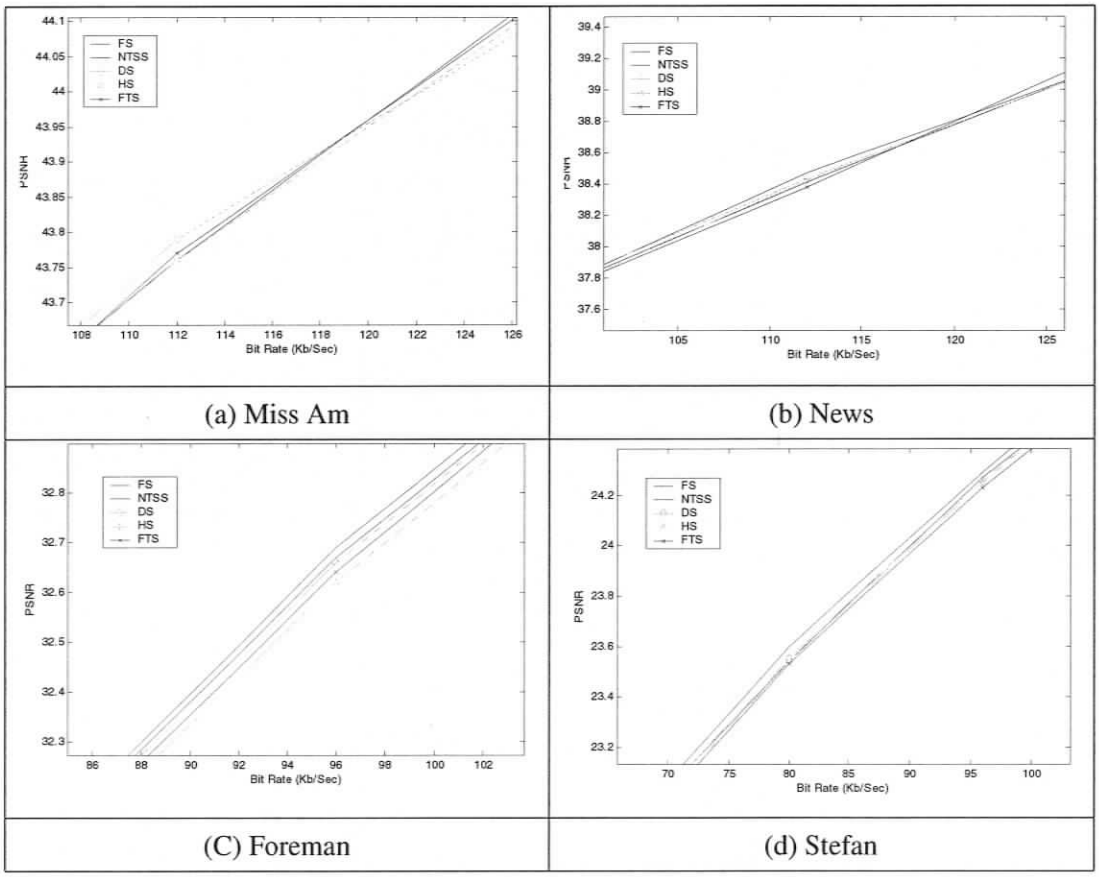


Figure 5.9. *$PSNR$ verses bit rate.*

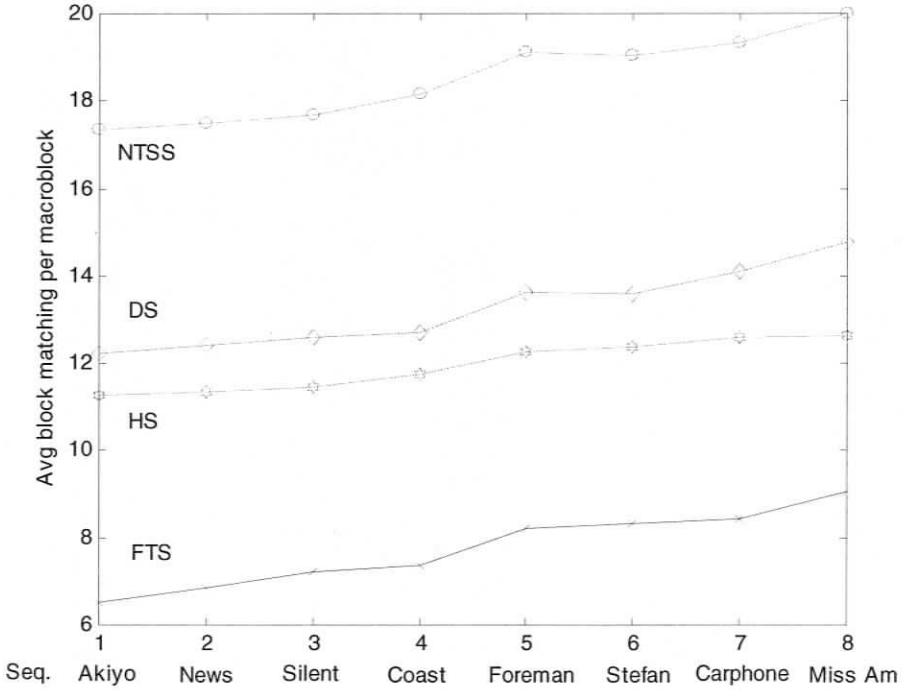


Figure 5.10. *Average number of block matching per macroblock for each algorithm.*

Table 5.9. *Computational improvement of the FTS algorithm relative to other algorithms.*

Sequence	**Number of block matching in *FTS***	**Improvement with respect to other algorithms, %**		
		NTSS	***DS***	***HS***
Miss America	9.04	54.80	38.75	28.54
Akyio	6.51	62.46	46.81	42.18
News	6.83	60.95	45.01	39.82
Silent	7.23	59.08	42.57	36.91
Coastguard	7.37	59.39	41.97	37.17
Foreman	8.20	57.09	39.79	33.12
Carphone	8.44	56.34	40.18	33.02
Stefan	8.33	56.23	38.75	32.61

Chapter 6

Extensions of the *FTS* Algorithm for Improved Block-Based Motion Estimation

This chapter presents several improvements to the *FTS* introduced in the previous chapter.

6.1 Introduction

The *FTS* algorithm described in the previous chapter has demonstrated exceptional performance. Further analysis of the algorithm's behavior has led to additional enhancements which will be described in this chapter. These enhancements are as follows:

- Reducing the amount of computation and increasing the accuracy of the *FTS* by avoiding repeated computations and premature exit.
- Extending the *FTS* to support a combined full-pixel and half-pixel motion search.
- Improving the selection of the initial triangle by adding prediction techniques.

6.2 Enhanced *FTS*

Analysis of the *FTS* performance showed that the *FTS* algorithm sometimes exits prematurely at the initialization stage if a reflection operation fails. Further, it is possible for

the search triangle to revisit the same search position more than once, especially if those positions lie on different triangles at different levels.

This section describes an enhanced *FTS* (*EFTS*) which uses a reduced number of block matching operations by avoiding repeated SAD computations. In addition, the *EFTS* increases the accuracy of the motion vectors obtained by avoiding premature exit.

6.2.1 *EFTS* Improvements

The proposed enhancements consist of the following:

- An extra condition to check whether the required SAD has been previously computed or not.
- An extra operation to load the stored SAD value or to save the newly computed one.

The reduction in the number of block matching operations is based on keeping track of the visited search positions for possible computational reuse. As shown in Figure 6.1, in order to avoid repeated calculations, each visited search position is recorded and the corresponding error value, SAD, is stored in a special memory buffer, the SAD buffer. Then, whenever a new SAD value calculation is needed, the SAD buffer is checked to see whether the required SAD value has already been calculated. If the checked condition is satisfied, the pre-computed SAD value is loaded from the buffer and used; otherwise the new SAD value is computed and then stored in the SAD buffer. A similar approach is also used in the H.264 encoder implementation [88].

The amount of computation required by the added conditions, to check whether the SAD value exists, is insignificant compared to the amount of computation needed for each new SAD evaluation. For example, for a 16×16 pixel macroblock, each SAD operation requires 3 operations for each pixel. These operations are subtraction, evaluation of absolute value, and addition. For the entire macroblock, a total of $16 \times 16 \times 3$ or 768 operations are needed. In our proposed approach, a buffer is used to store the calculated SAD values which adds only two operations per search point, check for the existence of the SAD value

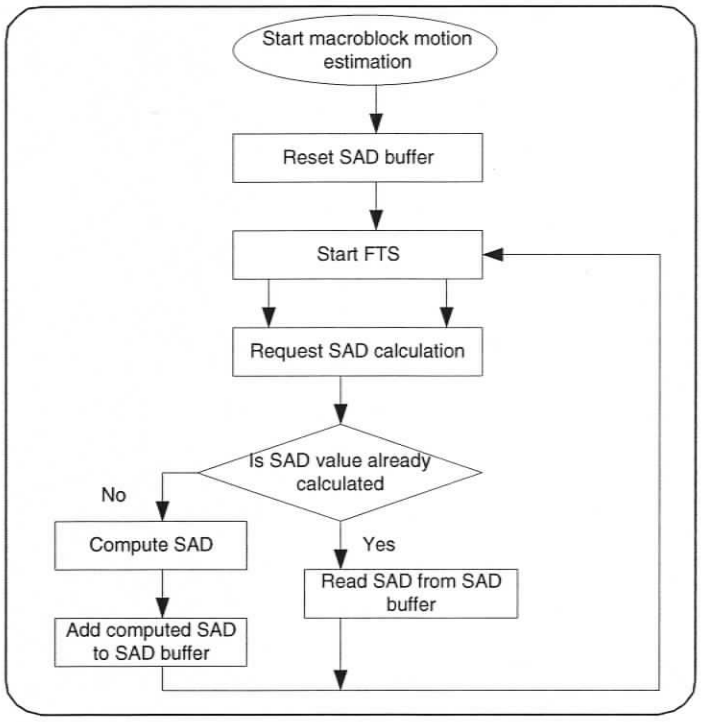


Figure 6.1. *SAD buffer flowchart.*

and load the old value or store the new one. The size of the SAD buffer is similar to the search area size and the SAD buffer is indexed by x, y coordinates.

The second enhancement leads to more accurate motion vector estimation by preventing the *FTS* from a premature exit. In the *FTS*, the contraction operation is executed whenever a reflection operation fails and if contraction is not possible, then the *FTS* exits the search. Analysis of the *FTS* showed that the algorithm will likely exit prematurely if reflection moves the triangle outside the search boundary. In order to avoid this situation, the contraction operation step of the *FTS* has been modified so that if contraction is not possible, the vertex with the second highest SAD is reflected before exiting the search for this macroblock. This condition is performed only at the beginning of the search, where the number of search iterations is less than 1.

Table 6.1. *Average number of block matching per macroblock (QCIF resolution).*

Sequence	*DS*	*MTSS*	*SMPLX*	*FTS*	*EFTS*
Foreman	14.69	21.49	14.84	7.19	**6.70**
Akyio	12.02	21.48	14.45	5.78	**5.57**
Coast	13.79	21.51	14.48	6.02	**5.96**
Miss America	14.90	21.54	15.68	7.77	**7.11**
News	12.66	21.46	14.26	6.15	**5.85**
Carphone	14.94	21.46	15.22	7.00	**6.56**
Silent	13.18	21.46	14.37	6.35	**6.12**

6.2.2 *EFTS* Simulation Results

The *EFTS* and the *FTS* were implemented as part of an H.263 encoder. The *EFTS* technique was compared with the *FTS*, *MTSS*, *DS*, and *SMPLX* algorithms.

Table 6.1 lists the average number of block matching operations per macroblock for the algorithms under consideration.

Tables 6.2 and 6.3 list results with respect to the $PSNR$ and the compression ratios, respectively.

It can be seen from the three tables that the *EFTS* requires a significantly smaller number of block matching operations than other algorithms and leads to a slightly improved compression ratio while maintaining almost the same visual quality of the reconstructed video sequence.

6.3 Half-Pixel *FTS*

Fractional pixel, usually half-pixel, accuracy is used in motion estimation to improve the motion estimation outcome. Half-pixel accurate motion estimation is very useful in increasing the $PSNR$ and reducing blocking artifacts in the reconstructed frames. Moreover, the

Table 6.2. *$PSNR$ comparison (QCIF resolution).*

Sequence	Algorithm	$PSNR(Y)$	$PSNR(U)$	$PSNR(V)$
Craft	*FTS*	31.64	34.06	35.06
	EFTS	31.63	34.02	35.06
	SMPLX	31.68	34.10	35.08
	MTSS	31.57	34.08	35.01
	DS	31.64	34.03	35.02
Foreman	*FTS*	30.81	35.96	36.36
	EFTS	30.82	35.89	36.4
	SMPLX	30.80	35.91	36.45
	MTSS	30.78	35.88	36.35
	DS	30.86	35.91	36.38
Akiyo	*FTS*	33.70	35.97	38.53
	EFTS	33.71	35.80	38.40
	SMPLX	33.70	35.92	38.37
	MTSS	33.71	35.8	38.46
	DS	33.71	35.78	38.43
Coast	*FTS*	29.93	38.90	38.91
	EFTS	29.97	38.85	39.02
	SMPLX	29.90	38.97	38.92
	MTSS	29.89	38.94	38.91
	DS	29.98	38.98	39.01

Table 6.3. *Compression ratio (QCIF resolution).*

Sequence	*DS*	*MTSS*	*SMPLX*	*FTS*	EFTS
Craft	44.10	40.96	38.44	41.91	**42.31**
Foreman	85.03	75.45	76.08	79.09	**81.44**
Akiyo	491.59	488.64	491.08	491.21	**492.82**
Coast	81.04	73.49	76.09	77.84	**80.87**
Miss America	254.42	237.84	264.59	269.58	**260.26**

compressed file size can also be reduced. Half-pixel search, however, requires additional SAD computations and search area interpolation in order to get a half-pixel resolution as shown in Figure 6.2 and thus motion estimation complexity is further increased.

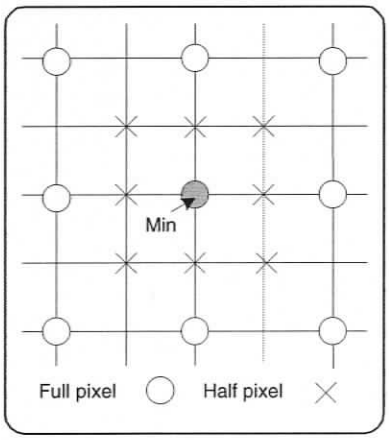


Figure 6.2. *Half-pixel search positions.*

In order to reduce half-pixel search complexity, most search algorithms perform half-pixel accurate motion-estimation in two separate and successive stages; full-pixel stage and half-pixel stage. Full-pixel search is done first on the specified search area and a full-pixel minimum is obtained. Half-pixel is performed next to full-pixel search by checking the eight positions around the full-pixel minimum position as shown in Figure 6.2. A full half-

pixel search algorithm evaluates the SAD values for all eight positions, Other techniques select a small subset of these eight positions for half-pixel evaluation [89].

Generally, the required number of half-pixel evaluations is less than the required number of full-pixel evaluations for each macroblock. However, due to the computational efficiency of the *FTS* algorithm, it was observed that the amount of computation required for half-pixel search is comparable to that of full-pixel search. Consequently, half-pixel search is a computation bottleneck. In order to improve the performance of half-pixel motion estimation, a half-pixel based *FTS* algorithm (*HP-FTS*) is proposed. The *HP-FTS* and its performance will be discussed in the following subsections. The *HP-FTS* was compared to the *FTS* algorithm with complete half-pixel search and the parabolic prediction-based half-pixel search ($PPHPS$) algorithm which uses a half-pixel approximation [89].

6.3.1 *HP-FTS* Description

In the *HP-FTS*, interpolation is performed with respect to the search area in order to achieve half-pixel resolution. Then motion search is performed directly in the interpolated search area. This approach would appear to be computationally expensive due to the interpolation operation which should be done for each macroblock. Fortunately, interpolation is also required in other parts of the encoder such as motion compensation and thus the interpolation operation is unavoidable. In fact, some of the existing video encoders implementations automatically carry out interpolation for the entire reference frame before encoding any new frame.

Based on the above observations, the proposed *HP-FTS* performs the motion search directly in the interpolated search area. Since interpolation increases the search area size, the use of level 0 as starting level may slow down the search. An alternative approach is to start with either level 0 or level 1 based on the majority vote criterion from the surrounding blocks as follows:

1. Initialize two counters: $useLevel_0$ and $useLevel_1$ to 0.

2. Check the motion vector for each of the left, top-left, and top macroblocks. If $x < 1$ and $y < 1$ increment $useLevel_0$ counter; otherwise increment $useLevel_1$.
3. If $useLevel_0 \leq useLevel_1$, then start the *FTS* with level 0; otherwise start the *FTS* with level 1.

6.3.2 Simulation Results

The *HP-FTS* was implemented as part of an H.263 encoder. The proposed technique was compared to three other search methods as follows:

1. *FTS-FP*: *FTS* at full-pixel resolution without half-pixel search
2. *FTS-FHP*: *FTS* at full-pixel resolution with full half-pixel search
3. *FTS-PPHPS*: *FTS* at full-pixel resolution with $PPHPS$ half-pixel approximation algorithm. $PPHPS$ usually performs half-pixel search using at most three half-pixel evaluations.

Table 6.4 lists the average number of block matching operations per frame. Tables 6.5 and 6.6 present results of $PSNR(Y)$ and compression ratio, respectively.

Table 6.4. *Average number of block matching per frame.*

Sequence	***FTS-FP***	***FTS-FHP***	***FTS-PPHPS***		***HP-FTS***	
	Number of SAD	**Number of SAD**	**Number of SAD**	**Improvement %**	**Number of SAD**	**Improvement %**
Craft	816	1575	1209	23.24	1240	21.27
Miss America	594	1384	990	28.47	848	38.73
Akiyo	448	1215	795	34.57	471	61.23
Silent	499	1263	865	31.51	635	49.72
Coast	489	1281	901	29.66	885	30.91
Carphone	546	1315	935	28.90	870	33.84
Foreman	585	1354	972	28.21	990	26.88

Table 6.5. *Average $PSNR(Y)$ comparison.*

Sequence	***FTS-FP***	***FTS-FHP***	***FTS-PPHPS***	***HP-FTS***
Craft	30.00	31.75	31.52	31.62
Miss America	34.89	36.45	36.44	36.44
Akiyo	32.83	33.70	33.58	33.66
Silent	30.76	31.73	31.66	31.70
Coast	28.77	29.93	29.76	29.83
Carphone	29.95	31.77	31.73	31.72
Foreman	29.11	30.81	30.71	30.72

Table 6.6. *Percentage improvement in file size.*

Sequence	**File size**	**Improvement %**		
	FTS-FP	***FTS-FHP***	***FTS-PPHPS***	***HP-FTS***
Craft	95983	13.37	9.39	12.10
Miss America	21853	3.19	4.81	10.23
Akiyo	29138	20.31	15.67	20.10
Silent	72122	14.45	12.08	14.16
Coast	189805	22.80	17.27	18.00
Carphone	200790	16.23	14.27	15.02
Foreman	256952	25.17	20.92	22.83

The results in Table 6.4 indicate that the improvement in the number of SAD evaluations using the *HP-FTS* compared to the *FTS-FHP* is between 20-60%. This improvement is higher than that of the *FTS-PPHPS*, which is around 20-30%. When comparing *HP-FTS* to *FTS-FP*, the number of SAD evaluations has increased by 4-70% depending on the sequence. However, the quality of the reconstructed sequence is higher and the file size is

smaller relative to those achieved with *FTS-FP*.

From Table 6.5, we note that the average $PSNR$ for the test sequences using the *HP-FTS* has improved over *FTS-FP* by an average of 1.26 dB compared to an average of 1.22 dB improvement over the *FTS-FP* using the *FTS-PPHPS* and 1.33 dB using the *FTS-FHP*.

The results in Table 6.6 indicate that the average improvement over the *FTS-FP* for the file size was 16.09% using *HP-FTS* compared to 16.66% using the *FTS-FHP* and 13.64% using the *FTS-PPHPS*. These results indicate that the file sizes using *HP-FTS* are slightly larger than those obtained using *FTS-FHP* in return for a reduction in the number of SAD evaluations by around 39%.

6.4 Prediction of the Initial Triangle in the *FTS*

This section describes the addition of a prediction mechanism to the *FTS* algorithm (*PFTS*) to efficiently select the initial search triangle and thus the initial direction of the search. This prediction enables the algorithm to move quicker to an area where the minimum SAD position is likely to be found.

6.4.1 Prediction Description

The *FTS* starts the search using triangle $T00$ and as shown in Figure 6.3, level 0 has four triangles. Each triangle lies in one quadrant around the search center. Starting the search with each of these triangles can lead to different results. Below, a procedure is proposed to find out which triangle among these four should be used as the initial triangle.

The SAD values for the four positions surrounding the search center are evaluated. Then, the two SAD values of the vertices that lie in the same quadrant are added up. Since there are four quadrants, there are also four values for the added SAD values in each quadrant. The quadrant that has the minimum added SAD value is chosen as the starting search quadrant. Consequently, the triangle that lies in this quadrant is chosen as the starting search triangle.

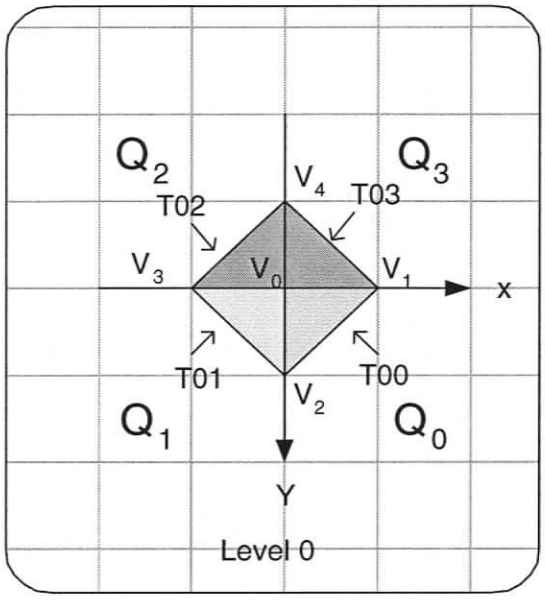


Figure 6.3. *Selection of starting triangle in level 0.*

The steps to select the initial triangle are as follows:

1. Calculate the SAD values for V_i for $i = 1, 2, 3, 4$.
2. Calculate $SAD(Q_i)$ for each quadrant Q_i as

$$SAD(Q_i) \quad = \quad SAD(V_{i+1}) + SAD(V_{i+2}), \tag{6.1}$$

   where $i = 1, 2, 3, 4$

3. Select the quadrant that has the minimum SAD_{Q_i}, Q_{min}.
4. Select the triangle that lies in Q_{min} as the *FTS* starting triangle.

6.4.2 Simulation Results

The *PFTS* was implemented as part of an H.264 encoder. The technique was compared with the *FTS* [85], *NTSS* [55], and *DS* [59]. Table 6.7 lists the average number of block

matching operations per macroblock. Tables 6.8 and 6.9 list results with respect to the compression ratio and the $PSNR$, respectively.

Table 6.7. *Average number of block matching per macroblock and improvement of PFTS over FTS.*

Sequence	**Number of block matching**			***PFTS***	
	DS	***NTSS***	***FTS***	**Number of block matching**	**Improvement %**
Miss America	14.76	20.00	9.04	7.83	13.38
Akyio	12.24	17.34	6.51	5.85	10.14
News	12.42	17.49	6.83	6.17	9.66
Silent	12.59	17.67	7.23	6.26	13.42
Coastguard	12.70	18.15	7.37	6.65	9.77
Foreman	13.62	19.11	8.20	7.56	7.80

Table 6.8. *PFTS: Compression ratio (QCIF frame).*

Sequence	***DS***	***NTSS***	***FTS***	***PFTS***
Miss America	276.33	279.19	280.37	280.69
Akyio	312.91	312.50	313.00	313.16
News	105.17	105.14	104.85	105.13
Silent	91.60	91.39	91.38	91.03
Coastguard	38.50	38.50	38.42	38.44
Foreman	54.60	54.82	54.70	54.65
Carphone	49.86	49.91	49.89	49.95
Stefan	13.80	13.80	13.70	13.77

It can be seen from the three tables that the *PFTS* requires a significantly smaller number of block matching operations than other algorithms and leads to a slightly improved

Table 6.9. *PFTS:* $PSNR$ *comparison per (QCIF frame).*

Sequence	**Algorithm**	$PSNR(Y)$	$PSNR(U)$	$PSNR(V)$
Stefan	*FTS*	33.76	36.04	35.73
	PFTS	33.75	36.04	35.75
	NTSS	33.76	36.03	35.72
	DS	33.76	36.03	35.73
Foreman	*FTS*	34.90	38.81	40.03
	PFTS	34.91	38.80	40.04
	NTSS	34.91	38.82	40.02
	DS	34.90	38.83	40.05
Akiyo	*FTS*	37.77	40.69	41.52
	PFTS	37.77	40.69	41.52
	NTSS	37.79	40.74	41.51
	DS	37.77	40.74	41.53
Coast	*FTS*	33.86	41.87	43.64
	PFTS	33.86	41.88	43.74
	NTSS	33.86	41.92	43.68
	DS	33.86	41.94	43.62
Miss America	*FTS*	39.61	39.10	38.93
	PFTS	39.61	39.10	38.96
	NTSS	39.63	39.11	38.94
	DS	39.63	39.15	38.97

compression ratio for almost the same visual quality of the reconstructed video sequence.

6.5 Conclusions

Three additional improvements to the *FTS* algorithm were discussed. These improvements include a more resilient exit criterion, efficient reuse of previously computed SAD values (the *EFTS*), selection of a better starting search triangle (the *PFTS*), and the integration of half-pixel and full-pixel searches (the *HP-FTS*). The *EFTS* requires a reduced number of SAD computations due to caching of computed SAD values, and is more resilient to premature exit than the *FTS*. The *HP-FTS* reduces the number of SAD computations by around 20-60% compared to the *FTS* with separate half-pixel search by incorporating full-pixel and half-pixel searches. The *PFTS* reduced the required number of SAD computations by 7 to 13% relative to that of the *FTS*. Results of the proposed improvements have been published in [90–92].

Chapter 7

Hardware Design for Motion Estimation

In this chapter, two hardware implementations for block-based motion estimation are presented. These implementations are based on the *FTS* and the *FS* algorithms. Both implementations were developed, modeled, verified, and synthesized for Xilinx FPGA using VHDL.

7.1 Introduction

The *FS* was chosen for implementation because of its regular flow, low control overhead, and suitability for hardware implementation. The *FTS*, on the other hand, was chosen for its low computational requirements as demonstrated in chapter 5. There are several implementations of the full search in [93, 94] as well as implementations of other fast search algorithms such as the hierarchal search [95], *TSS* [96–98], and *DS* [99].

The number of SAD computations is an important measure for the computational complexity of any motion estimation algorithm and its suitability for hardware implementation. However, there are other important factors such as the algorithm's regularity, suitability for pipelining and parallelism, control logic, memory throughput, and number of hardware units or gates needed, etc. [11]. These factors affect power consumption and the cost of the implementation. The choice of a specific hardware design is always a trade off between performance, speed, and cost.

7.2 Hardware Architecture for the *FTS* Algorithm

In this section, a hardware implementation for the *FTS* is presented. The *FTS* requires fewer SAD computations per macroblock. On the other hand, it requires complex control logic for switching between operations, different triangle levels, and different triangles, etc. The advantages of using *FTS* for motion estimation hardware implementation over other algorithms can be summarized as follows:

1. The *FTS* requires fewer SAD computations than many other fast BMA algorithms as demonstrated in chapters 5 and 6.
2. All *FTS* operations are done using integer calculations and LUTs which makes the hardware fast.
3. The flow control of the *FTS* algorithm can be represented by an algorithmic state machine (ASM) which can easily be modeled and synthesized.

7.2.1 Design Description

The *FTS* algorithm can be partitioned into two main units, regular and irregular parts. The regular part is used for computing SAD values that are common for all operations. The irregular part performs different computations depending on the operation to be performed. It also determines the next operation to be performed or the next positions in the search area for which SAD values should be evaluated. This irregular part can be represented by an algorithmic state machine ($FTS-ASM$) with the states shown in Figure 7.1. Table 7.1 describes the role of each state as well as the number of SAD computations required.

Based on the above discussion, two main hardware units should exist in the proposed design, namely, the $FTS-ASM$ and the SAD computing unit. Additional units can be added to increase the efficiency of the implementation by exploiting more parallelism and pipelining within the algorithm or reducing data transfer overhead. For example, caching of the current macroblock and the search area can be employed to reduce access to the external frame memory.

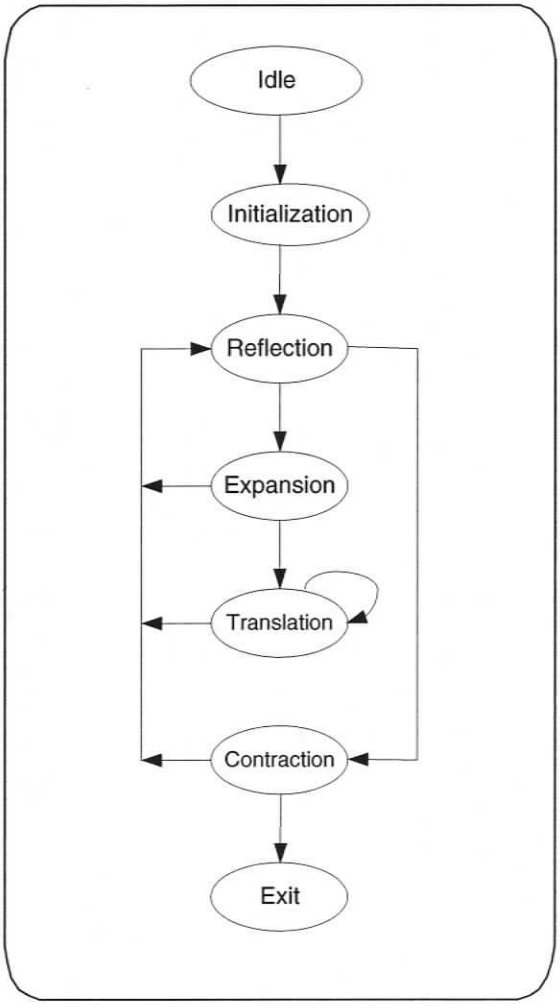


Figure 7.1. *FTS algorithmic state machine (FTS-ASM).*

The $FTS - ASM$ has 7 states (idle, initialization, reflection, expansion, contraction, translation, and exit) and it is required to perform the following tasks:

- Management and generation of the search triangle position, size, and vertices using LUTs.
- Selection of current and next triangle states
- Deciding the number of SAD computations needed (1 or 3)

Table 7.1. *Number of SAD computations for each triangle state.*

State	Description	number of SAD computation
Idle	waiting to start	0
Initialization	compute initial search triangle	3
Reflection	compute reflection vertex	1
Expansion	compute expansion vertex	1
Translation	compute translation vertex	1
Contraction	compute contraction vertices	3
Exit	end the search and output results	0

- Computing the coordinates (x, y) of the vertex or vertices to be evaluated
- Managing state change after the completion of each operation
- Output the final result when the search is completed

SAD computations can be performed in parallel and/or pipeline. A special hardware unit is used to compute the SAD as shown in Figure 7.2. This unit is designed using 1-D systolic arrays, which provide an efficient way for computation parallelism and pipelining. For a block size B, the SAD unit has B parallel units for absolute value calculations and additions, and each of these units produces a SAD value along a row. The number of processing elements (PE) in this array is equal to the number of rows in the macroblock. Following these units, there are multi-stage pipelined adders that add all the B results and produce the final 2-D SAD value. The number of these adder stages is $\log B$.

Since the number of SAD computations can be 1 or 3 depending on the search triangle state, as shown in Table 7.1, pipelining several SAD computations can be applied to make the hardware utilization more efficient than having several parallel SAD units which are not fully utilized all the time.

Based on the above discussion, a complete hardware architecture for the *FTS* is shown

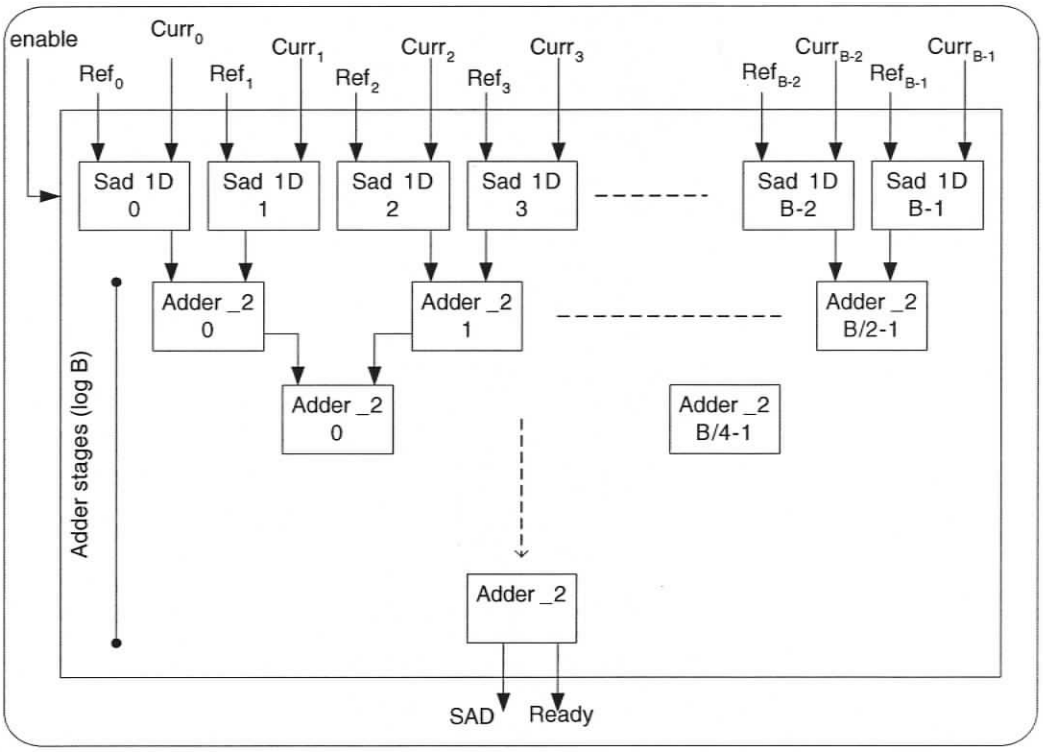


Figure 7.2. *SAD array design.*

in Figure 7.3. This architecture uses caching techniques for the current macroblock and the search area in order to improve the hardware performance. Memory caches are loaded with the correct data during the idle state of the $FTS - ASM$. In addition, a separate unit for computing and keeping track of the minimum SAD value is included. The hardware units used in the *FTS* are as follows:

1. Search area cache: This is a memory block that contains the search area data from the reference frame and is loaded before macroblock search starts. The size of this memory is $(2P + B)^2$ bytes where B is the block size and $(2P + 1)^2$ is the search area.

2. Current macroblock cache: This memory of $B \times B$ bytes holds the current mac-

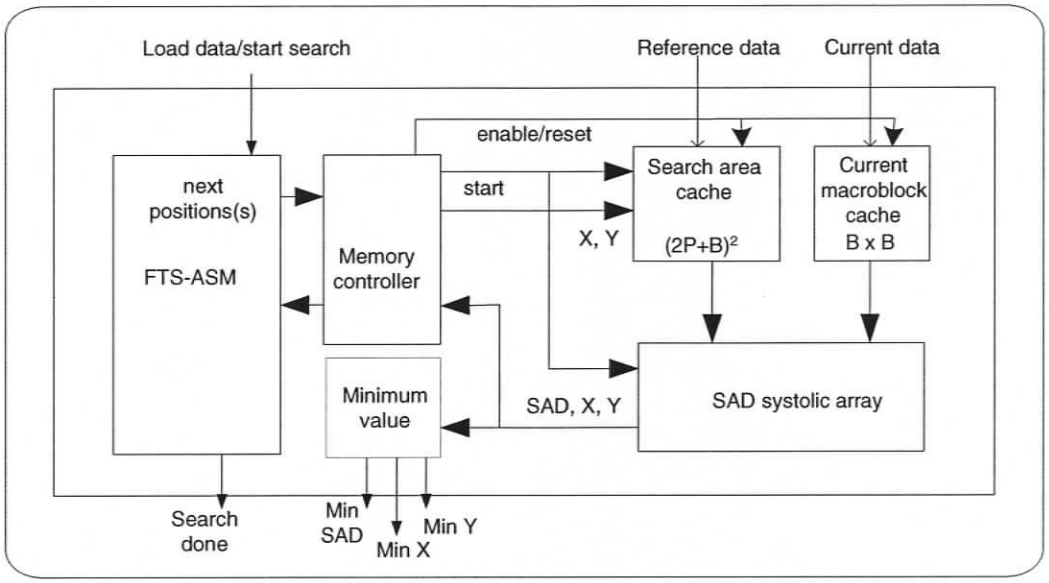


Figure 7.3. *The FTS hardware units.*

roblock data and is also loaded before macroblock search starts.

3. $FTS - ASM$: This is an algorithmic state machine of the form described before.
4. SAD systolic array: The SAD array is as described previously.
5. Minimum value: This unit keeps track of the minimum SAD value and the corresponding motion vectors.
6. Memory controller: This unit is considered an interface between the $FTS - ASM$ and the memory blocks and performs the following tasks:
   - Checks if the required search position is out of the search area boundary. In this case, the unit returns a maximum SAD value.
   - Pipelines the SAD data from the search area cache and current macroblock cache to the SAD array.
   - Returns the computed SAD values back to the $FTS - ASM$.

7.2.2 Simulation Results

The proposed implementation was modeled, simulated, and verified in VHDL and synthesized using Xilinx ISE 7.1. The device used for synthesizing was Xilinx Spartan 3 which has 8320 total combinational logic blocks (CLBs). Table 7.2 lists the number of cycles used by each state or operation for different block sizes. The hardware specifications for different macroblock sizes (B) produced by the synthesizer is shown in Table 7.3.

Table 7.2. *Number of cycles for each operation at different block sizes.*

FTS-ASM State	B=16	B=8	B=4
Idle	38	30	26
Initialization	57	32	22
Reflection	25	16	12
Expansion	25	16	12
Translation	25	16	12
Contraction	57	32	22
Exit	1	1	1

Table 7.4 compares the proposed design and the *FS* designs presented in [93, 94] for a macroblock size of 8. These full search designs were selected for comparison as they target similar FPGA implementations. The authors in [94] present two different architectures; $AB2$ and $AS2$. The table also includes the maximum supported frame rate at CIF resolution.

Results indicate that the proposed *FTS* hardware supports a much higher frame rate as compared to other *FS* implementations. This is because the *FTS* algorithm requires fewer SAD evaluations as compared to the *FS* designs. The *FTS* uses a slightly higher number of gates than the *FS* implementation presented in [93] and the one in FS-AB2 [94]. This is due to the use of local memory caches for the reference search area and current macroblock. In return, the *FTS* has fewer clock cycles per macroblock.

Table 7.3. *The FTS hardware specifications at different block sizes.*

Parameter	**Result**		
Macroblock size	**B=4**	**B=8**	**B=16**
Used CLBs (Out of 8320)	1344	2725	6142
FPGA utilization %	16	32	73
Maximum frequency (MHz)	84	84	78
Clock cycle per macroblock	72	148	202
Maximum CIF frame rate (frame/sec.)	184	358	975

Table 7.4. *Comparison between the proposed FTS and other FS designs for macroblock size = 8.*

Design	**[93]**	**AB2 [94]**	**AS2 [94]**	**Proposed**
Used CLBs	939	948	3732	**2725**
Maximum frequency (MHz)	110	30	22	**84**
Clock cycle per macroblock	2042	379	190	**148**
Maximum CIF frame rate (frame/sec.)	34	50	73	**358**

7.3 Hierarchical Hardware Design for Full Search Motion Estimation

The previous section presented a hardware implementation for the *FTS* algorithm. In this section, a hardware design for the *FS* is presented for the purpose of comparing its performance to the performance of the *FTS* hardware implementation. The *FS* was chosen because it is considered a bench mark for block matching algorithms.

7.3.1 Hierarchical Full Search Design

The motion estimation problem shown in Figure 3.2 can be simplified by decomposing it into hierarchical or multistage levels. The current frame is divided into $B \times B$ macroblocks. The current block is compared with other candidate blocks in the reference frame using a search area (window) with size $(2P + B)^2$.

The motion vector $\mathbf{v}$ associated with the minimum SAD value for the current block at origin $c(i, j)$, $\mathbf{v}(i, j) = [k^*, l^*]^t$ can be represented by

$$\begin{aligned} \mathbf{v}(i,j) &= \begin{bmatrix} k^* & l^* \end{bmatrix}^t \\ &= \arg\min_{k,l} [SAD(i,j,k,l)]; \qquad -P \leq k,l \leq P \qquad (7.1) \\ SAD(i,j,k,l) &= \sum_{m=0}^{B-1} \sum_{n=0}^{B-1} |c(i+m, j+n) - r(i+k+m, j+l+n)| \qquad (7.2) \end{aligned}$$

where $c(i + m, j + n)$ is the pixel value in the current block with coordinates (i, j), $r(i + k + m, j + l + n)$ is the pixel value in the search area in the reference frame, and (k, l) is the relative displacement between the current block and the reference block in the search area.

Equation (7.1) can be decomposed into progressive equations in hierarchical levels. Each level represents the operations to be performed. The goal is to achieve an efficient parallel hardware architecture for each level separately.

Fig. 7.4 shows the $(2P + 1)^2$ SAD values associated with a particular reference block. Each SAD value is obtained at different k, l relative shifts between the c and r blocks. The figure is based on $P = 3$ for simplicity and the dark circles indicate the minimum SAD value at each row.

Figure 7.5 presents a block diagram for the hierarchical decomposition of the full search motion estimation hardware. The functions of each hierarchical level is described in the following sections.

hierarchical Level 3 (left-most level) Referring to Figure 7.5, each block at this hierarchical level produces the sum value $D(i, j, k, l, m)$. Block $D(i, j, k, l, m)$ in the figure

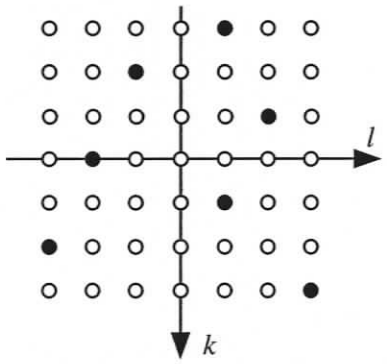


Figure 7.4. *$SAD(i, j, k, l)$ values for each reference block.*

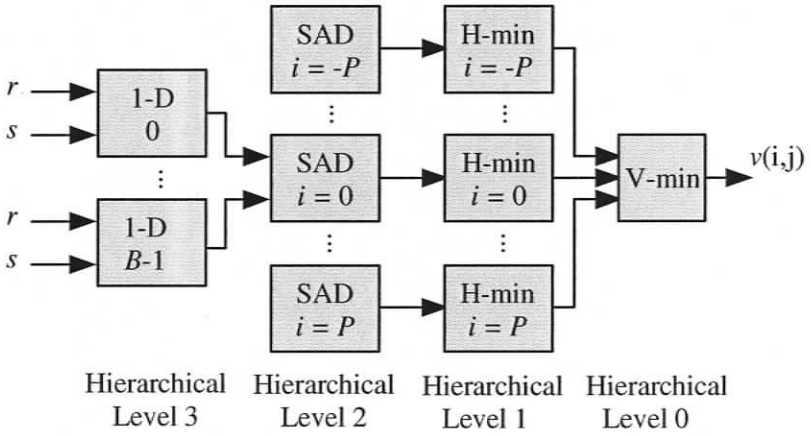


Figure 7.5. *A block diagram for the hierarchical decomposition of the FS algorithm.*

corresponds to the one-dimensional sum absolute difference between two rows of the c and r blocks.

The output of each D block is given by the expression

$$\mathrm{D}(i, j, k, l, m) = \sum_{n=0}^{B-1} |X(i+m, j+n) - Y(i+k+m, j+l+n)| \qquad (7.3)$$

where $n = 0,\ 1,\ 2,\ \cdots B-1$.

Hierarchical level 3 is the most important level since its hardware implementation will have the most impact on the timing and hardware resource requirements. The blocks at this

level of the hierarchy implement a one-dimensional sum absolute difference operation as described by (7.3). Using the mapping technique described in [100],the dependency graph (DG) of the above equation is shown in Figure 7.6 for $B = 3$, $P = 3$, and $W = 15$. The

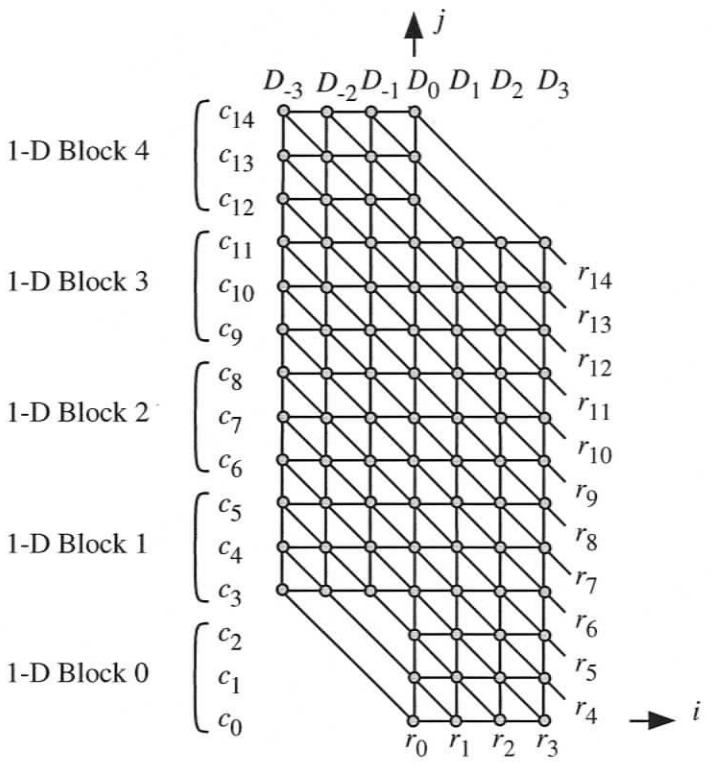


Figure 7.6. *DG for one-dimensional SAD computation.*

output variable $D(i)$ is represented by vertical lines so that each vertical line corresponds to a particular instance of D. The input variable c is represented by horizontal lines and the input variable r is represented by diagonal lines. Circles represent operations to be performed.

The resulting signal flow graph (SFG) is shown in Figure 7.7. The gray arrows indicate the directions of flow of pipelined data while empty circles represent partial results of the SAD computations and black circles represent valid outputs.

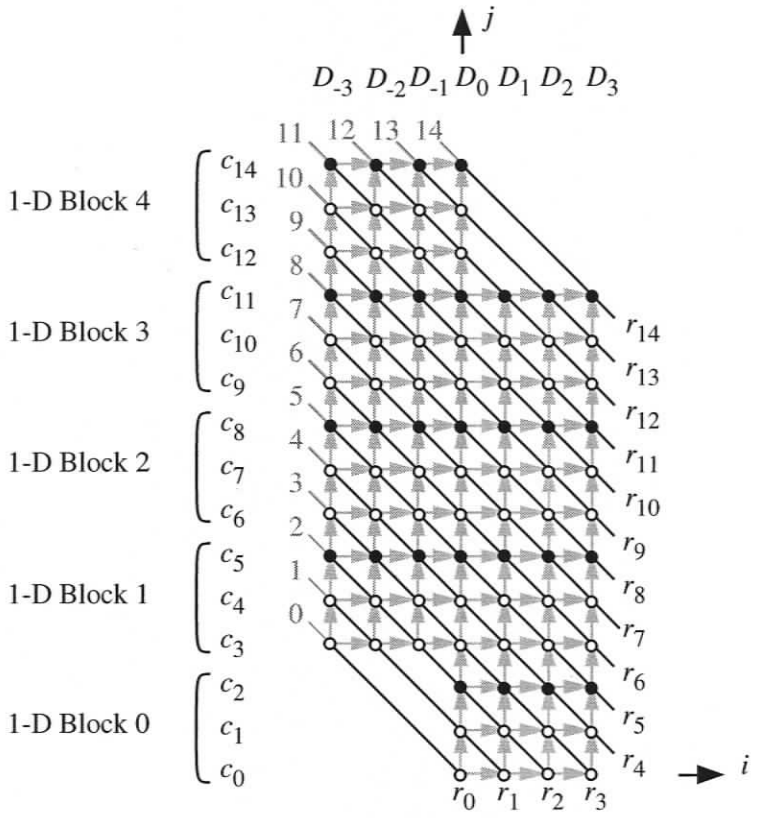


Figure 7.7. *SFG for one-dimensional SAD computation.*

hierarchical Level 2 Referring to Figure 7.5, each block at this hierarchical level produces a SAD value that is associated with a particular relative shift pair (k, l) which corresponds to one circle in Figure 7.4. Black $SAD(j)$ in the figure corresponds to the SAD value that results due to a relative horizontal shift of value j between c and r blocks.

The output of each *SAD* block can be written as

$$\mathrm{SAD}(i,j,k,l) = \sum_{m=0}^{B-1} \mathrm{D}(i,j,k,l,m) \tag{7.4}$$

where $\mathrm{D}(i,j,k,l,m)$ represents the outputs of the B blocks comprising hierarchical level 3.

hierarchical Level 1 Referring to Figure 7.5, each block at this hierarchical level produces

the minimum SAD value *H-min*(i, j, k) corresponding to one row in Figure 7.4. Block $H - min(i)$ in the figure corresponds to the H-min value that corresponds to a relative vertical shift of value i between r and s blocks. Each row has $(2P + 1)$ SAD values and the minimum value is indicated by the black circle.

The output of each *H-min* block can be written as

$$\text{H-min}(i, j, k) = \arg\min\left[\text{SAD}(i, j, k, l)\right] \tag{7.5}$$

where *arg min* is the function that selects the minimum of the $(2P+1)$ values and $SAD(i, j, k, l)$ represents the outputs of the $(2P + 1)$ blocks comprising hierarchy Level 2. The range of index l is given by

$$l = l_{min},\ l_{min} + 1,\ \cdots,\ l_{max}$$

where

$$\begin{aligned} l_{min} &= \max(0, j - P) \\ l_{max} &= \min(j + P, W - B) \end{aligned}$$

Hierarchical Level 0 Referring to Fig. 7.5, this hierarchical level produces the motion vector by selecting the minimum *SAD* value from among a set of minimum values *H-min*(i, j, k) that are indicated by the black circles in Figure 7.4.

The output of the *V-min* block corresponds to the output in (7.1). Using the notation of Figure 7.5 we can write the output as

$$\mathbf{v}(i, j) = \text{V-min}\left[\text{H-min}(i, j, k)\right] \tag{7.6}$$

where *V-min* is the function that selects the minimum of $2P + 1$ values and *H-min*(i, j, k) represent the outputs of the $2P + 1$ blocks comprising hierarchy Level 1. The ranges of the indices i, j, and k are

$$\begin{aligned} i &= 0,\ B,\ 2B, \cdots,\ (W/B - 1)B \\ j &= 0,\ B,\ 2B, \cdots,\ (H/B - 1)B \\ k &= k_{min},\ k_{min} + 1,\ \cdots,\ k_{max} \end{aligned}$$

where

$$k_{min} = \max(0, j - P)$$
$$k_{max} = \min(j + P, H - B)$$

7.3.2 Hardware Design

The proposed hardware design for each hierarchical level is described in this section.

Hierarchical Level 3

Based on the SFG in Figure 7.7, a proposed hardware design for level 3 is shown in Figure 7.8. This hardware uses $2P + 1$ processing elements (PEs) in order to achieve maximum speed and utilization performance. The design of each PE is shown in Figure 7.9.

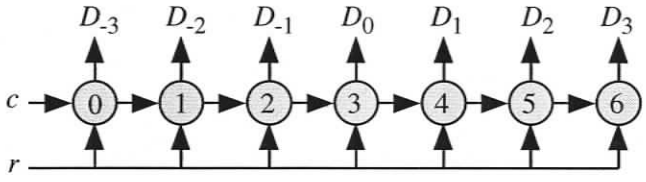


Figure 7.8. *The resulting systolic array for implementing one-dimensional SAD calculation when $B = P = 3$ and $W = 15$.*

Hierarchical Level 2 The blocks at this hierarchical level implement a one-dimensional sum operation as described by (7.4). The hardware implementation of such a block is a simple accumulator as shown in Figure 7.10.

Given the design parameters $B = P = 3$, it was observed that there are 2 parallel D values that can be produced at the same clock cycle. As a result, two parallel H-Min units can be used. For general values of B and P, the number of parallel H-Min units is given by $(2P + 1)/B$.

hierarchical Level 1

This level is simply a comparator and minimum value storage unit. Parallel H-min units can be used in order to speed up the computation. The number of these parallel units

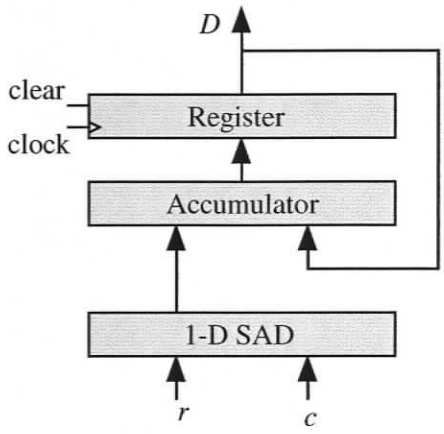


Figure 7.9. *Details of a PE for the systolic array in Fig. 7.8.*

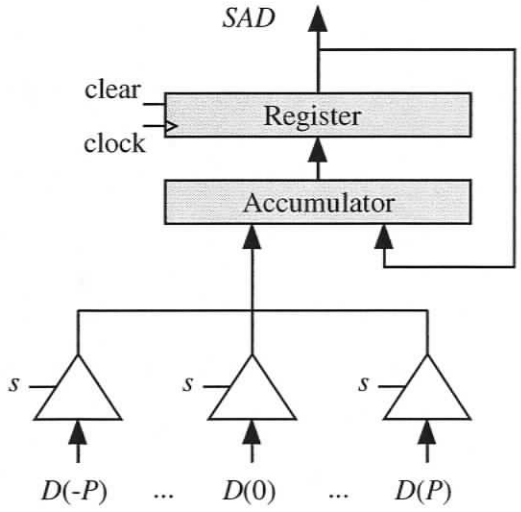


Figure 7.10. *Hardware implementation of hierarchical level 2.*

is $(2P+1)/B$.

Hierarchical Level 0

Hardware Design for level 0 is similar to that of level 1, H-Min, with the exception that H-Min receives only one input at a time and stores the minimum value while the V-min unit receives $2P+1$ parallel values as shown in Figure 7.12. In addition, V-min does not

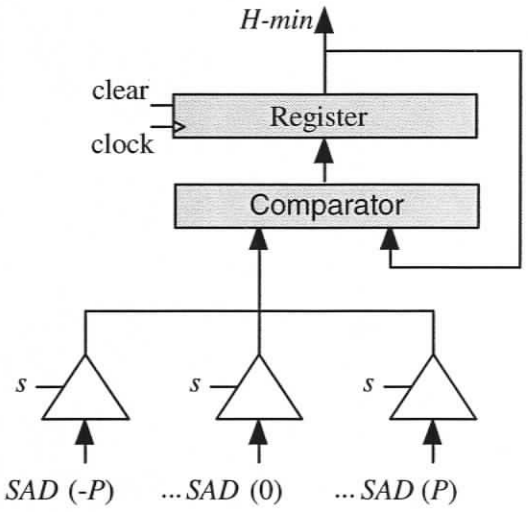


Figure 7.11. *Hardware Design for Hierarchical Level 1.*

need to accumulate the minimum value. In order to speed up the clock cycle at this stage, a pipelined V-min tree is used for comparison as shown in Figure 7.13 using $P = 3$.

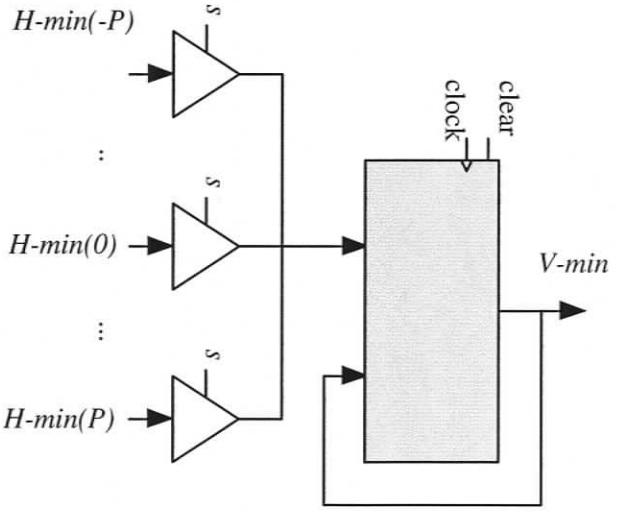


Figure 7.12. *Hardware Design for Hierarchical Level 0.*

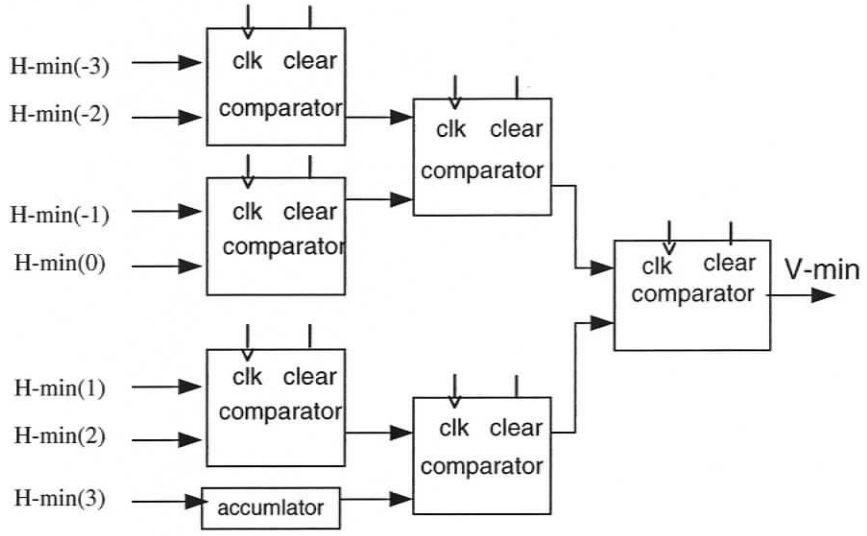


Figure 7.13. *V-min Tree Design for Hierarchical Level 0 (P=3).*

7.3.3 Simulation Results

The proposed design was modeled, simulated, and verified in VHDL and synthesized using Xilinx ISE 7.1. The device used for synthesizing was Xilinx Spartan 3 FPGA. Table 7.5 lists the hardware specifications for different macroblock sizes (B) produced by the synthesizer. The table also includes the maximum supported frame rate CIF resolution. From the table, it can be noticed that the proposed implementation achieves high frame rates for the different macroblock sizes, while using a low CLB count.

Table 7.6 compares the proposed implementation and the implementations presented in [93], [94] for macroblock size = 8. These designs were selected for comparison as they target similar FPGA implementations. The authors in [94] present two different architectures named AB2 and AS2.

Results in Table 7.6 indicate that the proposed hardware supports a higher frame rate as compared to the other two designs. At the same time, the used CLB count is kept as low as possible. This is attributed to the efficient array architecture that fully utilizes the available

Table 7.5. *The FS hardware specifications for different block sizes.*

Parameter	**Result**		
Macroblock size	**B=4**	**B=8**	**B=16**
Used CLBs (Out of 8320)	991	1712	5379
FPGA utilization %	12	21	65
Maximum frequency (MHz)	60	70	60
Clock cycle per macroblock	32	129	514
Maximum CIF frame rate (frame/sec.)	294	344	295

processors.

Table 7.6. *Comparison between proposed architecture and other designs for macroblock size = 8.*

Design	**[93]**	**AB2 [94]**	**AS2 [94]**	**Proposed**
Used CLBs	939	948	3732	**1712**
Maximum frequency (MHz)	110	30	22	**70**
Clock cycle per macroblock	2042	379	190	**129**
Maximum CIF frame rate (frame/sec.)	34	50	73	**344**

7.4 Comparison of the *FTS* and *FS* Implementations

This chapter presented two hardware implementations for the *FTS* and the *FS* algorithms. The *FS* is very regular and has simple control. On the other hand, the *FTS* design includes more complex logic in return for a lower number of cycles per macroblock.

Results showed that the *FS* implementation is more suitable for small block sizes while the *FTS* implementation is more suitable for bigger block sizes. This is attributed to the fact that the *FTS* design requires additional cycles for data loading and control logic but

at the same time requires less SAD computations. For large block sizes, the number of cycles required by the *FTS* data loading and control logic becomes less significant than the number of cycles required for SAD computations which makes the *FTS* hardware more efficient than the *FS* hardware.

The *FTS* requires more hardware than the *FS* due to the caching of the current block and the search area. The *FS* accesses the memory in sequential order and thus does not need to include memory caches.

The above comparison demonstrates the trade off between algorithm regularity and computation efficiency.

7.5 Conclusions

Hardware designs for two block-matching algorithms, namely, the *FTS* and the *FS* were presented. The *FS* implementation requires less hardware units and can achieve a maximum frame rate of 344 at CIF resolution. The *FTS* implementation achieves higher frame rate especially at a block size 16. However, it requires additional memory for caching data. Design details and simulation results for each implementation were presented. The differences between the two hardware implementations were discussed. The results of the proposed architectures have been published in [101, 102].

Chapter 8

Conclusions and Future Work

8.1 Contributions of the thesis

The goal of this thesis was to develop more computationally efficient motion estimation algorithms in both hardware and software while maintaining other important encoding factors such as compression ratio and quality of the decompressed videos. The performance of the developed algorithms was analyzed and compared with that of other state-of-the-art algorithms. The suitability of the proposed FTS technique for hardware implementation using FPGAs was also investigated.

8.1.1 Development of the *WSBM* Algorithm

The *WSBM* algorithm described in section 3.4 was used to predict the origin of the search window for a certain macroblock based on the motion search results from surrounding macroblocks. If the predicted origin yields a lower error than the original search origin, then the predicted origin is chosen as the new search origin. In addition, the search window size is reduced by $1/2$. If the predicted origin yields a higher error than the original origin, then the search continues with the original origin without changing the search window size. The *WSBM* algorithm was compared with another adaptive search technique. In addition, the *WSBM* was added to two search techniques and results indicate that the combined techniques produced better results than the original ones.

8.1.2 Development of the *SMPLX* Algorithm

The second contribution of the thesis was to implement the simplex optimization method as a block-matching algorithm. The simplex is a non-constraint optimization method that works for any n-dimensional problem. The proposed *SMPLX* algorithm uses a search triangle to locate the minimum error. The algorithm uses different operations during the search such as reflection, expansion, contraction, and reduction. These operations provide the *SMPLX* algorithm with high flexibility and make it more able to avoid local minima. Using these operations, the search triangle can easily change its direction and/or search step. Simulation results indicate that the *SMPLX* algorithm is very efficient in terms of reducing the number of search positions to be checked and thus reducing the amount of computation required by the search.

8.1.3 Development of the *FTS* Algorithm

The third contribution of the thesis entails the development of the *FTS* algorithm, which eliminated some of the limitations of the *SMPLX* algorithm. The *FTS* uses integer calculations and look-up tables. The *FTS* was compared with some of the state-of-the-art block-matching algorithm and was shown to have a superior performance.

8.1.4 Extensions of the *FTS* algorithm

The fourth contribution entails some additional extensions to the *FTS* algorithm such as the *EFTS*, *HP-FTS*, and *PFTS* as follows:

- In the *EFTS*, the exit condition was modified to avoid pre-mature exit of the *FTS* and a buffering mechanism for the computed SAD values was added in order to avoid repeated SAD computations.
- In the *HP-FTS*, full-pixel and half-pixel searches were incorporated. In addition, the starting level of the *FTS* search was modified based on prediction from neighboring

macroblocks. The *HP-FTS* was shown to have good performance compared to other half-pixel approximations techniques. *HP-FTS* has reduced the combined computational complexity by an average of 15%.

- The *PFTS* uses prediction techniques to select the direction of the initial search triangle. Using this approach, the *PFTS* can direct the search triangle in the correct direction from the beginning and thus reduce the computational effort required by the *FTS* by an average of 7% to 13%.

8.1.5 Hardware Implementation of the *FTS* Algorithm

The fifth contribution of the thesis was a hardware implementation for the *FTS* algorithm. The proposed implementation uses a systolic array design for SAD computations. In addition, the implementation uses an algorithm state machine to match the *FTS* control logic as well as caching buffer to reduce access to external memory. Simulation results showed that the proposed implementation requires a smaller number of cycles per macroblock search compared to other hardware implementations and thus can provide motion estimation at high frame rates.

8.1.6 Hardware Implementation of the *FS* Algorithm

The sixth contribution of the thesis was the development of an efficient hierarchical hardware implementation for the *FS* algorithm. The proposed implementation uses systolic arrays to perform motion search in parallel and pipelining stages with very efficient processor utilization. In addition, a hierarchical decomposition technique is applied to compute the 2-D SAD using successive 1-D hardware units. As a result, the proposed implementation offers high frame rates using a smaller number of hardware units as demonstrated by the simulation results.

8.2 Future Work

Some additional research can be undertaken to further improve the *FTS* algorithm and adapt it for different search applications. This section describes some of these potential research directions.

8.2.1 Predictive *FTS* algorithm

Additional features can be added to the *FTS* algorithm to further improve its overall performance as follows:

- Prediction of the starting level which can improve the search steps of the *FTS* algorithm.
- Prediction of the scene complexity which would lead to a better choice of initial search parameters. This can be done by observing the movements of the search triangle in the *FTS* algorithm.
- Modify the $x-y$ ratio of the search triangles at each level based on sequence analysis. In the current *FTS* algorithm, there are three levels with fixed $x - y$ ratios. These ratios can be changed based on the scene contents. For example, the triangles can be stretched in x or y direction or both.
- Detection of situations where the *FTS* can get stuck and revisit the same positions. In this case, an early exit can reduce the computational effort required by the *FTS* algorithm.

8.2.2 Extension of the *FTS* to sub-pixel search

The *HP-FTS* has shown promising results. A possible additional extension to this algorithm is to develop a fully integrated full-pixel, half-pixel, and quarter-pixel *FTS*. In addition, some half-pixel and quater-pixel approximations can be used to reduce the need for interpolation.

8.2.3 Multi-block size *FTS*

The *FTS* algorithm is currently working on one-size macroblock at a time. Another improvement is to add additional logic to support changing the macroblock size during the search and thus producing an integrated solution that combines the best motion vector and the best macroblock mode. This integrated solution could be very useful for H.264 encoding which supports several macroblock modes and requires a substantial amount of computations in order to identify the best mode.

8.2.4 *FTS* hardware Implementation

The hardware implementation of the *FTS* can be further improved to increase its performance. Examples of possible improvements are as follows:

- Adding a SAD buffer to avoid repeated computations as described by the *EFTS* algorithm in section 6.2.
- Adding early termination of SAD computations when the newly obtained value exceeds the current minimum.
- Using additional features in hardware such as pre-calculation of next states and addresses while the current state computations are being carried out. This will further speed up the performance of the hardware.
- More parallelism and pipelining can be added to speed up the performance of the *FTS* units that lie on the critical path.

Bibliography

[1] M. Nelson, *The Data Compression Book.* M&T publishing, 1992.

[2] *Information technology - digital compression and coding of continuous-tone still images: Requirements and guidelines*, ISO/IEC Std. 10 918-1, 1990.

[3] G. K. Wallace, "The JPEG still picture compression standard," *Communication of the ACM*, vol. 34, no. 4, pp. 31–44, Apr. 1991.

[4] *Information Technology- JPEG 2000 Image Coding System- Part I: Core Coding System*, ISO/IEC Std. 15 444-1, 2000.

[5] *Information Technology- JPEG 2000 Image Coding System- Part II: Extensions*, ISO/IEC Std. 15 444-2, 2000.

[6] C. Christopoulos, A. A. Skodras, and T. Ebrahimi, "The JPEG 2000 still image coding system: An overview," *IEEE Transactions on Consumer Electronics*, vol. 46, no. 4, pp. 1103–1127, Nov. 2000.

[7] Y. Wang, J. Ostermann, and Y. Zhang, *Video Processing and Communications.* Prentice Hall, Signal Processing Series, 2002.

[8] C. Poynton, *Digital Video and HDTV Algorithms and Interfaces.* Morgan Kaufmann Publishers, 2002.

[9] P. Symes, *Video Compression Demystified.* McGraw-Hill, 2001.

[10] V. Bhaskaran and K. Konstantinides, *Image and Video Compression Standards Algorithms and Architectures.* Boston: Kluwer Academic Publishers, Sept. 1995.

[11] P. Kuhn, *Algorithms, Complexity Analysis, and VLSI Architectures for MPEG-4 Motion Estimation.* Boston: Kluwer Academic Publishers, 1999.

[12] W. K. Pratt, *Image Transmission Methods.* N.Y.: Academic Press, 1979.

[13] A. S. Lewis and G. Knowles, "Video compression using 3D wavelet transform," *Electronics Letters*, vol. 26, no. 6, pp. 174–0177, Mar. 1990.

[14] M. S. Lazar and L. T. Bruton, "Fractal block coding of digital video," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 4, no. 3, pp. 297–308, June 1994.

[15] L. Torres and M. Kunt, *Video Coding the Second Generation Approach.* Boston: Kluwer Academic Publishers, 1996.

[16] J. E. Fowler, "Real-time video compression using differential vector quantization," *IEEE Transactions on Circuits and Systems for Video Technology*.

[17] K. S. Thyagarajan and D. Ericksont, "Variable rate self organization neural networks for video compression," *IEEE Computing Society Press*, vol. 1, pp. 244–248, 1994.

[18] *Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbits/s*, ISO/IEC Std. 11 172, 1992.

[19] D. L. Gall, "MPEG: A video compression standard for multimedia applications," *Communication of the ACM*, vol. 34, Apr. 1991.

[20] G. Morrison, "Video coding standards for multimedia: JPEG, H.261, MPEG," in *Proceedings of IEE Colloquium on Technology Support of Multimedia*, no. 88, Apr. 1992, pp. 2.1–2.4.

[21] D. L. Gall, "The MPEG video compression algorithm," *Signal Processing: Image Communication*, vol. 4, pp. 129–140, 1992.

[22] *Generic Coding of Moving Pictures and Associated Audio*, ISO/IEC Std. 13 818, 1995.

[23] A. G. Macinnis, "MPEG systems coding specification," *Signal Processing: Image Communication*, vol. 4, pp. 153–159, 1992.

[24] L. J. Nelson, "MPEG-1, MPEG-2 & You: Light in the middle of the tunnel," *Advanced Imaging*, vol. 9, no. 11, pp. 28–31, Nov. 1994.

[25] O. Avaro, A. Eleftheriadis, C. Herpel, G. Rajan, and L. Ward, "MPEG-4 systems: Overview." in *Multimedia Systems, Standards, and Networks*, A. Puri and T. Chen, Eds. New York: Marcel Dekker, 2000, pp. 331–65.

[26] *Coding of Audio-Visual Objects*, ISO/IEC Std. 14 496, 1999.

[27] *Recommendation H.261: Video codecs for Audiovisual services at* $p \times 64$ *kbits*, ITU-T Std., 1993.

[28] *Recommendation H.262: Generic Coding of Moving Pictures and Associated Audio Information*, ITU-T Std., 1995.

[29] *Recommendation H.263: Video Coding for Low Bit Rate Communications*, ITU-T Std., 1998.

[30] T. Chen, G. J. Sullivan, and A. Puri, "H.263 including H.263++ and other ITU-T video coding standards," in *Multimedia Systems, Standards, and Networks*, A. Puri and T. Chen, Eds. New York: Marcel Dekker, 2000, pp. 55–85.

[31] *Recommendation H.264: Advanced Video Coding for Generic Audio Visual Services*, ITU-T Std., 2003.

[32] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 560–576, July 2003.

[33] G. Raja and M. J. Mirza, "Performance comparison of advanced video coding H.264 standard with baseline H.263 and H.263+ standards," in *Proceedings of IEEE International Symposium on Communications and Information Technology, ISCIT*, vol. 2, Oct. 2004, pp. 743–746.

[34] N. Kamaci and Y. Altunbasak, "Performance comparison of the emerging H.264 video coding standard with the existing standards," in *Proceedings of International Conference on Multimedia and Expo, ICME'03*, vol. 1, July 2003, pp. 345–8.

[35] D. Himmelblau, *Applied Nonlinear Programming*. New York: McGraw-Hill, 1972.

[36] M. E. El-Mualla, C. N. Canagarajah, and D. Bull, *Video Coding for Mobile Communications: Efficiency, Complexity, and Resilience*. Academic Press, 2002.

[37] B. Bunday, *Basic Optimization Methods*. Edward Arnold Publishers, 1984.

[38] F. Pan, X. Lin, S. Rahardja, K. P. Lim, Z. G. Li, G. N. Feng, D. J. Wu, and S. Wu, "Fast mode decision algorithms for jvt intra prediction," *7th JVT Meeting JVT-G013*, Mar. 2003.

[39] K. P. Lim, S. Wu, J. Wu, S. Rahardja, X. Lin, F. Pan, and Z. G. Li, "Fast intermode decision," *9th JVT Meeting JVT-1020*, Sept. 2003.

[40] T. Cheng and Y.-Q. Zhang, "A new rate control scheme using quadratic rate distortion model," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, no. 1, pp. 246–250, Feb. 1997.

[41] A. Ortega and K. Ramchandran, "Rate-distortion methods for image and video compression," *IEEE Signal Processing Magazine*, vol. 15, no. 6, pp. 23–50, Sept. 1998.

[42] G. J. Sullivan and T. Wiegand, "Rate-distortion optimization for video compression," *IEEE Signal Processing Magazine*, vol. 15, no. 6, pp. 74–90, Sept. 1998.

[43] *Video Codec Test Model, near Term, Version 10, Draft I*, Std.

[44] H. S. D. Marpe and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264 video compression standard," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 620–636, July 2003.

[45] H. G. Musmann, P. Pirsch, and H. J. Grallert, "Advances in picture coding," in *Proc. IEEE*, vol. 73, no. 4, Apr. 1985, pp. 523–548.

[46] T. R. Hsing, "Motion detection and compensation coding for motion video coders: Technical review and comparison," in *Proceedings of IEEE Global Telecommunication Conf.*, 1987, pp. 60–64.

[47] T. G. Ahn, Y. H. Moon, and J. H. K., "Fast full-search motion estimation based on multilevel successive elimination algorithm," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 11, pp. 1265 – 1269, Nov. 2004.

[48] J. Kim and T. Choi, "A fast full-search motion-estimation algorithm using representative pixels and adaptive matching scan," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, no. 7, pp. 1040 – 1048, Oct. 2000.

[49] J. Kim, S. Byun, Y. Kim, and B. Ahn, "Fast full search motion estimation algorithm using early detection of impossible candidate vectors," *IEEE Transactions on Signal Processing*, vol. 50, no. 9, pp. 2355–2365, 2002.

[50] B.Song, K. Chun, and J. Ra, "A rate-constrained fast full-search algorithm based on block sum pyramid," *IEEE Transactions on Image Processing,*, vol. 14, no. 3, pp. 308–311, Mar. 2005.

[51] F. A. Kamangar and K. R. Rao, "Interfield hybrid image coding of component color television signals," *IEEE Transactions on Communications*, vol. 29, no. 12, pp. 1740–1753, Dec. 1981.

[52] J. R. Jain and A. K. Jain, "Displacement measurement and its application in interframe image coding," *IEEE Transactions on Communication*, vol. 29, no. 12, Dec. 1981.

[53] M. Ghanbari, "The cross-search algorithm for motion estimation," *IEEE Transactions on Communication*, vol. 38, no. 7, July 1990.

[54] T. Koga, K. Linuma, A. Hirano, Y. Lijima, and T. Ishiguro, "Motion compensated interframe coding for video conferencing," *Proc. NTC 81*, pp. c9.6.1–9.6.5, Nov. 1981.

[55] R. Li, B. Zeng, and M. L. Liou, "A new three step search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 4, no. 4, Aug. 1994.

[56] C. J. Duanmu, M. O. Ahmad, and M. N. S. Swamy, "A fast three-step search algorithm by the utilization of multilevel vector partial sums," *Proceedings of Canadian Conference on Electrical and Computer Engineering. IEEE CCECE*, vol. 3, May 2003.

[57] L. Po and W. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6,

no. 3, pp. 313–317, June 1996.

[58] B. B. Paul and E. Viscito, "Hierarchical motion estimation with 2-scale tilings," in *Proceedings of IEEE International Conference on Image Processing*, 1994, pp. 260–264.

[59] C. H. Cheung and L. M. Po, "A novel cross-diamond search algorithm for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 12, pp. 1168–1177, Dec. 2002.

[60] S. Zhu and K. k. Ma, "A new diamond search algorithm for fast block-matching motion estimation," *IEEE Transactions Image Processing,*, vol. 9, pp. 287–290, 2000.

[61] J. Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, "A novel unrestricted center-biased diamond search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, pp. 369–377, Aug. 1998.

[62] A. M. Tourapis, O. C. Au, M. L. Liou, and G. Shen, "Fast and efficient motion estimation using diamond zonal based algorithms," *J. Circuits, Systems and Signal Processing,*, vol. 20, no. 2, pp. 233–251, June 2001.

[63] A. M. Tourapis, O. C. Au, and M. L. Liou, "Highly efficient predictive zonal algorithms for fast block-matching motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 10, pp. 934–947, 2002.

[64] C. Zhu, X. Lin, and L. P. Chau, "Hexagon-based search pattern for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 5, pp. 349–355, 2002.

[65] K. Lo and J. Feng, "Multiresolution block matching algorithm for motion estimation in video coding," in *Conference Proceedings. Singapore ICCS*, Nov. 1994, pp. 1117–1120.

[66] J. Lee, K. W. Lim, B. Song, and J. Ra, "A fast multi-resolution block matching algorithm and its LSI architecture for low bit-rate video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 12, pp. 1289–1301, Dec. 2001.

[67] X. Lee and Y. Zhan, "A fast hierarchical motion-compensation scheme for video coding using block feature matching," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 6, pp. 627–635, Dec. 1996.

[68] S. Cucchi and D. Grechi, "A new features-based fast algorithm for motion estimation: decimated integral projection (dip)," in *Proceedings of International Conference on Information, Communications and Signal Processing, ICICS*, vol. 1, Sept.

1997, pp. 297–300.

[69] G. Sorwar, "Adaptive-centre candidate decimation distance-dependent thresholding search for motion estimation," in *Proceedings of Third International Conference on Information Technology and Applications, ICITA*, vol. 1, July 2005, pp. 674–679.

[70] C. Wu, C. Yao, B. Liu, and J. Yang, "DCT-based adaptive thresholding algorithm for binary motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 5, pp. 694–703, May 2005.

[71] R. Srinivasan and K. R. Rao, "Predictive coding based on efficient motion estimation," in *Proceedings of IEEE International Conference on Communications*, Amsterdam, pp. 521–526.

[72] P. I. Hosur and K. K. Ma, "Motion vector field adaptive fast motion estimation," in *Proceedings of Second International Conference on Information, Communications, and Signal Processing (ICICS'99)*, Singapore, Dec. 1999.

[73] A. M. Tourapis, O. C. Au, and M. L. Liou, "Predictive motion vector field adaptive search technique (PMVFAST)-enhancing block based motion estimation," in *Proceedings of Visual Communications and Image Processing (VCIP'01)*, 2001.

[74] J. S. Chana, "Development and analysis of block-based motion estimation techniques for efficient video compression," Master's thesis, Department of Electrical Engineering and Computer Engineering, University of Victoria, Canada, BC, CA, 1994.

[75] S. Zafar, Y. Zhang, and J. Baras, "Predictive block matching motion estimation for T.V. coding– Part I: Inter-block prediction," *IEEE Transactions on Broadcasting*, vol. 37, no. 3, pp. 102–105, Sept. 1991.

[76] S. Zafar, Y. Zhang, and J. S. Baras, "Predictive block matching motion estimation for T.V. coding– Part II: Inter-frame prediction," *IEEE Transactions on Broadcasting*, vol. 37, no. 3, pp. 102–105, Sept. 1991.

[77] K. linuma, "A motion-compensated interframe codec," *SPIE: Image Coding*, vol. 594, pp. 194–201, Dec. 1985.

[78] J. S. Chana and P. Agathoklis, "Adaptive motion estimation for efficient video compression," *Proceedings of 29th Asilomar Conference on Signals, Systems & Computers*, vol. 1, pp. 690–693, Nov. 1995.

[79] M. Rehan, P. Agathoklis, and A. Antoniou, "A new motion-estimation technique for efficient video compression," in *Proceedings of IEEE Pacific Rim Conference on Communications, Computers and Signal processing (PACRIM'97)*, Aug. 1997, pp. 326–329.

[80] M. Rehan, A. Antoniou, and P. Agathoklis, "A new fast block matching algorithm using the simplex technique," in *Proceedings of IEEE Symposium on Advances in Digital Filtering and Signal processing (DFSP'98)*, June 1998, pp. 30–33.

[81] M. E. Al-Mualla, C. N. Canagarajah, and D. Bull, "A simplex minimization for single and multiple- reference motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 12, pp. 1209–1220, Dec. 2001.

[82] M. E. Al-Mualla, C. N. Canagarajah, , and D. Bull, "Simplex minimisation for multiple-reference motion estimation," *Proceedings of The 2000 IEEE International Symposium on Circuits and Systems, ISCAS 2000 Geneva*, vol. 4, pp. 733–736, 2000.

[83] M. E. Al-Mualla, C. N. Canagarajah, , and D. R. Bull, "Simplex minimisation for fast long-term memory motion estimation," *Electronics Letters*, vol. 37, no. 5, pp. 290–292, 2001.

[84] ——, "Simplex minimisation for fast block matching motion estimation," *Electronics Letters*, vol. 34, no. 4, pp. 351–352, 1998.

[85] M. Rehan, P. Agathoklis, and A. Antoniou, "Flexible triangle search for block based motion estimation," in *Proceedings of IEEE Pacific Rim Conference on Communications, Computers and Signal processing (PACRIM'03)*, Aug. 2003, pp. 233–236.

[86] ——, "Flexible triangle search for block based motion estimation," *Journal of Applied Signal Processing*, 2006.

[87] ——, "Flexible polygon motion estimating method and system," *Patent number 11/212,486 U.S. and CA2477625 Canada (submitted)*, 2004.

[88] Joint Video Team Reference Software. (2003) H.264 Version 9.2 (JM 9.2). [Online]. Available: http://iphome.hhi.de/suehring/tml/download/

[89] D. Cheng, H. Yun, and Z. Junli, "PPHPS: A parabolic prediction-based, fast half-pixel search algorithm for very low bit-rate moving-picture coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 6, pp. 514–518, 2003.

[90] M. Rehan and P. Agathoklis, "Half-pixel accurate motion estimation using a flexible triangle search," in *Proceedings of IEEE Pacific Rim Conference on Communications, Computers and Signal processing (PACRIM'05)*, Aug. 2005, pp. 233–236.

[91] ——, "Block-based motion estimation using an enhanced flexible triangle search algorithm," in *Proceedings of Canadian Conference on Electrical and Computer Engineering (CCECE05)*, May 2005, pp. 259–262.

[92] ——, "Prediction-based flexible triangle search algorithm," in *Proceedings of Cana-*

dian Conference on Electrical and Computer Engineering (CCECE06), May 2006, pp. 2067–2070.

[93] M. Mohammadzadeh, M. Eshghi, and M. Azdfar, "An optimized systolic array architecture for full-search block matching algorithm and its-implementation on FPGA chips," in *Proceedings of the Third International IEEE-NEWCAS Conference*, 2005, pp. 174–0177.

[94] A. Ryszko and K. Wiatr, "Motion estimation operation implemented in FPGA chips for real-time image compression," in *Proceedings of the Second International Symposium on Image and Signal Processing and Analysis ISPA*, 2001, pp. 399–404.

[95] C. Wang, S. Yang, C. Liu, and T. Chiang, "A hierarchical N-Queen decimation lattice and hardware architecture for motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 4, pp. 429–449, 2004.

[96] S. Jung and S. Lee, "A 4-way pipelined processing architecture for three-step search block-matching motion estimation," *IEEE Transactions on Consumer Electronics*, vol. 50, no. 2, pp. 674–681, May 2004.

[97] K. Seth, P. Rangarajan, S. Srinivasan, V. Kamakoti, and V. B. Kuteshwar, "A parallel architectural implementation of the new three-step search algorithm for block motion estimation," in *Proceedings of the 17th International Conference on VLSI Design*, 2004, pp. 1071–1076.

[98] M. Sarma, D. Samanta, and A. Sundar, "VLSI architecture for multi-resolution three step search algorithm," in *Proceedings of the 5th International Conference on ASIC*, vol. 2, 2003, pp. 918–921.

[99] W. Chao, C. Hsu, Y. Chang, and L. Chen, "A novel hybrid motion estimator supporting diamond search and fast full search," in *Proceedings of IEEE International Symposium on Circuits and Systems, ISCAS*, vol. 2, 2002, pp. 492–495.

[100] F. Elguibaly and A. Tawfik, "Mapping 3-D IIR digital filters onto systolic arrays," *Multidimensional Signal Processing*, vol. 7, pp. 7–26, 1996.

[101] M. Rehan, P. Agathoklis, M. Watheq, and F. Gebali, "FPGA design of flexible triangle search algorithm for motion estimation," *Proceedings of IEEE International Symposium on Circuits and Systems, ISCAS 2006*, pp. 521–524, 2006.

[102] M. Rehan, M. Watheq, and F. Gebali, "Hierarchal full search motion estimation algorithm," *Springer Jounral of Multimedia*, 2006.

[1] All online references have been visited on June 29, 2006.