Fast motion estimation for video coding

Qi, Weisong

2005

Qi, W. (2005). Fast motion estimation for video coding. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/3158>

<https://doi.org/10.32657/10356/3158>

Nanyang Technological University

Downloaded on 26 Apr 2023 21:09:42 SGT

a357842

FAST MOTION ESTIMATION FOR VIDEO CODING

QI WEISONG

School of Electrical & Electronic Engineering

A thesis submitted to Nanyang Technological University
in fulfillment of the requirement for the degree of
Master of Engineering

2005

TK
6680.5
Q1
2005

Acknowledgement

First and foremost, I would like to express my earnest appreciation from the bottom of my heart to my supervisor, Dr. Zhu Ce, for his professional guidance, invaluable support and kindly encouragement during my master study as well as his constructive comments on this thesis.

It is my pleasure to spend around two years with many talent and kind friends in the Media Lab. I cherish the friendship and thank them for their selfless help.

This thesis is dedicated to my parents. Their love and inspiration are always with me.

Table of Contents

Acknowledgement	i
Summary	vi
List of Figures	iii
List of Tables	iv
1 Introduction	1
1.1 Motion Estimation	1
1.2 Motivation and Objective	3
1.3 Outline and Contributions	4
2 Fast Block Matching Algorithms	
-Literature Review	6
2.1 Introduction	6
2.2 Matching Criterion	6
2.3 Full Search Algorithm (FSA)	7

<i>TABLE OF CONTENTS</i>	iii
2.4 Fast Motion Estimation	7
2.4.1 Fast Search Based on Uni-Modal Error Surface Assumption . . .	8
2.4.2 Fast BMAs Using Temporal/Spatial Correlations	11
2.4.3 Hierarchical/Mutiresolution Algorithms	16
2.4.4 Algorithms Using Pixel Decimation/Subsampling Techniques . .	19
2.4.5 Partial Distortion Search Algorithms	20
2.4.6 Inequality-based Fast Matching Algorithms Using Sum Norms .	25
2.5 Conclusion	28
3 Predictive Fine Granularity Successive Elimination for Fast Optimal Block Matching Motion Estimation (FGSE)	29
3.1 Introduction	29
3.2 Review Relevant Successive Elimination Algorithms	31
3.2.1 Successive Elimination Algorithm (SEA)	31
3.2.2 Multilevel Successive Elimination Algorithm (MSEA)	32
3.3 Proposed Fine Granularity Successive Elimination Algorithm	35
3.3.1 Limitations of MSEA	35
3.3.2 Proposed FGSE Framework	36
3.3.3 Partition Order Based on Image Complexity	41
3.3.4 Predictive FGSE Utilizing Correlation of Matching Errors	43
3.3.5 Procedure of FGSE Algorithm	44
3.4 Experimental Results	45

TABLE OF CONTENTS	iv
3.5 Conclusion	50
4 Multilevel Adaptive Refining Search (MARS): A New Framework to Integrate Fast Matching and Fast Search for Fast Motion Estimation	52
4.1 Introduction	52
4.2 In-depth Perspectives on the Combination of Fast Search and Fast Matching	54
4.3 Multilevel Global Elimination Algorithm	56
4.3.1 Global Elimination Algorithm [126]	56
4.3.2 Multilevel GEA	58
4.4 Multilevel Adaptive Refining Search (MARS)	59
4.4.1 Proposed MARS Framework	60
4.4.2 Readjusting Initial Search Range in MARS	65
4.4.3 Comparative Analysis	68
4.4.4 Procedure of MARS Algorithm	69
4.5 Experimental Results	70
4.6 Conclusion	74
5 Conclusions and Future Work	75
5.1 Conclusions	75
5.2 Future Work	77
5.2.1 Variable Block Size Motion Estimation	77
5.2.2 Multiframe Motion Estimation	78

TABLE OF CONTENTS	v
5.2.3 Subpixel Motion Estimation	78
5.2.4 Rate-Distortion-Complexity Fast Motion Estimation	79
Publications	80
Bibliography	81

Summary

Without compression, the transmission and storage of video signals are unpractical and unreliable. In consideration of simplicity and regularity, motion-compensated transform coding scheme is widely adopted by all international video coding standards such as H.261/263/264 and MPEG-1/2/4 to exploit the temporal redundancy between successive video frames. Due to the lower complexity in hardware implementation, block matching method is usually used for practical motion estimation. A large number of fast block matching algorithms (BMAs) have been proposed to reduce the intensive computations of motion estimation. Fast BMAs generally achieve a high speedup gain either by reducing the number of candidates (Fast Search) or by lowering the matching computations at each checking point (Fast Matching). Along each direction, many well-known algorithms have been developed applying various techniques. In this project, it's our interest and task to develop some more efficient and faster BMAs while maintaining high prediction accuracy.

In **Chapter 3**, a new framework, fine granularity successive elimination (FGSE), is proposed for fast optimal block matching in motion estimation. The FGSE features providing a sequence of nondecreasing fine-grained boundary levels to reject a checking point using as less computation as possible, where image complexity is utilized to determine the order of partitioning larger subblocks into smaller subblocks in the creation of the fine-grained boundary levels. Moreover, in view that two adjacent checking points (blocks) share most of the block pixels with just one pixel shifting horizontally or vertically, we develop a scheme to predict the rejection level for a candidate by ex-

exploiting the correlation of matching errors between two adjacent checking points. The resulting predictive FGSE algorithm can further reduce computation load by skipping some redundant boundary levels.

For the trade-off between speed and prediction accuracy, fast search is more biased to speed while fast matching is to prediction accuracy. In other words, it is difficult to achieve the best trade-off points using only a fast search or a fast matching method alone. Meanwhile, the combination of fast search and fast matching may be more effective and efficient in practice. In **Chapter 4**, a new combination scheme, Multilevel Adaptive Refining Search (MARS) is proposed for achieving better trade-off points of speed and prediction accuracy in fast motion estimation. By applying different search strategies at different levels in a multilevel elimination hierarchy, an effective marriage of fast search and fast matching can be achieved thus resulting in an effect of “ $1 + 1 > 2$ ”. The proposed MARS can also be regarded as a progressive *maximum a posteriori* (MAP)-based estimation method, where priori information of boundary values is used to progressively guide the fast search level by level with increasing accuracy. Besides, strong neighboring correlations in temporal and spatial domains are employed to refine the search range in the MARS framework so as to find a most suitable one for each current block. With an adaptively determined search range, more computations can be saved.

List of Figures

1.1 Motion estimation for a current block from a MSW in the reference frame. Each “dot” represents a checking point 2

2.1 (a) DS (**Diamond Pattern**); (b) HEX (**Hexagon Pattern**) 9

2.2 Neighboring blocks of current block (i, j) , where blocks in gray are in the current frame with symbol CN while blocks in black are in the reference frame with symbol RN 11

2.3 Neighboring blocks used in some algorithms [21,53–58](only spatial blocks) 12

2.4 Neighboring blocks used in some algorithms [52,59–67] (spatial and temporal blocks together) 13

2.5 (a) Pyramid with different resolution at each level. (b) Multigrid where only active region is further partitioned (c) Hierarchial structure with the same image size at each level 16

3.1 SEA illustration: Elimination at Level 0 where each arrow represents a candidate before the calculation of matching errors at Level 1. 31

3.2 MSEA illustration: Sequential elimination from Level 0 to Level 3 before calculation of matching errors at Level 4. 33

LIST OF FIGURES

iii

3.3 An example of partition order for 8×8 , 4×4 and 2×2 subblocks, respectively. Numbers indicate the partition order starting from 1 . . . 35

3.4 Partition process in the FGSE algorithm based on the partition order in Fig. 3.3 where there are totally 86 levels in contrast with only 5 levels (Level 0, Level 1, Level 5, Level 21 and Level 85) in MSEA. 36

3.5 FGSE illustration: Sequential elimination from Level 0 to Level 85 with tighter and tighter boundary values, where each arrow represents a candidate. Matching errors are calculated at the last level. 39

3.6 Rejection rate distribution at each level in FGSE-1 algorithm and MSEA for “Table-tennis”. Here Level 5, 21 and 85 correspond to Level 2, 3 and 4 in MSEA, respectively. Rejection rates at Level 0 and 1 are not plotted since they are same in the two algorithms. 49

3.7 Rejection rate distribution at each level in FGSE-1 algorithm and MSEA for “M&D”. 49

4.1 The illustration of the MARS applying different search strategies at different levels from Level 0 to Level 4 for block size of 16×16 60

5.1 Categorization of Existing Fast BMAs 76

List of Tables

2.1 Procedure of searching strategy in [105] 23

3.1 Procedure of SEA 32

3.2 Procedure of MSEA 34

3.3 Distribution of differences between neighboring candidates' rejection levels for different video sequences. **Region 0:** $diff \leq 4$; **Region 1:** $4 < diff \leq 16$; **Region 2:** $16 < diff \leq 64$; **Region 3:** $64 < diff \leq 85$. . . 42

3.4 Level distribution for 4 clusters with N=16 43

3.5 Procedure of FGSE 45

3.6 Computational cost comparison of different algorithms for **QCIF** video sequences. (FGSE-1: no prediction; FGSE-2: with prediction). 46

3.7 Computational cost comparison of different algorithms for **CIF** video sequences. 47

3.8 Comparison of average rejection levels using different algorithms. 47

4.1	Procedure of GEA. l' refers to a predetermined boundary level, after which a number of predetermined M checking points are selected for SAD calculations, $l' \in [0, 3]$ for block size $N = 16$. $\mathcal{D}_0$ is the maximum displacement in horizontal or vertical direction.	57
4.2	Procedure of MGEA, where sequential elimination and spiral-scan order are applied	59
4.3	Prediction accuracy (%) and speedup gain using motion information obtained from neighboring blocks based on the scheme in Table 4.4 . .	67
4.4	Procedure of utilizing priori motion information to adaptively determine the search range, where $\mathcal{D}_0$ is the original maximum displacement, $\mathcal{D}_1$ is the refined maximum displacement and N_{nei} is the number of the valid neighboring motion vectors.	67
4.5	Procedure of Multilevel Adaptive Refining Search (MARS)	70
4.6	Speedup gain of different algorithms compared to FSA (± 15). We define the speedup gain as $Gain = \frac{\beta}{\alpha}$, where β is the overall computational cost for FSA and α is the overall computational costs for any other algorithms in terms of number of operations simulated in the experiments.	71
4.7	Average MAD per pixel (± 15)	71
4.8	Prediction accuracy for each algorithm, which is defined as the percentage of the best matches found in each algorithm as compared with the FSA (± 15)	72

Chapter 1

Introduction

1.1 Motion Estimation

Video compression is an essential technology to enable digital video reliable either with transmission or storage. With proper VLSI implementations, video compression standards such as H.26x and MPEGx make digital video applications such as Multimedia streaming services and Digital TV a reality [2]. If no compression is applied, the transmission of uncompressed NTSC video signal, with 720 pixels 480 lines, 8 bits per pixel for each color component, and 30 frames per second, requires a bandwidth of 248.8 Mbps, and 109.35 Gbytes of storage capacity are needed to store one-hour video. In the real world, to manufacture the devices providing such high transmission rate and large storage capacity is extremely uneconomical. Video compression technique is the key to solving this problem, while motion estimation lies in the heart of it.

In international video coding standards, video compression is achieved by using motion-compensated (MC) transform coding scheme, which is composed of two most time-consuming parts: motion estimation and DCT coding with prediction error (Fig. 1.1). A motion estimation process is conducted to minimize temporal redundancies due to the similarity between successive frames, while DCT coding exploits the spatial redun-

Figure 1.1: Motion estimation for a current block from a MSW in the reference frame. Each “dot” represents a checking point

dancies in the prediction errors, which are generated according to the motion vectors obtained from the motion estimation process. Thus, the efficiency of MC transform coding technique is mainly dependent on how accurately the motion vectors can be estimated.

Among various motion estimation approaches, block-based motion estimation is the most popular one. It assumes that each frame is composed of moving blocks, within which motion of individual pixels is ignored. In other words, only translational motion model is considered in the block-based motion estimation. Considering the computational complexity and the hardware implementation, block matching algorithm (BMA)

among various block based motion estimation methods has become the natural choice of all the current international standards for digital video compression.

In BMA, all frames are partitioned into a number of non-overlapping rectangular blocks and a motion vector (MV) is obtained for each block in the current frame. The searching process is done by evaluating candidates within a predetermined motion search window (MSW) in the reference frame as shown in Fig. 1.1 to find out the best match that produces the minimum matching error according to a certain matching criterion, e.g., the widely used mean absolute difference (*MAD*) or equivalently sum of absolute difference/error (*SAD/SAE*). Computation is mainly consumed on the calculation of matching errors.

In recent years, fast block matching motion estimation has attracted a substantial amount of research attention and efforts though it has been studied for about two decades. Many algorithms have been developed for different applications. In this project, to find novel and more efficient BMAs is our research focus.

1.2 Motivation and Objective

Why are fast BMAs so important and attracting so many research efforts? The original motivation is due to the extremely high computational load of the conventional Full Search Algorithm (FSA), which is the simplest block matching algorithm but can give the optimal estimation solution by exhaustively checking all candidates within the MSW one by one. However, such exhaustive search with full matching-error calculation (*SAD*) at each checking point yields FSA extremely computationally expensive. Let us take an example to illustrate the heavy computational load of the FSA. Suppose that the size of MSW is 31×31 and block size is 16×16 for a video sequence with a format of QCIF (176×144) at a frame rate of 30 frames/second, it can be worked out that around 2.192 billionⁱ operations per second (addition, subtraction, absolute and comparison)

ⁱ $31^2 \times (256 + 256 + 255 + 1) \times (176 \times 144 / 16^2) \times 30 \approx 2.192 \times 10^9$

are needed to implement the FSA using *SAD* as the measure criterion. Apparently, such high computational complexities will make real-time video applications impossible. In fact, finding a best match is the most computationally intensive part in a video coding system, which could consume up to 80% of the total computational power in some cases if the conventional FSA is used [1]. This also indicates that motion estimation is a bottleneck problem in real-time video applications [3].

Various fast BMAs have been developed using different ways based on uni-modal error surface assumption, partial distortion rule, neighboring correlation information, pixel decimation technique and successive elimination with mathematical inequality. Algorithms such as new three-step search (N3SS) [4], diamond search (DS) [5–7], hexagon-based search (HEXBS) [8], normalized partial distortion search (NPDS) [9] and successive elimination algorithm (SEA) [10], to name a few, have shown their very good performance in practice. However, there still exist some drawbacks with each of them such as degraded prediction accuracy, e.g., N3SS, DS, HEXBS and NPDS, or limited searching efficiency, e.g., SEA. Therefore, it's our interest and task to develop some faster yet accurate motion estimation algorithms to strike a better trade-off between accuracy maximization and complexity minimization.

1.3 Outline and Contributions

An overview on the existing fast BMAs is presented in Chapter 2. The main contributions of this thesis are presented in Chapter 3 (fast matching with fine granularity successive elimination) and Chapter 4 (effective and efficient combination of fast search and fast matching). The final conclusion is drawn in Chapter 5 along with several future research directions.

In **Chapter 3**, we propose a more efficient scheme than the successive elimination algorithm (SEA) [10] and the multilevel successive elimination algorithm (MSEA) [11, 12] for fast optimal block matching, where a sequence of fine granularity boundaries

are obtained to eliminate checking candidates with as less computation as possible. Here, we coin the name of the new scheme as fine granularity successive elimination (FGSE) in contrast with the MSEA. Our proposed FGSE is actually a generalized case of the SEA and MSEA, which will be shown in the following sections. In the creation of the fine granularity boundaries, image block complexity is utilized to determine an adaptive order of partition in order to obtain relatively higher boundary values as early as possible, thus increasing the possibility of rejecting non-best candidates in the fastest way. It is observed that the sequential rejection process is not necessarily efficient in minimizing the computational cost. By utilizing the strong correlation of neighboring search candidates, a prediction scheme is employed to choose a starting rejection level rather than always sequentially checking each boundary from the beginning. In this way, some redundant boundary levels from the beginning can be skipped.

In **Chapter 4**, we propose a progressive estimation scheme, Multilevel Adaptive Refining Search (MARS), where different fast search strategies are adaptively incorporated into a multilevel elimination framework in fast matching at different levels. The proposed MARS framework can be regarded as an effective marriage between fast search and fast matching, where the boundary values that are considered as a series of approximates to the true *SAD* values are used to guide fast search level by level progressively. In this way, the interplay of fast search and fast matching can be fully exploited to result in an effect of “ $1 + 1 > 2$ ”. Both priori and posteriori information are utilized together in the proposed MARS to produce better performance. Besides, we employ neighboring correlations in temporal and spatial domains to refine the search range for each current block so as to further improve the performance of the proposed MARS scheme.

Chapter 2

Fast Block Matching Algorithms -Literature Review

2.1 Introduction

In all the current video coding standards, block matching motion estimation has an undoubtedly important role in achieving high coding efficiency. However, its process may be too time-consuming and could make the whole coding system unreliable and unpractical, especially for real-time video applications. To overcome this problem, various fast block matching algorithms have been developed applying diversified methods.

In this chapter, we make a brief overview on these existing fast BMAs. Matching criteria are defined first in Section 2.2. The search strategy of the conventional FSA is briefly introduced in Section 2.3. All the fast BMAs are roughly classified and reviewed in Section 2.4 and we conclude this chapter in Section 2.5.

2.2 Matching Criterion

A checking point is evaluated using some matching criterion such as sum of absolute difference (*SAD*) or mean square error (*MSE*). Other matching criterions such as maximum matching pel count (*MPC*) will not be considered in this thesis.

Assume $f_c(i, j)$ and $f_r(i, j)$ are the pixel intensities (all positive) at location (i, j) in current frame and a reference frame, respectively. The size of a block used for matching is $N \times N$ pixels and (m, n) is a motion vector within the MSW in the reference frame.

The sum of absolute difference (*SAD*) is defined as follows:

$$SAD = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} |f_c(x+u, y+v) - f_r(x+m+u, y+n+v)| \quad (2.2.1)$$

which is the most popular choice for VLSI implementations and is used in this thesis as the matching criterion.

The mean square error (*MSE*) is performed as:

$$MSE = \frac{1}{N^2} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} (f_c(x+u, y+v) - f_r(x+m+u, y+n+v))^2 \quad (2.2.2)$$

which, however, is not commonly used in hardware implementation because the square operation is difficult to be realized in hardware.

2.3 Full Search Algorithm (FSA)

In the conventional FSA, optimal solutions for motion vectors are ensured with every checking point being evaluated within a predefined MSW by calculating its matching error. A block is determined as a better-matched one if its *SAD* is smaller than the current minimum *SAD* denoted as SAD_{min} . This searching process can be considered as an elimination process since it progressively eliminates those non-best candidates during the process. Although enabling optimal solutions for motion vectors, as mentioned, the computational complexity is extremely high, especially with a large MSW.

2.4 Fast Motion Estimation

The overall complexity of any fast BMA relies on (i) the number of candidates within the MSW, and (ii) computations required to identify whether a candidate is a better

match or not at each checking point. Therefore, the existing fast BMAs are mainly designed in the two directions, i.e., reducing the number of checking points (referred to as Fast Search) and lowering computational complexity at each checking point (referred to as Fast Matching) [13]. With the former, the best match is found only from a subset of all the checking points that are selected according to predetermined patterns (Section 2.4.1), neighboring correlations (Section 2.4.2) or the coarse-to-fine prediction in a hierarchical approach (Section 2.4.3). Instead of reducing the number of checking points, many fast matching algorithms have been developed to eliminate each non-best candidate using lesser computation such as partial distortion search (PDS) (Section 2.4.4), pixel decimation (Section 2.4.5) and inequality-based successive elimination search (Section 2.4.6). Note that fast matching algorithms do not necessarily provide optimal solutions for motion vectors as the FSA.

I. FAST SEARCH

2.4.1 Fast Search Based on Uni-Modal Error Surface Assumption

During the process of searching the best match, the texture feature of the search area is vital to the complexity of the motion estimation, i.e., the more complex the texture feature, the higher the computations generally required to find the best match and vice versa. In many cases, it may be reasonable to assume that the matching error SAD decreases monotonically as the search location moves away from the best matching location (global minimum). Based on this assumption, many fast search algorithms have been proposed to reduce computations by only evaluating a predetermined subset of the candidate checking points. Different patterns have been designed in this category of algorithms, such as Square Pattern [4, 14–16], Cross Pattern [17, 18], Diamond Pattern [5–7], Octagon Pattern [19], Hexagon Pattern [8], Pentagon Pattern [20], Rood Pattern [21], Kite Pattern [22, 23] and so on. Using different step size and search strategy, the well-known fast search algorithms based on

Figure 2.1: (a) DS (**Diamond Pattern**); (b) HEX (**Hexagon Pattern**)

the uni-modal error surface assumption include 2-D logarithmic search (2DLS) [17] and its modified version (M2DLS) [17], three-step search (3SS) [14] and modified 3SS such as new three step search (N3SS) [4], Dynamic search-window adjustment search (DSWA) [24] and those developed in [25–29], conjugate-direction search such as one-at-a-time search (OTS) [14] and 1-D gradient descent search (1DGDS) [30], parallel search (PHODS) [31], orthogonal search (OS) [32] and the modified version [33], cross search (CS) [18] and its modified version [34], four-step search (4SS) [15], block-based gradient descent search (BBGDS) [16], simple and efficient search (SES) [35], two-step search (TWSS) [36], minima bounded area search (MIBAS) [37], diamond search (DS) [5–7] and its modified version [38–42], polynomial search (PS) [43], adaptive rood pattern search (ARPS) [21] and the modified version [44,45], pentagonal search (PFBMA) [20] and the Hexagon-based search (HEXBS) [8] and its modified version [46,47]. Due to the considerable simplicity and regularity, the popular video coding standards have adopted some of these fast search algorithms to improve the coding efficiency. The common strategy of these algorithms is that the best point found in the current step

will be used as the search center in the next step. As a matter of fact, the uni-modal error surface assumption, however, does not always hold, and as a result the searching process could be trapped into a local minimum. To alleviate this problem, some algorithms may use several best points found in the current step for further searching in the next step, namely multi-candidate technique [25] or utilize the strong spatial/temporal correlations among neighboring blocks [30].

To gain an insight into the algorithms in this category, the procedures of DS and HEX are briefly illustrated as in Fig.2.1. In the initial step, the large pattern as defined is applied, including $MV_{(0,0)}$. The small pattern is used only when the SAD_{min} point is at the center of the large pattern. HEX generally performs better than DS, especially for medium to large motion estimation. The enhanced HEX algorithms with faster inner search and prediction [46, 47] can further reduce the number of checking points and distortions.

From the above description about those existing fast search algorithms, we are aware of that the searching process will not stop until all predetermined checking points have been evaluated. However, it may not be flexible and effective enough in reality because of computation power constraint in a real-time coding system. Tai et al. [48] proposed a computation-aware (CA) scheme, which can be integrated with some existing fast search algorithms such as N3SS, 4SS and DS to conduct the fast motion estimation only using available computation power, which may be less or more than the computation power consumed on evaluating the predetermined checking points as usual. Also, some algorithms adaptive to a target computational complexity are proposed in [49, 50].

In spite of the weakness of being easily trapped into local minimums due to the uni-modal error surface assumption and thus achieving only suboptimal solutions, the fast search algorithms in this category serve video compression applications reasonably well in the motion-compensated compression approach.

Figure 2.2: Neighboring blocks of current block (i, j) , where blocks in gray are in the current frame with symbol CN while blocks in black are in the reference frame with symbol RN

2.4.2 Fast BMAs Using Temporal/Spatial Correlations

In this subsection, we will review the algorithms that have been proposed in the last 20 years to give an insight of temporal/spatial correlations.

Neighboring blocks of the current block (i, j) both in temporal and spatial domains are shown in Fig. 2.2, where blocks (RN0 $\cdots$ RN8) in black refer to neighboring blocks in the reference frame while the blocks (CN0 $\cdots$ CN4) in gray are in the current frame, respectively. The corresponding motion vectors are defined as $MV_{CN0} \cdots MV_{CN4}$ and $MV_{RN0} \cdots MV_{RN8}$. According to the experiments conducted in [51, 52], the neighboring blocks in Fig. 2.2 can be sorted as: RN0, CN1, CN3, CN2, CN4, RN5, RN6, RN7, RN8, RN1, RN2, RN3, RN4, CN0 in descending order of priority, i.e., RN0 has the highest correlation with the current block.

In most moving scenes, the motion field varies smoothly in both temporal and spatial directions, which is actually due to two observations:

- objects often cover more than one blocks of size 16×16 .
- motion is temporally continuous in general.

Although these may not be always valid, especially when the moving scenes contain complex motions, they are still reasonable for most cases in the real world. Based on

Figure 2.3: Neighboring blocks used in some algorithms [21,53–58](only spatial blocks)

this hypothesis, many algorithms have been proposed to efficiently exploit the spatial or temporal correlations to reduce computations of the motion estimation [21, 51–69], mainly in three ways:

- 1) Find a good starting point (predictor) that may be close to the global minimum, following which a refinement process is generally applied. (Here the predetermined “starting point” is considered to be the center of MSW)
- 2) Apply thresholding technique by calculating *SAD* at the selected neighboring *MVs* or *MV* derived from the neighboring *MVs* to early stop the estimation process.
- 3) Predict the possible motion characteristics of the current block using motion information from the neighboring blocks, and then apply different search strategies accordingly.

These approaches may be combined together in a different way for different algorithms. For example, an algorithm may find a good starting point first and then use informa-

Figure 2.4: Neighboring blocks used in some algorithms [52, 59–67] (spatial and temporal blocks together)

tion from the neighboring blocks to adaptively change the size of the MSW, while a neighboring motion vector can be directly used as the best match based on thresholding technique in another algorithm.

Before processing the neighboring motion vectors, one interesting problem needs to be settled: how many neighboring motion vectors are enough for the prediction? In consideration of simplicity, a fixed number of neighboring blocks is generally selected regardless of whether the selected neighboring blocks have the same motion vectors or not. As in standards MPEG-1/2, Nie and Ma only use CN1 as the predictor in the rood-pattern search algorithm [21] (Fig. 2.3 (a)). Besides of CN1, CN3 is used in [53] (Fig. 2.3 (b)), CN3 and CN4 are used in [54, 55] (Fig. 2.3 (c)). In [56, 57], the 4 spatially neighboring blocks in the current frame around the current block are used (Fig. 2.3 (d)). Jou et

al. used CN0, CN1, CN3 and CN4 as shown in (Fig. 2.3 (e)) [58]. However, prediction based on the temporal motion information is also effective although it needs more additional memories to store the neighboring motion vectors. As shown in Fig. 2.4, many algorithms [52, 59–67] also utilize some temporally neighboring blocks in the reference frame besides of using spatially neighboring blocks. To deal with the uneven error surface more effectively, Namuduri utilized 9 neighboring blocks of CN1-CN4 and RN0-RN4 in [69]. However, a fixed number of neighboring blocks could lead to missing of the best match in case of large and complex motions. To solve this problem, Tsai et al. dynamically select four neighboring motion vectors by using a motion-vector-state table according to a priority order determined experimentally in [51]. The number of neighboring blocks varies along with different algorithms. It may be better to make a selection according to the priority order.

A good starting point, which is usually better than $MV_{(0,0)}$, can generally be obtained by using correlation information. Better match always produces smaller matching error, the algorithms in [51, 53, 61, 68, 69] use the neighboring motion vector that produces the minimum matching error (SAD_{min}) as the starting point. Different with others, Antonio et al. in [68] proposed a refinement scheme to correct the starting point to achieve the real motion, where the correction vectors are on a grid, which is actually a 3×3 mask centered at the starting point. Chuang and Chang chose the motion vectors of four neighboring blocks shown in Fig. 2.3 (d) as four sperate starting points, at each of which a small search window is used for full search [52].

$$\mathbf{P} = [P^x, P^y]^T = \left[\sum_{n=1}^M W_n MV_{nei}^x[n], \sum_{n=1}^M W_n MV_{nei}^y[n] \right]^T \quad (2.4.3)$$

Besides of directly using a neighboring motion vector as the starting point, it can also be derived from these neighboring motion vectors. In [54, 62], the starting point is calculated using Equ. (2.4.3), where M is the number of neighboring blocks used in the algorithm, $MV_{nei}[M]$ is the set of neighboring motion vectors, $\mathbf{P}$ is the starting

point and $\mathbf{W} = [W_1, W_2, W_3, \dots, W_M]^T$. In [62], $\mathbf{P}$ is corrected by using an offset which is calculated by considering the direction of MV_{RN0} in the reference frame. In [56], the starting point $\mathbf{P}$ can be obtained using Equ. (2.4.4), where Th is a predefined threshold and 4 spatially neighboring blocks ($CN1 - CN4$) are used as shown in Fig. 2.3 (d).

$$P = \begin{cases} MV_p & \text{if } \max \|MV_{nei}[n] - MV_m\| \leq Th, n \in [1, 4] \\ MV_{(0,0)} & \text{otherwise} \end{cases} \quad (2.4.4)$$

$$MV_p = \arg \min \|MV_{nei}[n] - MV_m\|$$

$$MV_m = \frac{1}{4} \sum_{n=1}^4 MV_{nei}[n]$$

In this way, the good starting point can be obtained while maintaining the center-biased property. Additionally, the median vector of $MV_{nei}[M]$ whose target blocks are shown in Fig. 2.3 (c) is adopted in H.263 and MPEG-4, which is studied in [67] where it was also observed that motion vectors are more median-biased than center-biased. Before determining the starting point, any neighboring motion vectors can be used as the best match based on a certain criterion such as thresholding technique [51, 59, 60, 64] and correlations among the neighboring motion vectors [65] or both [67].

After a good starting point is found, a refinement process is generally applied to mitigate the risk of missing the best mach. This refinement may be done by applying a relatively small search window with full search [59, 60], the size of which may be determined using magnitude of neighboring motion vectors [61]. Besides, some conventional fast search algorithms [51, 56, 65] or movable small search window [54, 58, 62] may also be applied. Additionally, different search pattern can be dynamically selected using motion information from the neighboring blocks [63, 64, 66, 67]. For example, small diamond pattern is used for small motion while large diamond pattern is used for large motion, which is usually estimated based on neighboring motion vectors. In this way, unnecessary evaluations can be avoided.

It is noted that neighboring motion vectors may not be used to find a good starting point but to predict a suitable search strategy. In [55], Turaga designed a majority

Figure 2.5: (a) Pyramid with different resolution at each level. (b) Multigrid where only active region is further partitioned (c) Hierarchical structure with the same image size at each level

voting algorithm based on the neighboring motion vectors to determine whether the conventional FSA or 3SS is applied to find the best match.

Computational complexity after exploiting the temporal/spatial correlation information could be much smaller than that of the FSA. However, these algorithms could not work well for large and random motions. Furthermore, prediction error of motion vectors could be propagated from block to block and thus resulting in a remarkable degradation. Therefore, BMAs currently available based on neighboring correlations may be limited to applications with relatively slow and steady motions such as video conference [56].

2.4.3 Hierarchical/Mutiresolution Algorithms

The hierarchical [70–73] or multiresolution [74–87] algorithms rely on the idea of predicting an approximate large-scale motion vector in a coarse-resolution stage first and then refining it in a finer-resolution stage. A rough estimate for the displacement is determined at the coarser resolution level. Then it is passed onto the next finer resolution level as an initial estimate, where a relatively smaller search window is generally

used since the process starts with a good initial estimate.

Fixed image size and different block sizes, as shown in Fig. 2.5 (c), are used with hierarchical methods [70–73], which are actually based on an assumption that larger block could produce good initial estimate for smaller block size. However, this assumption may not be always true, thus probably leading to a poor estimate. For multiresolution methods, hierarchical representations of images in the form of a Laplacian pyramid as shown in Fig. 2.5 (a) are used, where the image size gradually decreases as the resolution becomes coarser.

According to the block size used, multiresolution algorithms can be categorized into two groups: variable block size [76,79,85,87] and constant block size [74,75,77,78,80–84,86]. In the former, an initial estimate is obtained using large blocks at the coarse resolution, which is used as a search center at the next level. Generally, the motion vector obtained from a larger block is more noise resistant than that from a smaller block [85]. However, it may not be a good estimate since more than one motions may exist more likely in one large block. This problem can be relieved in the latter approach since the block size is constant at all levels. Hence, this technique can achieve a fast search with a little performance degradation, even for video sequences containing complex motions but the motion vectors may be less robust to noise.

To increase the robustness of motion vectors obtained in a constant-block multi-resolution algorithm, some algorithms have been proposed by incorporating the multiple candidate search into the multiresolution scheme [77,78,80,82–84,86]. In [77,78,86], candidates are selected only on the basis of minimum distortion, and as a result a high computational cost is required. In [82], the neighborhood relaxation scheme is used. However, it only aims to improve the coding efficiency in very low bit-rate coding rather than the motion estimation performance itself. Recently, Song et al. proposed a fast binary pyramid motion estimation (FBPME) algorithm [84], where four candidates corresponding to four super-blocks are used and the binary block matching is adopted to reduce compu-

tational complexity. Besides of using multiple candidates, the neighboring correlations (temporal/spatial) are used in [80, 82, 83] to select a set of initial MV candidates more efficiently and reduce the computational complexity further.

In a hierarchical/multiresolution structure, many motion vectors at the lower level may be accurate enough, i.e., equal to or very close to the best match, thus giving a satisfactory motion compensated performance. In this case, it is not necessary to conduct further processing for these motion vectors. Motivated with this observation, a thresholding technique is incorporated into hierarchical search [73] and multiresolution search [79], which can reduce the computations from 14% to 20% of the conventional multiresolution algorithm.

A natural scene generally contains different motions, each type of which may require different block sizes for achieving the same prediction accuracy. Usually, large motion vectors can be accurately predicted using large block size while small block size may be more suitable for small motion vectors. Based on this observation, a multigrid structure as shown in Fig. 2.5 (b) was applied in [71, 72]. The multigrid algorithm starts from the coarsest level as the conventional hierarchical method does. Then the grid is split only in regions where the current resolution is judged unsatisfactory [72]. By adaptively splitting those necessary regions, computations possibly consumed on regions with large motions can be avoided, thus the overall complexity may be significantly reduced.

As moving from coarse to fine, a good initial estimate can be obtained and thus computations are significantly reduced. However, since sub-images need to be reconstructed, additional memories are used with the methods in this category compared with other fast search algorithms.

II. FAST MATCHING

The algorithms mentioned above are all in the line of fast search, where only a subset

of candidates is evaluated in finding the best match. In the following three subsections, fast matching algorithms being classified into 3 groups are presented, where the best match is obtained by checking all candidates but with matching criterions tighter than *SAD*.

2.4.4 Algorithms Using Pixel Decimation/Subsampling Techniques

When evaluating each checking point within the MSW in a reference frame, the full matching error is usually calculated using every pixel of the block. Since block matching motion estimation is developed based on an assumption that all pixels in a block have the same movement or displacement, it is reasonable to obtain a good estimate for the whole block only using a fraction of pixels at each checking point/block. Without limiting the number of checking points, some algorithms that attempt to reduce the computational complexity by subsampling the pixel blocks have been proposed [36, 88–100]. Not using all pixels, a good performance comparable to the conventional FSA can be achieved with pixel decimation.

The pixel decimation is obtained by using two types of patterns: fixed pattern [36, 88, 90, 91, 93, 96–98] and adaptive pattern [89, 92, 94, 95, 99, 100]. An adaptive pattern is generally obtained by considering the spatial feature of the block, and thus some representative pixels can be selected more efficiently than using a fixed pattern. On the other hand, a fixed pattern is easily implemented in practice since only predetermined pixels are used.

Bierling [70] introduced a hierarchical motion-estimation technique in which a first approximation of the motion field is obtained from low-pass-filtered and subsampled images. However, this algorithm only uses a uniform fraction of the pixels to make the matching calculation, and thus it could seriously affect the prediction accuracy since many pixels are heuristically ignored. To mitigate this drawback, Liu and Zaccarin [88] designed 4 different fixed patterns (4:1 subsampling) and applied them in an alternating

manner. In this way, all pixels in the current block can be used for matching error calculations. Later, this alternating method has been modified to a 16:1 subsampling algorithm [90] and a multi-stage version [97]. Cheng and Sun in [36] experimentally determined the possibly optimum number of pixels used for matching error calculation so that a MSE very close to that using the FSA can be achieved by using dither-order decimation scheme. In [89,92–94], it was observed that edge and boundary information in the block is important and should be considered as representative pixels of the block for the decimated matching error calculation. Therefore, most representative pixels such as edges and textures are selected to make the matching calculation, which is based on the fact that the pixels from highly active areas will mainly contribute to SAD. Image complexity based on the first-order edge detection technique is used to make the selection of pixels in [93]. Total of 14 edge patterns being designed based on the image complexity information at each subblock are used as subsampling patterns. Another adaptive pixel decimation scheme is proposed based on Hilbert-scanning order in [95], in which the decimation adaptivity is extended from local to global. Recently, Wang et al. [99,100] proposed a hierarchical decimation lattice called N-Queen lattice, which fully characterizes the spatial features of the block in the vertical, horizontal, and diagonal directions for both texture and edge areas.

BMA's applying pixel subsampling technique (with no reduction of the number of searching points) can achieve a relatively high accuracy. However, their speedup gains are limited, e.g., 4 times faster than the FSA with 4:1 subsampling [88] and 16 times faster with 16:1 subsampling [90]. Thus, these algorithms are usually integrated with some other fast search algorithms to make an additional reduction of the computational complexity.

2.4.5 Partial Distortion Search Algorithms

The partial distortion search (PDS) algorithm was first introduced in [101] for a vector quantization (VQ) coding process. Later on, it was applied to reduce computa-

tions for motion compensation/estimation in video coding [102, 103]. Some novel algorithms utilizing the idea of PDS have been proposed recently to significantly reduce the computations, either by designing a better searching order within the search window or by proposing an adaptive grouping method or matching order at each checking point [9, 104–115].

Suppose B is the set containing all pixels of a block, B_n is a subset of B , and there are totally M subsets. The Equ. (2.2.1) can be rewritten as:

$$SAD = \sum_{n=1}^M \sum_{(u,v) \in B_n} |f_c(x+u, y+v) - f_r(x+m+u, y+n+v)| \quad (2.4.5)$$

Let $PSAD_k$ be the matching error of the first k subset of B_n , which can be calculated using Equ. (2.4.6). Equ. (2.4.6) is equivalent to Equ. (2.4.5) when $k = M$.

$$PSAD_k = \sum_{n=1}^k \sum_{(u,v) \in B_n} |f_c(x+u, y+v) - f_r(x+m+u, y+n+v)|, 1 \leq k \leq M \quad (2.4.6)$$

According to Equ. (2.4.6), if the k^{th} accumulated partial distortion is larger than the current SAD_{min} , i.e., $PSAD_k \geq SAD_{min}$, the candidate will be eliminated and thus the remaining computation power is saved at this position.

The PDS algorithm is widely used to reduce the intensive computations involved in motion estimation. However, its speedup gain is normally limited to only a few times faster than the conventional FSA, which still needs to be improved further. There are two deterministic factors that could have dominant effects on the overall performance of the PDS algorithms: matching order and searching order. In the conventional PDS algorithms, $PSAD_k$ is calculated in a sequential order from top to bottom, left to right in the checking blocks, which is also the searching order within the MSW. In view of regular memory access, a sequential order is the best choice. However, it may not be worth if it is obtained at a cost of significant improvements to the speed of the estimation process. To further improve the efficiency of PDS, many variants have been proposed.

In [107–110], Kim et al. proposed several PDS algorithms based on Equ. 2.4.7, which was derived based on Taylor series expansion, where (m', n') is the optimal motion vector of the current block. From Equ. 2.4.7, Equ. 2.4.8 and Equ. 2.4.9, where $G[f(x, y)]$ is the gradient of image $f(x, y)$, while G_x and G_y represent the gradient components in x and y directions, respectively, we can deduce that the matching error (SAD) at position (x, y) is proportional to the gradient magnitude of the current block, which can be considered as the image complexity of the current block. In other words, gradient magnitude can give indications about which part of the current block could produce large matching errors. Motivated from this derivation, Kim et al. designed different algorithms based on image complexity of the current block to quickly eliminate those non-best checking points. In [110], the matching order is determined by sorting the summation of gradient magnitudes of 2 out of 4 equal subblocks of the current block. In [108], row-matching and column-matching orders are proposed. In [107, 109], local image complexity at a column or a small square block is studied. Partial distortion is calculated starting from the region (column or square block) with the largest gradient magnitude. Although adding some overhead computations, it produces a better performance than that of those previously proposed. Furthermore, Kim et al. also presented a matching order derived from a dithering matrix. According to Kim's experimental results in [107], matching order at a basis of small-square block always offers better performance than the row-based matching order as well as the dithering order, which can removes up to 9% of computations consumed by the conventional PDS with the sequential matching order.

$$\begin{aligned}
 SAD &= \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} |f_c(x+u, y+v) - f_r(x+m+u, y+n+v)| \\
 &\approx \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} \left| f_r(x+m'+u, y+n'+v) - f_r(x+m+u, y+n+v) \right| \\
 &\approx \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} \left| \frac{\partial f_r(x+u+m', y+v+n')}{\partial x} (m-m') + \frac{\partial f_r(x+u+m', y+v+n')}{\partial y} (n-n') \right| \\
 &\approx \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} \left(\left| \frac{\partial f_c(x+u, y+v)}{\partial x} (m-m') \right| + \left| \frac{\partial f_c(x+u, y+v)}{\partial y} (n-n') \right| \right)
 \end{aligned} \tag{2.4.7}$$

$$\|G[f(x, y)]\| = \left\| \begin{bmatrix} G_x \\ G_y \end{bmatrix} \right\| = \left\| \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \right\| = \sqrt{G_x^2 + G_y^2} \approx |G_x| + |G_y| \tag{2.4.8}$$

$$|G_x| = |f(x+1, y) - f(x, y)| \quad (2.4.9)$$

$$|G_y| = |f(x, y+1) - f(x, y)|$$

In [104], Chan et al. presented an adaptive partial distortion search algorithm (APDS) using Hilbert scanning order to select the most representative pixels. In this way, edges and texture information that would contribute to the matching error most can be easily identified. Besides of the matching orders mentioned above, Cheung and Po [9] proposed a normalized partial distortion algorithm (NPDS) where each subset B_n consists of 16 pixels spaced equally between neighboring pixels. This method can prevent each $PSAD_k$ from being localized in a particular region. Cheung et al. modified the NPDS to an adjustable partial distortion algorithm [111, 113] where prediction accuracy can be adjusted against the searching speed with a specified quality factor.

As mentioned above, searching order is also important for PDS algorithms. In [107–110], Jim et al. modified the conventional PDS algorithm with a spiral searching order and achieved a very good performance, which is because motion vectors are generally center-biased. The earlier the best match can be found, the earlier the non-best candidates may be rejected and thus the more computations can be saved. However, the center-biased property could not be always valid, especially when video sequence contains fast moving objects. In [105], Chan and Siu proposed a new searching strategy utilizing partial distortion, which is integrated with the Hilbert scan algorithm in [106]. The procedure is shown in Table 2.1.

Table 2.1: Procedure of searching strategy in [105]

```

BEGIN
  for each current block
    Step 1: Calculate  $PSAD_1$  at each point using Equ. (2.4.6) within the MSW
       $B_1$  contains 64 pixels evenly distributed over a block of size  $16 \times 16$ 
    Step 2: Select  $M$  points with the smallest  $PSAD_1$ 
    Step 2: Use the point with minimum  $SAD$  among the  $M$  points as the search center
  end for
END

```


From Table 2.1, partial distortion $PSAD_1$ is used for predicting a better center. Since $PSAD_1$ contains quarter of total pixels of a block, it is a good prediction of the SAD at the same position. These M points are most likely around the best match and thus this algorithm is efficient to find the best match. From the similar motivation, a PDS algorithm in [114] is proposed. The $PSAD_k$ is first calculated for every candidate within the MSW and then SAD is computed at the point producing the minimum $PSAD_k$. The rest of points will be evaluated in a normal PDS process. This idea was extended to multilevel version in [116, 117], where a hierarchical partial distortion search (HPDS) was proposed in three steps. At each step, a number of candidate motion vectors producing smallest $PSAD_k$ are selected for further searching in the next level. Clearly, the probability to find the best match at the beginning can be significantly increased in this way.

Some PDS algorithms applied half-way thresholding technique to stop the accumulation calculation of $PSAD_k$. Moshnyaga in [115] proposed a PDS algorithm based on an assumption that a candidate, with high matching similarity with the current block, has a small $PSAD_k$ at the accumulation step k . Thus those candidates whose $PSAD_k > Th'$ will be eliminated from further processing, where Th' is a dynamically modified threshold based on the minimum SAD for the block at the same position in the reference frame. Actually, the NPDS in [9] can also be considered as a PDS algorithm with thresholding technique, where SAD_{min} is the threshold value which can be gradually updated through the motion estimation process. Krista [114] also designed a thresholding technique to reduce computations by limiting the number of candidates for further processing. These PDS algorithms undoubtedly only provide suboptimal solutions for the motion vectors.

In summary, PDS algorithm is effective for motion estimation. It can be implemented in fast motion estimation with or without loss in prediction accuracy. However, the speedup gain is still moderate by using this kind of algorithms, e.g., NPDS is 16 times faster than the FSA at most. Furthermore, PDS using a set of pixels uniformly dis-

tributed on the block performs better than those using one row or one column of pixels [114]. The reason is that uniformly distributed pixels can prevent each $PSAD_k$ being localized at a certain region in the block, i.e., more globalized information that represents the block better is used at each accumulation step of the partial distortion calculation.

2.4.6 Inequality-based Fast Matching Algorithms Using Sum Norms

Similar to the PDS algorithms discussed in Section 2.4.5, algorithms in this category [10–12, 118–130] also attempt to reduce the matching computation at each checking point by establishing some tighter matching criteria so that those non-best candidates can be rejected without calculating their complete SAD s. Different from using partial distortion, sum norms are applied here, which can be considered as summation of all pixels within certain region. The prime idea of all these algorithms is from Equ. (2.4.10) and Equ. (2.4.11) as shown below, where N is the block size, $\mathbf{X}$ and $\mathbf{Y}$ are two $N \times N$ dimensional vectors. The proof of these two well-known equations can be found in [131].

$$|\|\mathbf{X}\|_1 - \|\mathbf{Y}\|_1| \leq \|\mathbf{X} - \mathbf{Y}\|_1 \quad (2.4.10)$$

$$\|\mathbf{X} - \mathbf{Y}\|_1 \leq N \|\mathbf{X} - \mathbf{Y}\|_2 \quad (2.4.11)$$

Let $C_0 = \|\mathbf{X}\|_1$ and $R_0 = \|\mathbf{Y}\|_1$, where C_0 and R_0 are the sum norms of the current block and the reference block, which are equivalent to the summation of all pixel intensities of the current block and the reference block, respectively. Based on Equ. 2.4.10, we can see that $BV_0 = |C_0 - R_0|$ is always less or equal to SAD , i.e., $BV_0 \leq SAD$ [10].

Based on this derivation, a successive elimination algorithm (SEA) [10] was developed by providing a decision boundary based on the sum norms of blocks to eliminate some checking points without calculating the computation-intensive matching error.

In [11, 12], the SEA was extended into a multilevel version that provides multiple levels

of tighter and tighter boundaries by using the sum norms of the block and the subblocks with reduced sizes. MSEA reduces computations by rejecting non-best candidates sequentially from the lowest level to the highest level, which results in a higher rejection rate before calculating the matching error thus faster matching than SEA.

Similar to PDS algorithms, the searching order is also important for both SEA and MSEA. Chang et al. [128] proposed an improved MSEA by designing an adaptive searching order, where the MSW is equally divided into 9 subregions and 9 different searching orders are defined according to the motion trend and the position correlation between the center subregion and its eight neighboring subregions determined from the predicted initial motion vector. Besides of the improvement with searching order, boundary values established in MSEA can still be improved further. In [127], we proposed a novel algorithm, Fine Granularity Successive Elimination Algorithm (FGSE) to create 86 boundary levels in contrast to 5 levels in MSEA. The details about SEA, MSEA and FGSE will be discussed in Chapter 3.

Rather than dividing blocks into small subblocks for achieving a more robust performance than SEA, one algorithm proposed in [120] attempts to efficiently calculate the matching error SAD by using BV_0 . From Equ.2.2.1, the SAD can be obtained as:

$$SAD = BV_0 + \begin{cases} 0 & d(u, v) \times (R_0 - C_0) \geq 0 \\ 2 \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} |d(u, v)| & d(u, v) \times (R_0 - C_0) < 0 \end{cases} \quad (2.4.12)$$

where $d(u, v) = f_r(u, v) - f_c(u, v)$.

Clearly, the matching error SAD at checking points not rejected at Level 0 can be efficiently obtained using Equ.2.4.12, which is faster than Equ.2.2.1. To reduce the overhead computations for calculating $2|d(u, v)|$, a lookup table was designed.

Generally, SAD for a checking point will be calculated if it passes all the boundary levels (one level in SEA and four levels in MSEA excluding the SAD level). Since SAD calculation is very intensive, it could require a lot of computation power even if

most of the checking points have been eliminated. To further improve the efficiency of successive elimination algorithms, PDS is applied in [119,122]. After boundary evaluation has been finished, partial distortion will be accumulated according to Equ. 2.4.6 to further reduce the number of checking points for complete *SAD* calculations. In this way, better performance than SEAs can be achieved.

Besides of using *SAD*, the *MSE* may also be used in this category to generate some tighter boundary values. It is actually related to the second-order sum norm, which was introduced in [119,120]. Combining Equ. 2.4.10 and Equ. 2.4.11, we can get the following relationship:

$$|\|\mathbf{X}\|_1 - \|\mathbf{Y}\|_1| \leq \|\mathbf{X} - \mathbf{Y}\|_1 \leq N \|\mathbf{X} - \mathbf{Y}\|_2 \quad (2.4.13)$$

Squaring both sides and dividing by N^2 :

$$\frac{|\|\mathbf{X}\|_1 - \|\mathbf{Y}\|_1|^2}{N^2} \leq \frac{\|\mathbf{X} - \mathbf{Y}\|_1^2}{N^2} \leq \|\mathbf{X} - \mathbf{Y}\|_2^2 \quad (2.4.14)$$

Then we can get:

$$\frac{BV_0^2}{N^4} \leq \frac{SAD^2}{N^4} \leq MSE \quad (2.4.15)$$

We can see that *SAD* calculation is necessary only for checking points that satisfies the Equ. 2.4.15. If a point's value of $\frac{BV_0^2}{N^4}$ is larger than *MSE* which is only calculated at the current best matching point, then it is definitely not a good match and thus can be rejected. Equ. 2.4.15 was also used in [120] to approximate *MSE* following a similar approach as shown in Equ. 2.4.12.

We are aware of the truth that every boundary value at each checking point can be considered as a coarse version of the corresponding *SAD*. In other words, such boundary value may explore some information about the final *SAD*. Similar to identifying the best match within the MSW, the best match may be found in the boundary plane. In [121], Chen et al. proposed a winner-update strategy (WinUp), where the candidate producing minimum boundary value has the highest priority to calculate its boundary

value at next level, and finally the candidate giving the minimum boundary value at *SAD* level is the best match. As compared with MSEA, WinUp algorithm is around two times slower according to the experimental results in [126]. Since boundary values can produce useful information about each checking point, and thus we can use them to predict the most possible region where the best match could be within the MSW. The global elimination algorithm (GEA) in [124, 126] are proposed in this direction. Its procedure is shown in Table 4.1. Calculating *SAD* only at 7 points with smallest boundary values at Level 2, GEA performs better than SEA and a little poorer than MSEA. We have more discussions about GEA in Chapter 4.

SEAs, WinUp and GEA have shown their excellent performance in terms of both speedup gain and prediction accuracy. Some algorithms proposed in this project utilizing sum norms and inequality will be presented in Chapter 3 and Chapter 4.

2.5 Conclusion

In this chapter, the conventional FSA is firstly introduced. After that, an overview on the existing fast BMAs is provided and some well-known motion estimation algorithms are introduced and briefly analyzed. We classified the fast BMAs into two categories: fast search and fast matching. Each family of those algorithms has its own advantages as well as disadvantages. Methods in Fast Search generally achieve a very fast estimation process at a cost of degraded image quality such as N3SS, DS and HEXBS, while some Fast Matching approaches may ensure the optimality of motion vectors but the speedup gain could not be significant as desired such as PDS, SEA and MSEA.

Chapter 3

Predictive Fine Granularity Successive Elimination for Fast Optimal Block Matching Motion Estimation (FGSE)

3.1 Introduction

According to the classification presented in Chapter 2, the algorithms of fast search reducing the number of checking points achieve the computation reduction by sacrificing the prediction accuracy. Some applications may suffer with inappropriate motion vectors estimated from such lossy fast search algorithms. Also the algorithms in fast matching using subsampling techniques normally introduce estimation errors. Therefore, the only way to accomplish the fast motion estimation with optimality guarantee is to perform full search in combination with such fast matching methods as partial distortion algorithms or algorithms based on sum norms and mathematical inequality. However, the speedup gain using partial distortion with optimality is considerably

limited [113], while the later one has shown more potential results.

In this Chapter, we propose a more efficient scheme than the successive elimination algorithm (SEA) [10] and the multilevel successive elimination algorithm (MSEA) [11, 12] for fast optimal block matching, in which a sequence of fine granularity boundaries are obtained to eliminate checking candidates with as less computation as possible. Here, we coin the name of the new scheme as fine granularity successive elimination (FGSE) in contrast with the MSEA. Our proposed FGSE is a generalized case of the SEA and MSEA, which will be shown in the following sections. In the creation of the fine granularity boundaries, image block complexity is utilized to determine an adaptive order of partition in order to obtain relatively higher boundary values as early as possible, thus increasing the possibility of rejecting non-best candidates in the fastest way. Further, the sequential rejection process from the first start is not necessarily efficient in minimizing the computational cost. By utilizing the strong correlation of neighboring search candidates, a prediction scheme is employed to choose a starting rejection level rather than always sequentially checking each boundary from the beginning. In this way, some redundant boundary levels from the beginning can be skipped.

To have a better understanding of fast block matching, SEA and MSEA are firstly reviewed in Section 3.2 before elaborating our proposed FGSE framework in Section 3.3. Experimental results of the proposed FGSE as compared to SEA and MSEA are shown in Section 3.4. Finally, we conclude this paper in Section 3.5.

Figure 3.1: SEA illustration: Elimination at Level 0 where each arrow represents a candidate before the calculation of matching errors at Level 1.

3.2 Review Relevant Successive Elimination Algorithms

3.2.1 Successive Elimination Algorithm (SEA)

According to the Equ. 2.4.10 in Chapter 2, where $|\|\mathbf{X}\|_1 - \|\mathbf{Y}\|_1| \leq \|\mathbf{X} - \mathbf{Y}\|_1$, the following inequality can be obtained:

$$|C_0 - R_0| \leq \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} |f_c(x+u, y+v) - f_r(x+m+u, y+n+v)| \quad (3.2.1)$$

where C_0 and R_0 are the sum norms of the current block and the reference block, respectively, i.e.,

$$\begin{cases} C_0 = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} f_c(x+u, y+v) \\ R_0 = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} f_r(x+m+u, y+n+v) \end{cases} \quad (3.2.2)$$

With Equ. (2.2.1), Equ. (3.2.1) can be rewritten as follows:

$$|C_0 - R_0| \leq SAD \quad (3.2.3)$$

From the Equ. (3.2.1), we can deduce that a candidate block will not be a better-matched block if its sum norm satisfies $|C_0 - R_0| \geq SAD_{min}$, where SAD_{min} is the current minimum SAD . Hence, the Equ. (3.2.1) can be considered as providing a boundary that can work as a “filter” to filter out a lot of non-best candidates before calculating computation-intensive matching errors, which leads to SEA [10].

Table 3.1: Procedure of SEA

<p>Algorithm: Successive Elimination Algorithm (SEA)</p> <p>Input: A current block and its candidate blocks within a predefined MSW</p> <p>Output: Best motion vector-MV_{best}</p> <p>Preprocessing:</p> <ol style="list-style-type: none"> 1) Calculating sum norms 2) Initialize SAD and MV_{best}, set $SAD_{min} = SAD_{MV_{(0,0)}}$, $MV_{best} = MV_{(0,0)}$ <p>BEGIN</p> <p> FOR each current block</p> <p> for each checking point</p> <p> Step 1: Calculate BV_0, if $BV_0 \geq SAD_{min}$, goto end for</p> <p> Step 2: Calculate SAD, if $SAD < SAD_{min}$, $SAD_{min} = SAD$, $MV_{best} = MV_{cur}$</p> <p> end for</p> <p> end FOR</p> <p>END</p>

The elimination process of SEA can be illustrated intuitively in Fig. 3.1. Assume sum norms of all candidates and of the current block are available. The procedure of SEA is listed in Table 3.1, where $BV_0 = |C_0 - R_0|$, namely boundary value. For each checking point, boundary value $|C_0 - R_0|$ at Level 0 is calculated first. Then it is compared with SAD_{min} . If $|C_0 - R_0|$ is not less than SAD_{min} , this candidate can be eliminated (filtered out). Otherwise, the candidate will pass through Level 0 filter and reach the level where its matching error SAD will be calculated. The more candidates are eliminated at Level 0, the more computational saving the SEA can achieve.

3.2.2 Multilevel Successive Elimination Algorithm (MSEA)

We can find that the sum norms of blocks contain a certain number of partial sum norms corresponding to smaller subblocks. Suppose $C_1^{(i)}$ and $R_1^{(i)}$ are partial sum norms for C_0 and R_0 , respectively, i.e., $C_0 = \sum_i C_1^{(i)}$, $R_0 = \sum_i R_1^{(i)}$. Based on Equ. 2.4.10, we can

Figure 3.2: MSEA illustration: Sequential elimination from Level 0 to Level 3 before calculation of matching errors at Level 4.

easily get the following inequality:

$$|C_0 - R_0| \leq \sum_i |C_1^{(i)} - R_1^{(i)}| \quad (3.2.4)$$

Such inequality is utilized to extend SEA to a multilevel case by partitioning blocks into subblocks with equal size at each level and then using sum norms of subblocks to provide several tighter boundary values.

In MSEA, $L = \log_2 N - 1$ additional boundary levels are established by dividing each block of $N \times N$ into subblocks of $N/2 \times N/2$, and further into $N/4 \times N/4$ until 2×2 . All subblocks at the same level are of the same size. At the l^{th} level where $0 \leq l \leq L$, the number of subblocks is $S_l = 2^{2l} = 4^l$ and the size of each subblock is $N_l \times N_l$, where $N_l = N/2^l$. Similar to the Equ. (3.2.4), the following can be obtained:

$$BV_l \leq BV_{l+1} \leq SAD, \quad BV_l = \sum_{k=0}^{S_l-1} |C_l^{(k)} - R_l^{(k)}| \quad (3.2.5)$$

where $C_l^{(k)}$ and $R_l^{(k)}$ denote the sum norms of the k^{th} subblocks at the l^{th} level in the current block (in the current frame) and in the checking/candidate block (in the reference frame), respectively.

Table 3.2: Procedure of MSEA

```

Algorithm: Multilevel Successive Elimination Algorithm (MSEA)
Input: A current block and its candidate blocks within a predefined MSW
Output: Best motion vector -  $MV_{best}$ 
Preprocessing:
    1) Calculating sum norms
    2) Initialize  $SAD$  and  $MV_{best}$ , set  $SAD_{min} = SAD_{MV_{(0,0)}}$ ,  $MV_{best} = MV_{(0,0)}$ 
BEGIN
  for each checking point
    Step 1: Calculate  $BV_l$  for  $0 \leq l \leq L$  using Equ. (3.2.5)
    Step 2: if  $BV_l \geq SAD_{min}$ , goto end for
             else  $l++$ , goto Step 1, until  $l = L$ 
    Step 3: Calculate  $SAD$ , if  $SAD < SAD_{min}$ ,  $SAD_{min} = SAD$ ,  $MV_{best} = MV_{cur}$ 
  end for
END

```

Clearly, we can see from the Equ. (3.2.5) that the boundary value BV_l increases with boundary Level l bounded by the matching error SAD . In fact, the matching error SAD is the boundary value for 1×1 subblocks. Also, SEA is actually a special case of MSEA, corresponding to 0-level MSEA [12]. In other words, MSEA can offer tighter boundary levels than SEA and thus more candidates may be eliminated before calculating the matching error. With $N = 16$, 4 boundary levels corresponding to 16×16 , 8×8 , 4×4 , 2×2 block/subblocks are established respectively in MSEA as shown in Fig. 3.2, which intuitively illustrates elimination process level by level in MSEA. According to the elimination procedure described in Table 3.2, one candidate is evaluated sequentially from the lowest Level 0 to the highest Level 3, where Level 0 in MSEA is the same as that in SEA. Each level contributes to eliminate a certain number of non-best candidates. At last, there are only a small number of candidates remaining for matching-error calculations, thus reducing a large amount of computation load compared with SEA. Similarly, the overall efficiency of MSEA is also somewhat affected by how fast the sum norms are calculated. MSEA employs a fast scheme for the calculation of sum norms in the multilevel case with about three times of memory usage and around double computation as compared with the SEA [12].

Figure 3.3: An example of partition order for 8×8 , 4×4 and 2×2 subblocks, respectively. Numbers indicate the partition order starting from 1

3.3 Proposed Fine Granularity Successive Elimination Algorithm

3.3.1 Limitations of MSEA

Although MSEA can substantially reduce the computational complexity [12], we still find two significant limitations in MSEA as follows:

- (1) The granularity between elimination levels is necessarily coarse due to the uniform partition at each level in MSEA. In other words, gaps between any two adjacent boundary levels are still very large, especially the gaps between the ending levels in MSEA. For example, before the computation-intensive *SAD* calculation at Level 4, in fact there exists a sequence of tighter boundaries between Level 3 and Level 4, which have not been taken into consideration in MSEA. Therefore, the efficiency of MSEA is undermined by losing the chance to reject some candidates earlier due to such large gaps. Also, image complexity is not considered in the creation of boundary levels.
- (2) The elimination process in MSEA always starts from the lowest level, although some candidates are unlikely to be rejected at lower levels. In fact, the rejection levels for neighboring candidates are reasonably correlated due to their spatial overlapping of most pixels.

Figure 3.4: Partition process in the FGSE algorithm based on the partition order in Fig. 3.3 where there are totally 86 levels in contrast with only 5 levels (Level 0, Level 1, Level 5, Level 21 and Level 85) in MSEA.

To attack these deficiencies in MSEA for higher efficiency, a new scheme termed as fine granularity successive elimination (FGSE) is proposed, which will be elaborated in the following four subsections.

3.3.2 Proposed FGSE Framework

As indicated in Equ. (3.2.4), more partitions can yield higher boundary values. It is worthwhile to note that those partitions can be in any types of shapes and sizes, e.g., formable or deformable, equal or unequal. For simplicity, we generally divide a block/subblock into formable partitions, particularly squares of equal size. A block of size $N \times N$ is firstly divided into 4 subblocks of equal size. Consider further partitioning each of the four subblocks one by one into 4 smaller subblocks of equal size. After

partitioning the first subblock, we have 7 subblocks (4 smaller subblocks and 3 larger subblocks), which can give a higher boundary value than that by those four larger subblocks. Larger and larger boundary values can be produced as the four larger subblocks are partitioned one after another, where after each partition of a subblock, a new and larger boundary value is generated. In this way, a sequence of boundary levels whose values gradually increase can be obtained. As a matter of fact, partitioning a single subblock into 4 smaller subblocks will increase the total number of subblocks by 3. Then the number of partitioning (denoted as “ L ”) and the total number of subblocks (denoted as “ W ”) have the relationship as $3 \times L + 1 = W$.

If we carry on such partition process until the size of a partitioned subblock is 1×1 (pixel size), we finally have $W = N \times N$ when all the partitioned subblocks are of size 1×1 . With $N = 16$, we can obtain $L = 85$. This implies that we can obtain a total of 85 boundaries excluding the one at Level 0 without partition, in sharp contrast with only 4 boundaries in MSEA. SAD (matching error) is calculated at Level 85. It can be conceived that the most efficient way in computation is to reject a candidate at the earliest possible level whose boundary value just exceeds the current threshold.

In the proposed FGSE algorithm, totally $L = (N \times N - 1)/3$ boundary levels are established by each partition of a block/subblock of size $m \times m$ into four subblocks of size $(m/2) \times (m/2)$ until no more partition for size 1×1 . There may be different partition ways. In FGSE, subblocks of smaller size will not be partitioned further until all the other larger subblocks have been partitioned. At the same level two different subblock sizes, $m \times m$ and $(m/2) \times (m/2)$ may exist. At the l^{th} level where $0 \leq l \leq L$, the number of subblocks is $S_l = 3l + 1$. In the following, we will show the relationship of all the boundary values from the first level to the last level.

Here we use $C_{m \times m}^{(k)}$ and $R_{m \times m}^{(k)}$ as the sum norms of the k^{th} subblocks of size $m \times m$ in the current block and in the reference block, respectively. Suppose the k^{th} subblock is divided into 4 subblocks of equal size, the sum norms of which are denoted as

$C_{(m/2) \times (m/2)}^{(k+2j+i)}$ and $R_{(m/2) \times (m/2)}^{(k+2j+i)}$, respectively, $i, j = 0, 1$. Then the following inequality holds:

$$\left| R_{m \times m}^k - C_{m \times m}^k \right| \leq \sum_{j=0}^1 \sum_{i=0}^1 \left| R_{(m/2) \times (m/2)}^{(k+2j+i)} - C_{(m/2) \times (m/2)}^{(k+2j+i)} \right| \quad (3.3.6)$$

We denote the boundary value at Level l as BV_l , which is obtained by:

$$BV_l' = \sum_{k=0}^{S_l-1} \left| R_{n(k,l) \times n(k,l)}^{(k)} - C_{n(k,l) \times n(k,l)}^{(k)} \right| \quad (3.3.7)$$

where $n(k, l) \times n(k, l)$ is the size of the k^{th} subblock at Level l . As mentioned above, $n(k, l) \times n(k, l)$ may be $m \times m$ or $(m/2) \times (m/2)$ depending on subblock index k and Level l . As l increases by 1, there is one more subblock of size $m \times m$ being partitioned into 4 smaller subblocks of size $(m/2) \times (m/2)$. Therefore, based on the Equ. (3.3.6), we can obtain the following relationships:

$$BV_l' \leq BV_{l+1} \quad \text{with} \quad BV_L' = SAD \quad (3.3.8)$$

Equation (3.3.8) means that boundary value BV_l tends to become larger as l increases, and all boundary values are not larger than the matching error SAD . On the other hand, the calculation of BV_l involves more computations as l increases.

Remarks:

A comparative analysis on the proposed FGSE scheme against MSEA is outlined in terms of computation efficiency in the following.

- 1) Boundary levels in MSEA are just a small subset of those used in our FGSE scheme. For $N = 16$, 85 boundary values from Level 0 to Level 84 have been established in FGSE algorithm, among which only 4 boundary values are used in MSEA. Much more boundary values in FGSE algorithm can help to reject candidates as early as possible. According to our experiments, the total rejection rate before the last level in our FGSE algorithm is significantly increased

Figure 3.5: FGSE illustration: Sequential elimination from Level 0 to Level 85 with tighter and tighter boundary values, where each arrow represents a candidate. Matching errors are calculated at the last level.

to around 99% for all the video sequences tested. In this way, computational cost in the last level of *SAD* calculation can be remarkably reduced. However, different partition order will result in different boundary level sequences although the number of boundary levels in each sequence remains unchanged. Therefore, we need to find the partition way which will produce the best possible sequence in terms of rejecting non-best candidates the fastest. Fig. 3.3 illustrates an example of partition order for partitioning 4 subblocks of 8×8 into 16 subblocks of 4×4 , then partitioning 16 subblocks of 4×4 into 64 subblocks of 2×2 and finally partitioning 64 subblocks of 2×2 into 256 subblocks of 1×1 , respectively. It should be noted that the partition order for 2×2 subblocks is based on the partition order for 4×4 subblocks. A detailed analysis on the partition order will be presented in the following *Subsection C*. Fig. 3.4 shows a partition process conducted according to the partition order illustrated in Fig. 3.3.

- 2) On the other hand, the increasing number of boundary levels in FGSE may also introduce some overhead computation, especially for those candidates that can only be rejected at the last level, since more boundary levels will be evaluated

before going to the last level. There are two ways to mitigate the side effect. The first one is the prediction of rejection level, which will be elaborated in the following subsection. Another possible way is to merge some boundary levels in the FGSE to produce a coarser boundary sequence with fewer boundary levels. Please note that increasing the levels of granularity may not always improve the performance. However, how to find the optimal number of levels will be a more challenging task. In this paper, we adopt the prediction scheme to alleviate the problem, which will be elaborated in *Subsection D*.

- 3) In FGSE scheme, the difference between boundary values at two adjacent levels $l - 1$ and l only lies on one subblock that has just been partitioned at the current Level l . Therefore, a large partial boundary value calculated at Level $l - 1$ can be utilized in establishing boundary value at Level l , thus saving computation power in obtaining boundary values. In contrast, in MSEA, all the boundary values have to be calculated independently at each level with equal-sized subblocks, since there is no any identical subblock at two levels.
- 4) From Fig. 3.4, we know that the sum norms for all the 8×8 , 4×4 and 2×2 partitioned subblocks are required to be calculated and stored before computing boundary values in block matching. As the same, all these sum norms are needed in MSEA for calculating boundary values at level 1, 5 and 21 shown in Fig. 3.4. In this sense, no additional computation or memory is required for calculating and storing the sum norms in our proposed FGSE compared with MSEA. The sum norms of all the blocks and subblocks can be efficiently calculated on the whole image using the fast calculation schemes in [10, 12] before conducting the block matching process.

3.3.3 Partition Order Based on Image Complexity

As mentioned above, a subblock will be partitioned into four smaller subblocks so that a tighter boundary value can be obtained. In a natural way, those larger subblocks may be partitioned in a sequential order from left to right and top to bottom. However, partitioning of different subblocks may offer different increases in boundary values due to the fact that subblocks have their own local features. According to the calculation formula for boundary values, i.e., Equ. (3.3.7), partition of a block with a larger variation will normally result in a big increase in the boundary value. Therefore, partition order of subblocks will affect how fast to reject a non-best candidate. In this subsection, image complexity of each subblock is utilized to determine the partition order, where subblocks with higher complexity will be partitioned first so that larger boundary values can be obtained as early as possible.

There are a few ways that can be used to measure the image complexity such as gradient magnitude and variance. Considering computational complexity, we use the gradient magnitude, the calculation of which is shown in Equ. (2.4.8). Gradient magnitudes at a pixel point is calculated in Equ. (2.4.9).

Let $\|GB(X, Y)\|$ represent the gradient magnitude for a subblock. It can be obtained by summing up all the gradient magnitudes of the pixels in the subblock as follows:

$$\|GB(X, Y)\| = \sum_{(x, y) \in \beta} \|G[f(x, y)]\| \quad (3.3.9)$$

where β represents the subblock, and (X, Y) is the position of the subblock. Based on the gradient magnitude of each subblock of the same size, we partition subblocks in the order of decreasing gradient magnitudes of subblocks, i.e., starting from the one with the largest subblock gradient magnitude, which can be expected to produce larger boundary values as earlier as possible.

In view that for each current block there are many candidate blocks to be evaluated, gradient magnitudes are only calculated in the current block instead of in the candidate

Table 3.3: Distribution of differences between neighboring candidates’ rejection levels for different video sequences. **Region 0:** $diff \leq 4$; **Region 1:** $4 < diff \leq 16$; **Region 2:** $16 < diff \leq 64$; **Region 3:** $64 < diff \leq 85$

Sequences		Region 0	Region 1	Region 2	Region 3
QCIF	Carphone	97.931	1.184	0.648	0.237
	Coastguard	97.535	1.945	0.320	0.200
	Miss	92.787	5.413	1.711	0.090
	M&D	97.389	1.618	0.935	0.060
CIF	BBC	87.535	8.677	2.933	0.855
	Dancer	81.896	12.364	5.135	0.605
	Garden	92.998	6.039	0.887	0.077
	Foreman	96.345	2.412	0.914	0.330
	Monitor	93.704	4.707	1.461	0.129
	Stefan	94.844	3.157	1.561	0.438
	Susie	90.840	7.618	1.438	0.104
	Table-tennis	85.046	12.411	2.253	0.288

blocks, which can save a lot of overhead computation. In addition, when a subblock becomes smaller such as 2×2 , the differences among the gradient magnitudes for these smaller subblocks are generally small due to the local smoothness of images. Therefore, for the partition order of the 64 subblocks of 2×2 , we make use of the partition order for 4×4 subblocks as a basis in combination with the conventional left-to-right, top-to-bottom order for 2×2 subblock partition within each 4×4 subblock. The example shown in Fig. 3.3 follows this partition order scheme. In this way, the high overhead computations for sorting 64 subblocks of 2×2 are saved.

It can be anticipated that this adaptive partition way will increase the possibility to reject checking points at lower levels due to that it can produce higher boundary values as earlier as possible. Our experiments show that the adaptive partition scheme can achieve around 10% speedup gain in average for 10 tested video sequences as compared with the conventional left-to-right, top-to-bottom method.

Table 3.4: Level distribution for 4 clusters with N=16

Cluster (k)	Level (l)
1	0, 1
2	2, 3, 4, 5
3	6, 7, 8, 9, ..., 21
4	22, 23, 24, 25,, 85

3.3.4 Predictive FGSE Utilizing Correlation of Matching Errors

In this subsection, we exploit the strong spatial correlation between two adjacent checking points to further improve the performance of our FGSE algorithm. From Fig. 3.4, we can see that non-best candidates will be rejected at a certain level from 0 to 85. In view that neighboring candidates in the search window share most of pixels with just one pixel shifting, their corresponding rejection levels are expected to be the same or very close to each other due to similar matching errors. To verify the correlation, more than 10 video sequences in two different formats have been used for the testing. The statistical results are tabulated in Table 3.3, where neighboring candidates refer to any two adjacent checking points along a spiral search path. It reveals that 82%-99% neighboring candidates are rejected at levels with differences less than 4 (i.e. in Region 0). Therefore, we can make use of this kind of correlation to predict a possible rejection level for the current candidate. In this way, we can overleap some levels and then avoid many overhead computations involved in establishing these boundary levels.

To simplify the prediction, we design a 1 – D and 1 – $Order$ prediction scheme based on the spiral-ordering search path. In this scheme, the rejection level for the current candidate is predicted by the rejection level for its immediately preceding candidate along the spiral path. In addition, we group all the 86 levels into 4 clusters shown in Table 3.4 in an aim to further simplify the prediction process. Another reason for the clustering is to make the prediction more robust in view that there are still some candidates whose rejection levels are quite different from their preceding ones although a majority of rejection level differences are contained in Region 0. The k^{th} cluster

contains levels from $\sum_{b=0}^{k-2} 4^b + 1$ to $\sum_{b=0}^{k-1} 4^b$ except the first cluster that starts from Level 0 to Level 1, where k is the cluster index ($1 \leq k \leq \log_2 N$). The first-order prediction scheme is designed as follows:

$$l' = \begin{cases} \sum_{b=0}^{k-2} 4^b, & \text{if } \sum_{b=0}^{k-2} 4^b + 1 \leq l \leq \sum_{b=0}^{k-1} 4^b, \quad k \geq 2 \\ 0, & \text{if } l \leq 1, \text{ i.e. } k = 1 \end{cases} \quad (3.3.10)$$

where l' refers to the predicted rejection level for the current candidate and l is the actual rejection level for the preceding candidate. We attempt to make the predicted rejection level less than or equal to the actual minimum rejection level so as to achieve the highest computational efficiency. The higher the rejection level is, the more computations will be involved. Moreover, the boundary value at Level 0 is always checked firstly before jumping to the predicted rejection level so that we can eliminate the possibility that a candidate that can be rejected at Level 0 is, however, rejected at some higher level due to a possible inaccurate prediction. We know that checking at Level 0 is the simplest, the computation of which is negligible compared with the checking in other higher levels. By trial and error, we find the above cluster-based prediction works well according to our experiments.

3.3.5 Procedure of FGSE Algorithm

The proposed predictive FGSE can be summarized using the C pseudocode in Table 3.5 on evaluating each candidate within a MSW for a current block:

From the above procedure, we can see that for each candidate, after the checking at Level 0, the evaluation starts from the predicted rejection level until the last level or the boundary value at a certain level is not less than SAD_{min} , whichever is earlier.

Table 3.5: Procedure of FGSE

Algorithm: Fine Granularity Successive Elimination (FGSE)
Input: A current block and its candidate blocks within a predefined MSW
Output: Best motion vector - MV_{best}
Preprocessing:
1) Calculating sum norms
2) Determine partition order based on image complexity for the current block
2) Initialize SAD and MV_{best} , set $SAD_{min} = SAD_{MV_{(0,0)}}$, $MV_{best} = MV_{(0,0)}$ BEGIN
do {
$l = 0$, calculate the boundary value BV_l ;
if ($BV_l' \geq SAD_{min}$)
return false; // not a better match
else {
$l = l'$; // l' is the predicted rejection level
if ($l == 0$) // predicted level is at Level 0
$l++$; // to check the next boundary level based on the determined partition order
}
for (; $l \leq 85$; $l++$){
calculate BV_l ;
if ($BV_l' \geq SAD_{min}$)
return false; // not a better match
else {
if ($l == 85$){
$SAD_{min} = BV_l$; // update the current SAD_{min}
return SAD_{min} ;
}
}
}
}while(!LAST CANDIDATE); // for all checking points along the spiral search path in MSW
END

3.4 Experimental Results

The above-described algorithms including SEA, MSEA and predictive FGSE algorithms were simulated on 12 video sequences with two different formats. These sequences provide a variety of motions ranging from slow movements (“Miss”), medium to fast movements (“Coastguard”, “Dancer”), to camera panning (“Stefan”) and zooming (“Table-tennis”). The luminance components of the first 100 frames in these video sequences are used in the simulation. As usual, the sum of absolute difference (SAD) is used as the distortion measure. The maximum displacement in the search window is ± 15 pixels in both horizontal and vertical directions for 16×16 block size. The coding pattern is IPP......

Table 3.6: Computational cost comparison of different algorithms for **QCIF** video sequences. (FGSE-1: no prediction; FGSE-2: with prediction).

	M&D		Carphone	
	Total operations	Speedup	Total operations	Speedup
FS	599808.000	1	599808.000	1
SEA	82815.955	7.243	89972.396	6.667
MSEA	24045.452	24.945	18461.632	32.489
FGSE-1	18116.683	33.108	14348.187	41.804
FGSE-2	16785.763	35.733	13737.208	43.663

	Coastguard		Miss	
	Total operations	Speedup	Total operations	Speedup
FS	599808.000	1	599808.000	1
SEA	211025.48	2.842	154658.941	3.878
MSEA	24427.897	24.554	57500.717	10.431
FGSE-1	17677.976	33.930	38050.745	15.763
FGSE-2	17462.417	34.349	34420.103	17.426

Three aspects of operations consume computational power in SEA, MSEA and our FGSE algorithms, i.e., (1) computation of boundary values, (2) calculation of sum norms, and (3) comparison of boundary values with the current minimum *SAD*. The first one consumes the dominant proportion of the total power, with above 90% according to our experimental results. The amount of computational power for calculating sum norms of blocks and subblocks can be significantly reduced using those schemes designed for fast calculations of sum norms in [10] and [12]. In our FGSE algorithm, computational power spent on the calculation of the sum norms is the same as those in MSEA. The overhead computation complexity is approximately around $5N^2$ operations per block in SEA and $9N^2$ per block in MSEA and our FGSE algorithms.

Since all the discussed algorithms are lossless and fast versions of the conventional FS algorithm, only computational cost is used for evaluation. Tables 3.6-3.8 tabulate all the simulation results for our proposed FGSE algorithm with or without prediction (denoted as FGSE-2 with prediction and FGSE-1 without prediction, respectively) as compared to the other fast algorithms SEA and MSEA. Tables 3.6-3.7 compare different algorithms in overall computational cost including addition, subtraction, absolute and comparison. The computations consumed on calculating the sum norms of all the blocks/subblocks and gradient magnitudes have been included in the overall compu-

Table 3.7: Computational cost comparison of different algorithms for CIF video sequences.

	BBC		Dancer	
	Total operations	Speedup	Total operations	Speedup
FS	666624.000	1	666624.000	1
SEA	258279.453	2.581	160496.607	4.154
MSEA	154636.602	4.311	55510.397	12.009
FGSE-1	116647.268	5.715	37346.721	17.850
FGSE-2	104383.091	6.386	34100.658	19.549

	Garden		Foreman	
	Total operations	Speedup	Total operations	Speedup
FS	666624.000	1	666624.000	1
SEA	205582.163	3.243	148507.700	4.489
MSEA	108653.893	6.135	35252.329	18.910
FGSE-1	83828.393	7.952	26289.178	25.357
FGSE-2	75021.191	8.886	24656.017	27.037

	Susie		Monitor	
	Total operations	Speedup	Total operations	Speedup
FS	666624.000	1	666624.000	1
SEA	199393.673	3.343	281213.232	2.371
MSEA	115378.160	5.778	181234.174	3.678
FGSE-1	74488.761	8.949	129327.708	5.155
FGSE-2	65270.701	10.213	115328.473	5.780

	Stefan		Table-tennis	
	Total operations	Speedup	Total operations	Speedup
FS	666624.000	1	666624.000	1
SEA	244395.623	2.728	368044.961	1.811
MSEA	42486.581	15.690	186662.470	3.571
FGSE-1	31332.963	21.275	120033.134	5.554
FGSE-2	30026.635	22.201	106108.264	6.282

Table 3.8: Comparison of average rejection levels using different algorithms.

	BBC	Dancer	Garden	Foreman	Monitor	Susie
FS	85.000	85.000	85.000	85.000	85.000	85.000
SEA	32.432	35.355	25.715	18.440	19.969	24.926
MSEA	14.401	16.942	9.985	2.939	4.877	10.626
FGSE-1	9.230	10.278	6.519	1.773	2.685	5.749
FGSE-2	9.326	10.397	6.557	1.821	2.740	5.799
	Stefan	Table-tennis	M&D	Carphone	Coastguard	Miss
FS	85.000	85.000	85.000	85.000	85.000	85.000
SEA	30.662	46.422	11.235	12.248	29.398	21.413
MSEA	3.646	17.468	2.099	1.506	2.153	5.668
FGSE-1	2.209	9.532	1.238	0.896	1.229	3.078
FGSE-2	2.300	9.646	1.261	0.930	1.274	3.137

tational costs. According to these results, our algorithms FGSE-1 and FGSE-2 can achieve a significant reduction in computation compared with others. For instance, for “Susie” sequence, the speedup gain of our FGSE-2 (i.e., the predictive FGSE) is 76.8% over MSEA. For “Carphone” sequence, our FGSE-2 is up to 43.66 times faster than the FS. The speedup gain of our predictive FGSE algorithm FGSE-2 over the MSEA is in the range of 34.4% - 76.8% for all the video sequences, while the FGSE-2 is around 3.1 - 12.1 times faster than SEA.

To further demonstrate the reasons behind the superiority of our FGSE algorithm to the MSEA, Fig. 3.6-3.7 plots the rejection rates at different levels for our FGSE-1 algorithm against the MSEA for video sequences “Garden” (FGSE-2 produces a similar distribution of rejection rates to that of FGSE-1). Different from the MSEA which can only reject checking points at a few levels (corresponding to Level 0, 1, 5, 21, 85 in FGSE), our FGSE algorithm eliminates candidates at each possible level. For example, most candidates which are rejected at Level 5, 21 or 85 by the MSEA, are now eliminated at earlier levels between Level 2 and 5, between Level 6 and 21, or between Level 22 and 85, respectively, by using our FGSE. Table 3.8 provides the average rejection levels for each video sequence using different algorithms. As expected, our FGSE-1 algorithm achieves the lowest average rejection levels since it can find the minimum rejective level for each checking point by sequentially and exhaustively evaluating each level. The FGSE-2 has achieved very close results in average rejection level to those of FGSE-1, which implies the proposed prediction scheme works well. Normally the smaller the rejection level is, the less computation will be consumed. However, FGSE-1 performs slightly poorer than FGSE-2 although FGSE-1 can find the smallest rejective level for each candidate. The reason is that FGSE-1 has to perform checking at each level sequentially before rejecting/accepting a candidate. Conceivably, the ideal algorithm is that which can reject a checking point just at its minimum rejective level without going through checking at any preceding levels. In practice, the faster to reach the minimum rejective level, the more efficient an algorithm becomes.

Figure 3.6: Rejection rate distribution at each level in FGSE-1 algorithm and MSEA for “Table-tennis”. Here Level 5, 21 and 85 correspond to Level 2, 3 and 4 in MSEA, respectively. Rejection rates at Level 0 and 1 are not plotted since they are same in the two algorithms.

Figure 3.7: Rejection rate distribution at each level in FGSE-1 algorithm and MSEA for “M&D”.

This is what our prediction scheme attempts to achieve by skipping as more unnecessary checking levels as possible. From the results shown in Table 3.6-3.7, up to 14.12% operations can be further saved using our prediction scheme in the FGSE algorithm.

3.5 Conclusion

A new framework, fine granularity successive elimination (FGSE) for fast optimal block matching in motion estimation has been developed, which provides a sequence of fine granularity boundary levels for candidate evaluation. Image complexity is utilized in determining the order of partitioning subblocks to establish the fine-grained boundary sequence so that larger boundary values can be obtained as earlier as possible, thus increasing the possibility of rejecting non-best candidates faster. In fact, the known SEA and MSEA are degraded cases within our proposed FGSE framework, in the sense that the boundary levels used in SEA and MSEA are just very small subsets of those in our FGSE. In view of the strong correlation of matching errors between adjacent candidates in the search window, a simple scheme to predict rejection levels has been proposed, making it possible to save some more computational effort by overleaping the unnecessary earlier levels. Experimental results have clearly shown that our proposed FGSE algorithm produces substantially higher speed-up gain over SEA and MSEA (up to 12.1 times faster than SEA and 76.8% improvement over MSEA) for all the testing video sequences in different formats and varying video contents.

As mentioned in Chapter 2, many other successive elimination algorithms have also been proposed. In our experiments, we only compared the proposed FGSE with SEA and MSEA because they are the most relevant algorithms to our FGSE. For the most recently developed global elimination algorithm (GEA) [124, 126], we don't make a comparison with our FGSE since it is a lossy method but the FGSE is lossless.

It is worth noting that some variants in the FGSE framework can be developed, e.g., some boundary levels in the FGSE can also be merged to form a coarser boundary sequence, or the partition way/order can be further modified. As more partitions are generated, especially at high levels such as those after Level 21, the increase of boundary values contributed by any further partition could not be significant. In this case, it may be more efficient to merge some boundary values so that the determining

boundary value can be reached fast. Additionally, we only use an 1-D prediction scheme to estimate rejection level for current candidate. Alternatively, a 2-D scheme may be designed, which can provide more information for the prediction process thus more accurate prediction for the current candidate's rejection level.

Chapter 4

Multilevel Adaptive Refining Search (MARS): A New Framework to Integrate Fast Matching and Fast Search for Fast Motion Estimation

4.1 Introduction

The intensive computations of the conventional FSA can be reduced by limiting the number of checking points (Fast Search) or lowering the computations consumed on matching each checking point (Fast Matching). Both approaches may only offer a limited performance, i.e., fast search can only achieve suboptimal solutions for motion vectors while the speedup gain by fast matching is relatively low compared with those well-known fast search algorithms. In other words, for the trade-off of speed and

prediction accuracy in the motion estimation, fast search is more biased to speed while fast matching is to accuracy (especially for those fast optimal matching algorithms). It seems to be difficult to find the best trade-off point using only a fast search or a fast matching method alone, if not impossible. Conceivably, it would be more effective and efficient if algorithms in these two classes are combined together to conduct the fast block matching motion estimation. In this way, the best motion estimation in terms of speed and accuracy may be achieved in practice.

However, how to effectively and efficiently combine both categories is still a question to be investigated. Conventionally, fast matching algorithms are generally used to reduce the matching power required at matching each checking point covered by a fast search algorithm as in [121]. That is, fast matching is incorporated into the fast search to further accelerate the motion estimation processing without changing prediction accuracy if not compromising. Although they have been combined together in this way, fast search and fast matching are in fact performed independently. No information from fast search is used for fast matching and vice versa. In other words, such combination is just simply putting them together with the effect akin to “ $1 + 1 = 2$ ”.

In this chapter, we propose a progressive estimation scheme, Multilevel Adaptive Refining Search (MARS), where different fast search strategies are adaptively incorporated into a multilevel elimination framework in fast matching at different levels. The proposed MARS framework can be regarded as an effective marriage between fast search and fast matching, where the boundary values that are considered as approximates to the true *SAD* values are used to guide fast search progressively level by level. In this way, the interplay of fast search and fast matching can be fully exploited to result in an effect of “ $1 + 1 > 2$ ”. Both priori and posteriori information are utilized together in the proposed MARS to produce better performance. Besides, in view that motions of adjacent blocks are highly correlated with each other, we employ neighboring correlations in temporal and spatial domains to adjust the size of MSW for each current block so as to further improve the performance of the proposed MARS scheme.

Some insightful remarks on the combination of fast search and fast matching are introduced first in Section 4.2. Then a multilevel version of global elimination algorithm (MGEA) is presented in Section 4.3, where the global elimination algorithm (GEA) [126] is reviewed at first. Our proposed MARS framework is elaborated in Section 4.4. Experimental results of the proposed MARS as compared to some well-known fast search and fast matching algorithms are shown in Section 4.5. Finally, we conclude this chapter in Section 4.6.

4.2 In-depth Perspectives on the Combination of Fast Search and Fast Matching

Most of the existing fast BMAs are developed either in the class of Fast Search or Fast Matching for motion estimation. As mentioned, the best trade-off point of speed and prediction accuracy may not be achieved using a fast search or a fast matching method alone, whereas the combination of these two classes can produce more encouraging performance with better trade-off points of speed and prediction accuracy.

As a natural approach, fast matching is generally incorporated into fast search to speedup the search process by reducing computations at each checking point [88, 121]. However, it is observed that fast search and fast matching are still performed independently in this way. That is, fast search is used to reduce the number of checking points while fast matching is employed to lower the computation power required for matching each checking point. In other words, this combination approach is just doing like “ $1 + 1 = 2$ ”, which means the performance of the combined method is a simple aggregation of predetermined performance of the fast search and fast matching. For example, in the combination of 3SS with 2:1 pixel decimation [88] and 3SS with fast matching method WinUp [121], the speedup gain of the combined method is due to the contributions of 3SS and fast matching, respectively, while its prediction accuracy is a little worse [88] or the same as that of 3SS [121].

In point of the effectiveness and efficiency for the combination, we have two comments as follows:

- 1) To maximize the effectiveness and efficiency for the combination of fast search and fast matching, fast search and fast matching needs to be performed interactively rather than independently. In other words, fast search and fast matching can contribute to each other during the motion estimation process so that useful information generated by one side can be utilized in another side for more efficient fast search or fast matching. In this sense, this combination approach can be regarded as an effective marriage between fast search and fast matching, where both behave interactively to achieve better trade-off points between speed and prediction accuracy.
- 2) A fast search method normally reduces the number of checking points using some neighboring information or a predefined search pattern. The fast search method alone cannot tap into the distortion information of checking points to reduce the number of checking points since the distortion information is unknown until the points have been examined. However, we know that many distortion related information such as boundary values in the fast matching method MSEA [12] is generated during the fast matching process. These boundary values can be treated as approximates to the final *SAD* distortions. Therefore, making use of the boundary values as *a priori* information for the distortions, the combination of fast search and fast matching can be formulated as a *maximum a posteriori* (MAP)-based or Bayesian-based motion estimation method [2]. In contrast, the conventional fast search method does not utilize the *a priori* information thus it can be regarded as a *maximum likelihood* (ML)-based approach by assuming each checking point selected having the same probability to be the best match.

Based on these the perspectives, we propose a *maximum a posteriori* (MAP)-based progressive estimation method by performing a more effective and efficient combination of

fast search and fast matching. In a multilevel elimination hierarchy, priori information of boundary values is fully utilized to guide fast search level by level progressively, where boundary values are regarded as approximates to the true *SAD* values with increasing accuracy. In this way, fast search is smartly guided by fast matching rather than being performed independently from fast matching.

In the following, the Global Estimation Algorithm (GEA) and its modified version Multilevel Global Estimation Algorithm (MGEA) will be firstly discussed before presenting our proposed more effective combination algorithm. The GEA and MGEA can be regarded as two preliminary and simplified examples of the effective combination of fast search and fast matching.

4.3 Multilevel Global Elimination Algorithm

4.3.1 Global Elimination Algorithm [126]

According to Equ. 3.2.5, boundary value BV_l increases with boundary level l bounded by the matching error *SAD*. In other words, boundary values move closer to *SAD* as more partitions are introduced, where more detailed information about the checking block is utilized as boundary level l increases. As a matter of fact, every boundary value at each checking point can be considered as an approximate to the corresponding *SAD*. As a whole, boundary values of all checking points within the MSW at the same level create a boundary-value plane together, considered as a coarse version of *SAD* values in the MSW. As l increases, it becomes closer and closer to the true *SAD* values. Intuitively, a checking point with smaller boundary value compared with the other candidates with larger boundary values at the same level is more likely to be the best match. In other words, small boundary value may lead to small *SAD* and vice versa. Based on this hypothesis, Huang, Chien, Hsieh and Chen [126] developed a global elimination algorithm (GEA), where early rejection decision is separated from

SAD calculation for each candidate block to ensure a regular data flow thus particularly suitable for hardware design. Its procedure is briefly described in Table 4.1.

Table 4.1: Procedure of GEA. l' refers to a predetermined boundary level, after which a number of predetermined M checking points are selected for *SAD* calculations, $l' \in [0, 3]$ for block size $N = 16$. $\mathcal{D}_0$ is the maximum displacement in horizontal or vertical direction.

Algorithm: Global Elimination Algorithm (**GEA**)
 Input: A current block, its candidate blocks within a predefined MSW, level l' and the number M
 Output: Best motion vector - MV_{best}
 Preprocessing: Calculating sum norms
BEGIN
 Step 1: Calculate $BV^{l'}[n]$, $n \in [0, (2\mathcal{D}_0 + 1)^2 - 1]$, where $(2\mathcal{D}_0 + 1)^2$ is the number of candidates in the MSW
 Step 2: Find the first M points with smallest $BV^{l'}$
 Step 3:
 for each candidate checking point with $m \in [0, M - 1]$
 Calculate $SAD[m]$ and find the minimum *SAD* and the corresponding MV_{best}
 end for
END

In GEA, boundary values $BV^{l'}$ at Level l' for all the candidate blocks are calculated first. Then, a predefined number of M candidates producing the M smallest boundary values are selected for *SAD* calculations, among which the candidate block producing the minimum *SAD* is used as the best match. In this way, computations required in *SAD* calculations for the rest $(2\mathcal{D}_0 + 1)^2 - M$ checking points are avoided, where $\mathcal{D}_0$ is the maximum displacement in horizontal and vertical directions and $(2\mathcal{D}_0 + 1)^2$ is the total number of candidates in the MSW. Since computation consumed on boundary value calculations is much lower than that required in *SAD* calculations, a large amount of computation power can be saved. From the experimental results tabulated in Table 4.6 and Table 4.8 with appropriate choices of M and l' , we can see that a significant computation reduction can be achieved in GEA, which is constantly 10.806 and 10.948 times faster than the FSA for QCIF and CIF video sequences, respectively. Meanwhile a comparable prediction accuracy to that of the FSA is maintained.

As we know, a very regular data flow is extremely important in hardware design. To ensure this, the early rejection before level l' is not applied in GEA and raster scan

order is used. However, a regular data flow is not necessary in software design for fast motion estimation. Therefore, a more efficient and modified GEA can be naturally obtained by applying the sequential elimination at levels before l' in order to reduce as more computations as possible, where the spiral order is also employed. The modified version of GEA is elaborated in the following subsection.

4.3.2 Multilevel GEA

In this subsection, we modified the GEA into a multilevel version for software design purpose, namely multilevel global elimination algorithm (MGEA), where the sequential elimination is naturally applied to sequentially reject non-best candidates from Level 0 to Level l' . In this way, many non-best candidates may be rejected before Level l' , and only a small number of checking points may be remaining for more computation-complex boundary value calculations at Level l' .

There are two parameters that totally determine the trade-off point of speedup gain and prediction accuracy in GEA and MGEA, i.e., the level l' and the number M . However, the choices of M and l' seem to be difficult. According to Equ. 3.2.5, small speedup gain and high accuracy are achieved when l' is set high due to the truth that boundary value at a checking point becomes closer to the corresponding SAD as boundary level l increases. Similarly, large M results in high accuracy but leads to a small speedup gain, which may decrease logarithmically as l' increases. An empirical choice of l' and M is presented in [126], where $l' = 2$ and $M = 7$ are experimentally found to be a good choice.

The procedure of MGEA is described in Table 4.2. In MGEA, every boundary value at each level is compared with the current SAD_{min} and the corresponding checking point will be rejected if it is not smaller than SAD_{min} . At last, M points producing minimum $BV^{l'}$ are selected for SAD calculations. In comparison with GEA, computation saving in MGEA is mainly from the reduction of checking points remaining for boundary value

calculations at the selected Level l' . Additionally, the number of remaining checking points at Level l' , i.e., candidate checking points remaining for SAD calculations, may be smaller than the selected number M in some cases, thus more computations can be saved as compared with the original GEA.

Table 4.2: Procedure of MGEA, where sequential elimination and spiral-scan order are applied

```

Algorithm: Multilevel Global Elimination Algorithm (MGEA)
Input: A current block, its candidate blocks within a predefined MSW, level  $l'$  and the number  $M$ 
Output: Best motion vector -  $MV_{best}$ 
Preprocessing:
    1) Calculating sum norms
    2) Initialize  $SAD$  and  $MV_{best}$ , set  $SAD_{min} = SAD_{MV_{(0,0)}}$ ,  $MV_{best} = MV_{(0,0)}$ 
BEGIN
  Step 1:
    for each boundary level  $l$  where  $0 \leq l \leq l'$ 
      1) Calculate and compare  $BV^l[n]$  with  $SAD_{min}$ , where  $BV^l[n]$  is the boundary value at the  $n^{th}$ 
         checking point in a checking list consisting of all candidate blocks for further processing
      2) if  $BV^l[n] \geq SAD_{min}$ , the checking point is eliminated and removed from the checking list
      3) when the last checking point in the checking list at Level  $l$  is evaluated, set  $l++$  and goto 1).
    end for
  Step 2: Find the first  $M$  points producing the smallest  $BV^{l'}$ 
  Step 3:
    for each candidate checking point with  $m \in [0, M-1]$ 
      Calculate  $SAD[m]$  and find the minimum  $SAD$  and the corresponding  $MV_{best}$ 
    end for
END

```

From the experimental results tabulated in Table 4.6, we can see that MGEA is 1.45-9.22 times faster than GEA in terms of number of operations while giving the same prediction accuracy as shown in Table 4.8.

4.4 Multilevel Adaptive Refining Search (MARS)

To further exploit the level-by-level boundary values, which increasingly approximate the true SAD , we proposed a scheme, termed as Multilevel Adaptive Refining Search (MARS), which is an effective and efficient combination of fast search and fast matching. The MARS scheme can produce a very high speedup gain and simultaneously prevent

Figure 4.1: The illustration of the MARS applying different search strategies at different levels from Level 0 to Level 4 for block size of 16×16

any significant degradation of video quality as generally produced in most fast search algorithms.

4.4.1 Proposed MARS Framework

As we point out above, the level-by-level boundary values are approximates to the true *SAD* with increasing accuracy. Therefore, they can be used to predict the most possible regions for the best match. Considering the boundary values may reveal different levels of information for different video contents, the methodology exploiting such information needs to be adaptive to the image contents and different types of motions contained in various scenes. In the proposed MARS algorithm, we attempt to utilize this information in a coarse-to-fine way. At lower levels, we provide a search strategy that attempts not to exclude any possible best match with coarse information of *SAD* values. Gradually, a strategy based on more accurate estimation of *SAD* values can be biased to fast search.

Considering the block size of $N = 16$, we incorporate the idea into the 5-level multilevel

successive elimination algorithm (MSEA) [12] to develop a “lossy” but high-accuracy fast motion method combining fast search and fast matching. In the multilevel elimination hierarchy, boundary values can be considered as approximated *SAD* values for all the candidates in the MSW. In view of the approximation accuracy in different levels, we apply different strategies accordingly from Level 0 to Level 4 in our proposed MARS framework as shown in Fig. 4.1.

1) Level 0 ($l = 0$): Full Search

Level 0 is the first boundary level in the hierarchy thus no previous boundary value information is available. Hence, a full search with maximum displacement $\mathcal{D}_0$ is naturally applied to evaluate the boundary values at Level 0 for all the checking points.

2) Level 1 ($l = 1$): Full Search

At Level 1, full search is also applied in order to avoid the inaccurate prediction with very coarse information of the approximated *SAD* values using the boundary values at Level 0. Since prediction error can be propagated from level to level within the hierarchy, any error by missing any possible best match candidate at beginning levels will result in an unacceptable result at the end. On the other hand, establishing lower boundary levels requires a very small amount of computations, especially at Level 0 & 1. Thus it's safe and also economical to conduct an exhaustive search at the beginning Level 0 & 1. Note that elimination is performed at each level so that all the points whose boundary values are larger than the current SAD_{min} will not be passed to the next level.

3) Level 2 ($l = 2$): Partial full search around some selected checking points

Despite of being finer than the information from Level 0, the information from Level 1 with partitioned subblock size of 8×8 is still not accurate enough. In consideration of this, we conduct the motion search at Level 2 by using partial full search, i.e., a number of small square regions located at different points in

the MSW are selected and a full search is carried out within each square region to evaluate boundary values at Level 2 on 4×4 partitioned subblocks.

The search center of each square region is determined by the boundary values available from Level 1. Only some checking points with small boundary values are selected as the centers. The selection of such center points is determined by Equ. (4.4.1), where $BV^{l-1}[n]$ is the boundary value at the n^{th} checking point, BV_{min}^{l-1} is the minimum boundary value at Level $l - 1$, Z_{l-1} is the total number of checking points remaining after Level $l - 1$ and MA refers to motion activity value. The MA value is calculated based on neighboring motion vectors as shown in Equ. (4.4.2), where $MV_{nei}[j]$ is the j^{th} neighboring motion vector and N_{nei} is the total number of valid neighboring motion vectors around the current block in both temporal and spatial domains. Equ. (4.4.1) means that a checking point is selected as a search center when its boundary value does not exceed MA multiples of the minimum boundary value BV_{min}^{l-1} . Then a full search is conducted within a small square region centered at this point at Level l (here $l = 2$). The maximum displacement ($disp[m]$) of each square region centered at the m^{th} selected point is obtained using Equ. (4.4.3). Since the checking point with the minimum boundary value BV_{min}^{l-1} has the highest possibility to be the best match, a larger size of search region is assigned around this point. In Equ. (4.4.3), the square region centered at the point of BV_{min}^{l-1} has the largest search range of 9×9 and others have the same search range of 3×3 .

From Equ. (4.4.1), we know that the number of small square regions and the corresponding search centers is adaptively determined by boundary values and the estimated motion activity of the blocks. For blocks containing large and complex motions, a relatively large number of search regions are generally used in the motion search, while only a few square regions are used for blocks containing slow motions.

$$BV^{l-1}[n] \leq MA \times BV_{min}^{l-1}, \quad n \in [0, Z_{l-1}] \quad (4.4.1)$$

$$MA = \frac{\max(MV_{nei}^x[j], MV_{nei}^y[j])}{2}, \quad j \in [0, N_{nei} - 1] \quad (4.4.2)$$

$$disp[m] = \begin{cases} 4 & BV^{l-1}[m] = BV_{min}^{l-1} \\ 1 & else \end{cases} \quad (4.4.3)$$

4) Level 3 ($l = 3$): Fast hexagonal search around selected points

At Level 3, blocks are further partitioned into subblocks of size 2×2 , thus boundary values $BV^l(l = 3)$ are very close to the true *SAD* values. Besides, although most of searching points may be rejected from Level 0 to Level 2, there may still be a relatively large number of points remaining in some cases, especially for blocks containing large and complex motions. Considering these two factors, we apply a hexagonal fast search [8], primarily due to its significant performance for medium to large motions, at Level 3 to speedup the process further. Several hexagonal searches are separately conducted in different square regions that are centered at selected checking points. In other words, multiple hexagonal searches are conducted at Level 3.

Each square region is located at a checking point whose boundary value at Level 2 needs to satisfy Equ. (4.4.4), where $2^l = 8$ with $l = 3$. That means a checking point will be used as a search center of a small square region when its boundary value at Level 2 is less than or equal to $\frac{9}{8}$ times of the BV_{min}^{l-1} , i.e., $BV^{l-1}[n] \leq \frac{9}{8}BV_{min}^{l-1}$. Thus, the number of small square regions as well as the corresponding locations are fully determined by boundary values. In other words, the search is guided by boundary value information from Level 2. As mentioned above, the checking point with the minimum boundary value BV_{min}^{l-1} is most likely to be the best match, especially when l increases, thus we set a large search area of 9×9 for a hexagonal search around the point to ensure a high prediction accuracy. Meanwhile, the search range for the rest small regions is all set to 5×5 as indicated in Equ. (4.4.5).

A conventional fast search always suffers from being trapped into local minimum

due to a poor initial starting point. In the proposed search strategy at Level 3, this weakness can be substantially amended by using boundary value information at previous level for the choice of starting points together with possibly multiple concurrent searches.

$$2^l \times (BV^{l-1}[n] - BV_{\min}^{l-1}) \leq BV_{\min}^{l-1} \quad (4.4.4)$$

$$disp[m] = \begin{cases} 4 & BV^{l-1}[m] = BV_{\min}^{l-1} \\ 2 & \text{else} \end{cases} \quad (4.4.5)$$

4) Level 4 ($l = 4$): SAD evaluation on a few selected points

The partitioned subblock at last level is of size 2×2 , that is, the block of size 16×16 is divided into 64 subblocks of size 2×2 . Thus, the boundary values calculated based on these 64 subblocks of size 2×2 will be very close to the corresponding *SAD* values. In this case, we conduct the fast search at Level 4 by only evaluating a small number of checking points selected according to their boundary values at Level 3, rather than using a full search within small regions at Level 2 or hexagonal search at Level 3.

We choose the checking points to be evaluated at Level 4 according to Equ. (4.4.4), where $2^l = 16$ with $l = 4$. In other words, the *SAD* values at Level 4 are calculated only at those checking points whose boundary values at Level 3 are less than or equal to $\frac{17}{16}BV_{\min}^{l-1}$, i.e., $BV^{l-1}[n] \leq \frac{17}{16}BV_{\min}^{l-1}$. It is noted that the criterion for selecting checking points as search centers becomes more and more strict with increasing level l ($\frac{9}{8}$ at Level 3 and $\frac{17}{16}$ at Level 4), since the boundary values at higher level l provides more accurate approximates to the true *SAD* values. Only the point with the smallest boundary value and the points with a little larger boundary values are selected. In this way, the number of selected checking points for further processing is not predefined by a fixed number but adaptively determined by boundary values.

From the above 5 steps corresponding to 5 levels of boundary value evaluation, we can see that the acceleration of motion estimation is due to several factors such as thresholding for selection of checking points in Level 2 to 4, elimination of checking points level by level and refining the search based on the refined boundary values with increasing level. In the meantime, prediction accuracy is maintained as high as possible by applying different search strategies in different levels in view of the approximation accuracy of boundary values at different levels. Therefore we term the proposed method as Multilevel Adaptive Refining Search (MARS), also for mnemonic purpose.

Due to the elimination process, the value of the current SAD_{min} has significant effects on the speed. The earlier the true SAD_{min} is found, the earlier a checking point may be eliminated before its SAD calculation. Therefore, besides of applying different search strategies, we also attempt to refresh the current SAD_{min} from Level 2 to 4 so that more checking points may be rejected at lower levels. The SAD is calculated at the point with BV_{min}^{l-1} at each level and then compared with the current SAD_{min} for updating. All points remaining from previous level are re-examined based on the updated SAD_{min} and their available boundary values at previous level to further reduce the number of checking points at higher levels. In this way, some more computations can be saved without affecting the prediction accuracy.

4.4.2 Readjusting Initial Search Range in MARS

Conventionally, a fixed search range (e.g., 31×31) is usually used in most existing BMAs. However, since video sequences contain widely varying contents and motions, a fixed search range is obviously inefficient not only in speed but also in coding efficiency. In this subsection, neighboring correlations in temporal and spatial domains are used to accordingly readjust the search range $(2\mathcal{D}_0 + 1) \times (2\mathcal{D}_0 + 1)$ for each block in the proposed MARS framework.

Here we choose four spatially neighboring blocks of left (CN_1), up-left (CN_2), up

(CN_3) and up-right (CN_4) and one temporally neighboring block of RN_0 as candidate neighboring blocks according to the priority order experimentally determined in [51]. The layout of the selected neighboring blocks is illustrated in Fig. 2.4 (b).

The information from neighboring correlations includes two types: motion information from the neighboring motion vectors and spatial information such as texture features and similarities between the current block and its neighboring blocks. In our algorithm, only the former is utilized in the following subsection to readjust the search range for each current block.

Utilizing Motion Information

As mentioned in Chapter 2, motion vectors of neighboring blocks may be highly correlated to each other in spatial and temporal domains, where objects may spatially cover a few blocks and motions are generally temporally continuous. Motivated from these observations, we attempt to utilize the neighboring motion vectors to predict a search range most suitable for the current block so that a very high prediction accuracy is ensured while achieving a significant reduction of computational complexity. In consideration of center-biased property of motion vector distribution, $MV_{(0,0)}$ is used as the center of MSW.

A brief description about this scheme is presented in Table 4.4, where $MV_{nei}[j]$ is the motion vector of the j^{th} neighboring block. The magnitudes of MV_{nei}^x and MV_{nei}^y components are used to determine a suitable search range for the current block. We first set the readjusted maximum displacement $\mathcal{D}_1$ to the double of value $\max(MV_{nei}^x[j], MV_{nei}^y[j])$. In order to maintain as high accuracy as possible, $\mathcal{D}_1$ is refined to 2 if $2 \times \mathcal{D}_1 \leq 1$, which means the smallest search range in our scheme is 5×5 . This ensures that any incorrect prediction from zero motion vector in the neighboring motion vectors will be substantially alleviated by using the medium search window of 5×5 . Besides, we also set $\mathcal{D}_1$ to $\mathcal{D}_0$ so that no additional checking points will be introduced as compared with the FSA

Table 4.3: Prediction accuracy (%) and speedup gain using motion information obtained from neighboring blocks based on the scheme in Table 4.4

	Carphone	Coastguard	M&D	Silent	Salesman	Dancer
%	99.364	99.949	99.232	99.586	100.00	99.854
Speedup	3.023	4.435	4.235	4.102	5.482	1.055
	Foreman	Stefan	Table-tennis	Bus	Paris	Waterfall
%	99.480	99.157	98.535	99.705	99.710	100.00
Speedup	2.812	1.592	2.074	1.155	6.342	9.891

centered at $MV_{(0,0)}$ with the maximum displacement of $\mathcal{D}_0$. Note that the readjustment process in Table 4.4 is performed only when not less than 2 valid neighboring motion vectors ($N_{nei} \geq 2$) are available. If $N_{nei} < 2$, the search range remains unchanged, i.e., $\mathcal{D}_1 = \mathcal{D}_0$.

From the experimental results tabulated in Table 4.3, which are obtained by conducting a full search with a readjusted search range as in Table 4.4, we can see that the prediction accuracy is always maintained very high, in a range of 98.535%-100.0%, for all video sequences, while a considerable improvement in speedup gain is achieved.

Table 4.4: Procedure of utilizing priori motion information to adaptively determine the search range, where $\mathcal{D}_0$ is the original maximum displacement, $\mathcal{D}_1$ is the refined maximum displacement and N_{nei} is the number of the valid neighboring motion vectors.

Input: A current block and its neighboring motion vectors MV_{nei} The initial maximum displacement $\mathcal{D}_0$
Output: Readjusted maximum displacement $\mathcal{D}_1$
Preprocessing: Determine N_{nei} , where only valid neighboring motion vectors are taken into account
BEGIN
Step 1: Calculate $\mathcal{D}_1$:
$\mathcal{D}_1 = 2 \times \max(MV_{nei}^x[j], MV_{nei}^y[j]), j \in [0, N_{nei} - 1]$
Step 2: Refine $\mathcal{D}_1$
if $N_{nei} \geq 2, \mathcal{D}_1 = \begin{cases} \mathcal{D}_0 & 2 \times \mathcal{D}_1 > \mathcal{D}_0 \\ 2 & 2 \times \mathcal{D}_1 \leq 1 \\ \mathcal{D}_1 & else \end{cases}$
else $\mathcal{D}_1 = \mathcal{D}_0$
END

4.4.3 Comparative Analysis

A comparative analysis on the proposed MARS scheme against fast lossless algorithms MSEA, modified lossy algorithm MGEA and conventional lossy fast search algorithms DS and HEX, is outlined in terms of computation efficiency in the following.

- 1) Unlike the MSEA or our proposed FGSE in Chapter 3, which performs the motion estimation by finishing the evaluation of a checking point before evaluating the next checking point, the MARS scheme performs the motion vector search level by level as a whole. The characteristics of all checking points at lower boundary levels can guide the search at higher levels, thus making the interaction between fast search and fast matching possible.
- 2) As compared to MGEA, boundary value information from multiple levels is fully utilized in our proposed MARS scheme. For $N = 16$, information from each level is progressively exploited in MARS, among which only information at Level 2 is used in MGEA. Also, the way to utilize the information in our proposed MARS is substantially different from that in MGEA. In the MARS, a more suitable and effective exploitation of boundary value information is carried out. Furthermore, the proposed MARS has the capability to produce adaptive number of checking points to be evaluated at each level depending on level-by-level boundary values and motion activity of the current blocks. In GEA/MGEA, only the fixed number of first M points with the smallest boundary values at Level l' are selected. The choice of l' and M is a very difficult problem. In this way, the dilemma of the choice of level l' and number M can be smartly solved in our MARS.
- 3) The conventional fast search algorithms, such as HEX and DS, always suffer relatively large degradations of prediction accuracy since many real video scenes do not follow the error surface uni-modality assumption. In contrast, MARS can conduct a more intelligent search strategy based on the prior information on the SAD values. That means the proposed MARS can be regarded as a MAP-based

estimation, where priori and posteriori information are applied concurrently to conduct the motion estimation.

- 4) In a conventional combination approach, fast search and fast matching are in fact performed independently. That means it is done in a way like “ $1 + 1 = 2$ ”. In this sense, the effectiveness of the combination can not be fully explored. In our MARS, this problem is well solved. The proposed MARS algorithm is actually an effective marriage of fast search and fast matching resulting in the effect of “ $1 + 1 > 2$ ”, where different fast search strategies are incorporated into the multilevel elimination framework to exploit the interplay of fast search and fast matching. In this way, fast search is progressively directed by information of boundary values level by level, which carry the partial information of *SAD* values.
- 5) The performance of our MARS scheme in terms of prediction accuracy and speed can be adjusted by changing some parameters such as motion activity *MA* and applying different search strategies at different levels. In this sense, the proposed MARS can also be regarded as a scalable motion estimation method.

4.4.4 Procedure of MARS Algorithm

The proposed MARS algorithm can be summarized using the *C* pseudocode in Table 4.5.

From this procedure, we can see that for each block, different search strategy is applied at each level accordingly in a way from finer to coarser with coarser-to-finer boundary value information obtained level by level. Such approach ensures a better trade-off point in prediction accuracy and speed. In the beginning levels, full search or nearly full search is used to guarantee the prediction accuracy. With more information available in higher levels, fast search can be applied to speed up the process without comprising the prediction accuracy.

Table 4.5: Procedure of Multilevel Adaptive Refining Search (MARS)

<p>Algorithm: Multilevel Adaptive Refining Search (MARS) Input: A current block, its candidate blocks within a predefined MSW The initial maximum displacement $\mathcal{D}_0$ Output: Best motion vector - MV_{best} Preprocessing: 1) Calculating sum norms 2) Initialize SAD and MV_{best}, set $SAD_{min} = SAD_{MV(0,0)}$</p> <p>BEGIN</p> <p>Step 1 (Refining): Obtain the refined maximum displacement $\mathcal{D}_1$ following the refinement process in Table 4.4</p> <p>Step 2: Calculate the Motion Activity (MA) of the current block based on the neighboring motion information as in Equ. (4.4.2), i.e., $MA = \frac{\max(MV_{nei}^x[j], MV_{nei}^y[j])}{2}$</p> <p>Step 3 (Thresholding & Eliminating): for $l = 0; l++; l \leq 4$ $l = 0 \& 1$: (Level 0 & 1) Conduct an exhaustive search with a maximum displacement of $\mathcal{D}_1$ $l = 2$: (Level 2) L2:1) Obtain the minimum boundary value $BV_{min}^{l-1} = \min\{BV^{l-1}[n]\}$ with $n \in [0, Z_{l-1} - 1]$ and $n = n'$ for BV_{min}^{l-1}, where Z_{l-1} is the number of checking points remaining after Level $l - 1$, i.e., Level 1 with $l = 2$ L2:2) Calculate $SAD[n']$ at the point producing BV_{min}^{l-1}. If $SAD[n'] < SAD_{min}$, set $SAD_{min} = SAD[n']$, $MV_{best} = MV_{l-1}[n']$ and re-examine boundary values $BV^{l-1}[n]$ for $n \in [1, Z_{l-1} - 1]$ L2:3) Conduct Full Search within small square regions to evaluate the checking points remaining after Level 1. The search center for each small region is at the point whose boundary value satisfies $BV^{l-1}[n] \leq MA \times BV_{min}^{l-1}$ with search range of $disp[n] = \begin{cases} 4 & BV^{l-1}[n] = BV_{min}^{l-1} \\ 1 & \text{else} \end{cases}, n \in [0, Z_{l-1} - 1]$ $l = 3$: (Level 3) L3:1) Repeat L2:1) and L2:2), then goto L3:2) L3:2) Conduct Hexagonal Search within small square regions to evaluate checking points remaining after Level 2. The search centers are at points whose boundary values satisfy $2^l \times (BV^{l-1}[n] - BV_{min}^{l-1}) \leq BV_{min}^{l-1}$ and the corresponding search range is $disp[n] = \begin{cases} 4 & BV^{l-1}[n] = BV_{min}^{l-1} \\ 2 & \text{else} \end{cases}, n \in [0, Z_{l-1} - 1]$ $l = 4$: (Level 4) L4:1) Repeat L2:1) and L2:2), then goto L4:2) L4:2) Calculate SAD values of checking points whose boundary values at Level 3 satisfy $2^l \times (BV^{l-1}[n] - BV_{min}^{l-1}) \leq BV_{min}^{l-1}, n \in [0, Z_{l-1} - 1]$ L4:3) Set $MV_{best} = MV_{l-1}[n]$, when $SAD_{min} = SAD[n]$ end for END</p>
--

4.5 Experimental Results

The above-described algorithms including MSEA, GEA, multilevel GEA (MGEA), diamond search (DS) and Hexagon search (HEX) algorithms and our MARS were sim-

Table 4.6: Speedup gain of different algorithms compared to FSA (± 15). We define the speedup gain as $Gain = \frac{\beta}{\alpha}$, where β is the overall computational cost for FSA and α is the overall computational costs for any other algorithms in terms of number of operations simulated in the experiments.

	Carphone	Coastguard	M&D	Silent	Salesman	Dancer
MSEA	32.489	24.554	24.945	88.965	106.994	3.678
GEA	10.806	10.806	10.806	10.806	10.806	10.948
MGEA	51.322	22.622	59.839	86.535	99.639	15.881
MARS	102.579	97.024	136.827	166.522	226.090	19.718
DS	66.006	68.040	72.601	72.609	75.684	35.437
HEX	93.538	93.319	98.794	98.860	101.886	53.138
	Foreman	Stefan	Table-tennis	Bus	Paris	Waterfall
MSEA	18.910	15.690	3.571	11.524	65.835	67.223
GEA	10.948	10.948	10.948	10.948	10.948	10.948
MGEA	29.818	18.462	14.776	14.129	60.016	57.504
MARS	61.776	40.626	38.344	29.185	144.590	180.709
DS	57.463	52.441	55.040	45.295	73.471	78.156
HEX	82.340	74.688	78.431	65.799	100.122	105.334

Table 4.7: Average MAD per pixel (± 15)

	Carphone	Coastguard	M&D	Silent	Salesman	Dancer
MSEA	2.412	4.300	1.787	1.543	1.217	2.702
GEA	2.416	4.308	1.795	1.551	1.218	2.765
MGEA	2.416	4.308	1.795	1.551	1.218	2.765
MARS	2.422	4.319	1.800	1.560	1.218	2.749
DS	2.441	4.382	1.801	1.563	1.219	2.961
HEX	2.680	4.578	1.843	1.596	1.242	3.134
	Foreman	Stefan	Table-tennis	Bus	Paris	Waterfall
MSEA	2.835	6.627	6.480	7.454	2.500	2.952
GEA	2.843	6.691	6.553	7.541	2.511	2.952
MGEA	2.843	6.691	6.553	7.541	2.511	2.952
MARS	2.855	6.718	6.612	7.612	2.525	2.953
DS	2.938	8.879	6.978	10.756	2.531	2.952
HEX	3.383	9.250	7.578	11.355	2.647	3.007

ulated on 12 video sequences with two different formats. These sequences provide a variety of motions ranging from slow motion (“Carphone”, “Silent”), medium to fast motions (“Coastguard”, “Dancer”), camera panning (“Stefan”) and zooming (“Table-tennis”). The luminance components of the first 100 frames in these video sequences are used in the simulation. As usual, the sum of absolute difference (*SAD*) is used as the distortion measure. The maximum displacement in the search window is ± 15 pixels in both horizontal and vertical directions for 16×16 block size. The coding pattern is

Table 4.8: Prediction accuracy for each algorithm, which is defined as the percentage of the best matches found in each algorithm as compared with the FSA (± 15)

	Carphone	Coastguard	M&D	Silent	Salesman	Dancer
MSEA	100.00	100.00	100.00	100.00	100.00	100.00
GEA	96.384	99.394	94.293	99.040	99.960	73.273
MGEA	96.384	99.394	94.293	99.040	99.960	73.273
MARS	94.303	99.091	93.586	99.071	99.838	70.568
DS	93.505	98.889	94.455	98.990	99.939	48.965
HEX	77.182	94.677	89.505	95.970	98.778	31.437
	Foreman	Stefan	Table-tennis	Bus	Paris	Waterfall
MSEA	100.00	100.00	100.00	100.00	100.00	100.00
GEA	95.331	90.556	88.061	94.773	98.980	99.975
MGEA	95.331	90.556	88.061	94.773	98.980	99.975
MARS	92.828	93.604	83.520	93.003	98.513	99.798
DS	87.970	80.634	77.942	66.318	98.745	100.00
HEX	55.404	74.088	57.497	52.790	93.326	95.121

IPP.....,

It's noted that all overhead computations involved in all the comparisons and additions required have been included in the overall computation costs of the algorithms. Moreover, multiplications here are converted to shift operations to avoid high computing power. Actually, high-standard C compiler nowadays usually carry on this conversion automatically in view of computing efficiency.

Table 4.6-4.8 tabulate all the simulation results for our MARS algorithm as compared to the lossless algorithms MSEA, fast but nearly lossless algorithms GEA, MGEA and fast lossy algorithms DS and HEX.

Table 4.6 compares different algorithms in speedup gain calculated based on the overall computations of the conventional FSA, which include addition, subtraction, absolution, comparison and shift operations. According to these results, our MARS algorithm can achieve a significant reduction in computation compared with the fast algorithms of MSEA, GEA and MGEA. For instance, for "Coastguard" sequence, our MARS algorithm is 4.29 faster as compared to MGEA. For "Salseman" sequence, our MARS is up to 226.09 times faster than FSA. The speedup gain of our MARS over MGEA is in the range of 124.16%-428.89% for all video sequences. More importantly, such

significant improvements are archived not in the cost of prediction accuracy as shown in Table 4.7-4.8. MARS produces an increase in the average *MAD* per pixel compared to MGEA in a negligible range of -0.016-0.071. The values of prediction accuracy tabulated in Table 4.8 have demonstrated the similar results in comparison of our MARS with MGEA. In spite of very sequence-dependent, the accuracy difference between both algorithms is constrained to a range of -4.54%-3.05%.

However, compared with those lossy fast algorithms DS and HEX, the MARS cannot always produce much larger speedup gains for all video sequences, especially for those sequences containing large and complex motions. In contrast, fast search algorithms such as DS and HEX cannot provide such intelligent capacity and are easily trapped into local minimum when a complex error surface is present. For example, for “Bus” sequence, the prediction accuracy is only 66.32% for DS, which is much lower than 93.00% for MARS and thus leading to a large increase in *MAD*, i.e., 10.756 in DS and 11.355 in HEX, which is much larger than 7.612 in MARS as shown in Table 4.7. In other words, those conventional fast lossy search algorithms may produce high speedup gain by sacrificing video quality. This situation can generally be avoided in MARS because information obtained from lower levels can give enough support to the fast search conducted at higher levels. On the other hand, MARS offers higher speedup gain than DS and HEX for video sequences containing small and regular motions. For instance, the speedup gain from MARS, DS and HEX are 226.09, 75.68 and 101.89, respectively, for “Salesman” sequence, while sharing a very similar prediction accuracy and a close *MAD* value.

From the results in Table 4.6-4.8, we can see that the proposed MARS can always find a better trade-off point in prediction accuracy and speedup gain for all video sequences. In this way, a better complexity-distortion performance has been obtained.

4.6 Conclusion

A new framework, Multilevel Adaptive Refining Search (MARS) has been proposed, where fast search and fast matching interact with each other to achieve a progressive motion estimation. Our MARS can be regarded as a MAP-based progressive estimation method, where level-by-level boundary values that contain more and more partial information of the true *SAD* values are used to progressively direct fast search in a multilevel elimination hierarchy. By incorporating different search strategies at different levels accordingly, an effective marriage of fast search and fast matching can be achieved thus resulting in an effect of “ $1 + 1 > 2$ ”. Moreover, spatial/temporal motion correlations are utilized to refine the initial motion search range for each current block in the MARS framework. Simulation results have clearly shown that our proposed MARS algorithm produces substantially higher speedup gain over MSEA, GEA and MGEA (up to 4.29 times faster over MGEA), while achieving considerably better performance than those conventional fast search algorithm DS and HEX. Therefore, better trade-off points in prediction accuracy and speedup gain can be achieved using our MARS.

Finally, we also find another important advantage of our MARS, that is, the MARS is very suitable for computation-aware (CA) motion estimation [48]. When computation power allocated to a block is used up, we can naturally choose the point with the minimum boundary value at the stopping level as the best motion vector, which is expected to be a very good estimation to the true best MV, especially when the stopping level is higher.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Over the last two decades, fast block matching motion estimation has gained tremendous research interests in video coding because of its vital role in all popular video coding standards such as H.261/263/264 and MPEG-1/2/4. However, it's quite computation-intensive if exhaustively evaluating all candidates in a predefined motion search window for each current block. To overcome this problem, a lot of fast block matching algorithms have been developed to minimize the heavy computational requirements while attempting to maintain a high prediction accuracy.

The existing fast block matching algorithms are designed mainly in two directions: reducing the number of checking points (Fast Search) and lowering the computations at matching each checking point (Fast Matching) as shown in Fig. 5.1. However, a degraded quality is expected in the fast search methods while the speedup gain may be very limited in the fast matching methods. In this thesis, based on the intensive research on various existing fast BMAs, we have developed some more efficient algorithms to achieve both speed and prediction accuracy maximally, one is the fast matching method in **Chapter 3**, and another is the effective combination method of fast search and fast

Figure 5.1: Categorization of Existing Fast BMAs

matching in **Chapter 4**.

In **Chapter 3**, a new algorithm called fine granularity successive elimination algorithm (FGSE) has been proposed in the direction of Fast Matching. FGSE can significantly reduce the computational complexity by successively eliminating those non-best candidates based on a mathematical inequality incorporated with sum norms of blocks and subblocks. A sequence of rejection levels is established and used to evaluate each candidate sequentially from the lowest level to the highest level within the predefined motion search window, where image complexity is utilized to determine the order of partitioning larger subblocks into smaller subblocks in the creation of fine-grained boundary levels. Furthermore, correlation information from neighboring candidates are utilized to predict rejection level to further accelerate the rejection process. Since all the search points are examined as in the FSA, FGSE can ensure the optimality of motion vectors. Experimental results have substantially shown that FGSE can contribute a significant computational reduction and produce a much larger speedup gain than those obtained from the well-known SEA and MSEA.

In **Chapter 4**, a new combination scheme of fast search and fast matching, namely

Multilevel Adaptive Refining Search (MARS) has been proposed, where fast search and fast matching interact with each other to achieve a progressively guided motion estimation. Our MARS can be regarded as a maximum a posteriori (MAP)-based estimation method, where priori information of boundary values that are a series of approximates to the true *SAD* values with increasing accuracy are used to progressively guide fast search level by level in a multilevel elimination hierarchy. By incorporating different search strategies at different levels accordingly, an effective marriage of fast search and fast matching can be achieved thus resulting in an effect of “ $1 + 1 > 2$ ” in contrast to the conventional combination approach, which performs like “ $1 + 1 = 2$ ”. Moreover, spatial/temporal correlations are utilized to refine the search range for each current block in the MARS framework so that an adaptive search range can be obtained based on prior information varying from block to block. Simulation results have clearly shown that our proposed MARS algorithm can achieve a high speedup gain not in sacrifice of prediction accuracy and vice versa, i.e., better trade-off points between prediction accuracy and speedup gain can be obtained as compared to fast search and fast matching algorithms.

5.2 Future Work

Despite the satisfactory results mentioned above, many more issues remain to be done in the fast block matching motion estimation. We list some future research directions in the following subsections.

5.2.1 Variable Block Size Motion Estimation

In the existing video coding standards, variable block size has outperformed fixed block size in terms of coding efficiency. In view of this, further research efforts will be put into adapting the proposed algorithms in **Chapter 3** and **Chapter 4** to variable block

size smaller than or equal to 16×16 .

5.2.2 Multiframe Motion Estimation

With the fast development of semiconductor technologies, memories with high capacity and fast access are available at increasingly lower cost. This makes it possible that video coding can be performed using more memories while keeping the process sufficiently fast. Multi-frame motion estimation is also called long-term memory motion search. It has been adopted by many international video coding standards such as H.263+, H.26L, H.264 and MPEG-4. Its basic idea is to extend the temporal motion search range by using multiple decoding frames as reference frames [132]. The long-term memory may cover several seconds of decoded frames at the encoder and decoder. However, apparently, computational complexity is dramatically increased as compared with the conventional single-frame methods. Although some new techniques such as hierarchical search strategy in [132] and subsampling techniques in [133] have been developed to overcome this problem, the computational complexity still remains very high. Hence, the heavy computational load of the multiframe motion estimation remains a big challenge for further research.

5.2.3 Subpixel Motion Estimation

In the real world, the true displacements may be unrelated to the integer pixel sampling grid. In other words, the motion estimation accuracy in the algorithms mentioned above is limited to integer-pixel level. In order to achieve noticeable additional prediction gain, subpixel motion estimation has been adopted in popular video coding standards such as MPEG-4 and H.264 (with $\frac{1}{4}$ pixel accuracy), where it is used as a refinement process for the integer pixel motion search and can provide much higher prediction accuracy. However, it requires a plenty of computations not only in the subpixel search process but also in the pixel interpolation. In [134–136], some novel algorithms at a fractional

pixel basis have been developed, but they may need to be improved further. In our future work, we are going to expand our research from integer pixel to subpixel to further improve the video coding efficiency.

5.2.4 Rate-Distortion-Complexity Fast Motion Estimation

In consideration of the overall performance of a hybrid video coding system in practice, three crucial factors need to be balanced: rate, distortion and complexity. In the conventional rate-distortion theory, bit rates are distributed over typically dependent motion and residual information in order to ensure satisfactory image quality [137]. In other words, only rate and distortion are considered here. However, the speed of the coding system is also very important, especially for real-time applications. In our future work, we propose to develop some fast block matching algorithms to balance the selection of rate, distortion and complexity so that an optimal performance can be achieved for the video coding.

Publications

- [1] Ce Zhu, Wei Song Qi and Wee Ser, "Predictive Fine Granularity Successive Elimination for Fast Optimal Block Matching Motion Estimation", *IEEE Transaction on Image Processing*, Volume 14, Issue 2, Feb. 2005 Page(s):213 - 221.
- [2] Ce Zhu, Wei Song Qi and Wee Ser, "A New Successive Elimination Algorithm For Fast Block Matching In Motion Estimation", *IEEE International Symposium on Circuits and Systems*, 23-26 May 2004, Page(s):III - 733-6, Vol.3, Vancouver.

Bibliography

- [1] P. Pirsch, N. Demassieux, and W. Gehrke, "VLSI architectures for video compression- A survey," *Proc. of the IEEE*, vol. 83, pp. 220–246, Feb. 1995.
- [2] A. M. Telkap, *Digital video processing*. NJ: Prentice Hall, 1995.
- [3] B. Zeng, R. Li, and M. L. Liou, "Optimization of fast block motion estimation algorithms," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 7, pp. 833–844, Dec. 1997.
- [4] R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 4, pp. 438–443, Aug. 1994.
- [5] S. Zhu and K. K. Ma, "A new diamond search algorithm for fast block matching motion estimation," in *ICICS-1997*, vol. 1, pp. 292–296, Sept. 1997.
- [6] J. Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, "A novel unrestricted center-biased diamond search algorithm for block motion estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 8, pp. 369–377, Aug. 1998.
- [7] S. Zhu and K. K. Ma, "A new diamond search algorithm for fast block matching motion estimation," *IEEE Trans. Image Processing*, vol. 9, pp. 287–290, Feb. 2000.
- [8] C. Zhu, X. Lin, and L. P. Chau, "Hexagon-based search pattern for fast block motion estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 12, pp. 349–355, May 2002.
- [9] C. K. Cheung and L. M. Po, "Normalized partial distortion search algorithm for block motion estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 10, pp. 417–422, Apr. 2000.
- [10] W. Li and E. Salari, "Successive elimination algorithm for motion estimation," *IEEE Trans. Image Processing*, vol. 4, pp. 105–107, Jan. 1995.
- [11] C. H. Lee and L. H. Chen, "A fast motion estimation algorithm based on the block sum pyramid," *IEEE Trans. Image Processing*, vol. 6, pp. 1587–1591, Nov. 1997.

- [12] X. Q. Gao, C. J. Duanmu, and C. R. Zou, "A multilevel successive elimination algorithm for block matching motion estimation," *IEEE Trans. Image Processing*, vol. 9, pp. 501–504, Mar. 2000.
- [13] K. Lengwehasarit and A. Ortega, "Probabilistic partial-distance fast matching algorithms for motion estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, pp. 139–152, Feb. 2001.
- [14] T. Koga, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion-compensated interframe coding for video conferencing," in *Proc. NTC81*, pp. G5.3.1–G5.3.5, Nov. 1981.
- [15] L. M. Po and W. C. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 6, pp. 313–317, June 1996.
- [16] L. K. Liu and E. Feig, "A block-based gradient descent search algorithm for block motion estimation in video coding," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 6, pp. 419–423, Aug. 1996.
- [17] J. R. Jain and A. K. Jain, "Displacement measurement and its application in interframe image coding," *IEEE Trans. Commun.*, vol. 29, pp. 1799–1808, Dec. 1981.
- [18] M. Ghanbari, "The cross-search algorithm for motion estimation," *IEEE Trans. Commun.*, vol. 38, pp. 950–953, July 1990.
- [19] L. P. Chau and C. Zhu, "A fast octagon based search algorithm for motion estimation," *Signal Processing*, vol. 83, pp. 671–675, Mar. 2003.
- [20] Y. C. Lim, K. Y. Min, and J. W. Chong, "A pentagonal fast block matching algorithm for motion estimation using adaptive search range," in *ICASSP-03*, vol. 3, pp. 669–672, Apr. 2003.
- [21] Y. Nie and K. K. Ma, "Adaptive rood pattern search for fast block-matching motion estimation," *IEEE Trans. Image Processing*, vol. 11, pp. 1442–1449, Dec. 2002.
- [22] C.-W. Lam, L. M. Po, and C. H. Cheung, "A novel kite-cross-diamond search algorithm for fast block matching motion estimation," in *ISCAS-04*, vol. 3, pp. 729–732, May 2004.
- [23] C. W. Lam, L. M. Po, and C. H. Cheung, "A novel kite-cross-diamond search algorithm for fast video coding and videoconferencing applications," in *ICASSP-04*, vol. 3, pp. 365–368, May 2004.
- [24] L. W. Lee, J. F. Wang, J. Y. Lee, and J. D. Shie, "Dynamic search-window adjustment and interlaced search for block-matching algorithm," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 3, pp. 85–87, Feb. 1993.
- [25] H. M. Jong, L. G. Chen, and T. D. Chiueh, "Accuracy improvement and cost reduction of 3-step search block matching algorithm for video coding," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 4, pp. 88–90, Feb. 1994.

- [26] P. Lakamsani, B. Zeng, and M. Liou, "An enhanced three step search motion estimation method and its VLSI architecture," in *ISCAS-96*, vol. 2, pp. 754 – 757, May 1996.
- [27] J. N. Kim and T. S. Choi, "A fast three-step search algorithm with minimum checking points using unimodal error surface assumption," *IEEE Trans. Consumer Electronics*, vol. 44, pp. 638–648, Aug. 1998.
- [28] H. Nisar and T. S. Choi, "An advanced center biased three step search algorithm for motion estimation," in *ICME-00*, vol. 1, pp. 95–98, Aug. 2000.
- [29] X. Jing and L. P. Chau, "An efficient three-step search algorithm for block motion estimation," *IEEE Trans. Multimedia*, vol. 6, pp. 435–438, June 2004.
- [30] O. T. C. Chen, "Motion estimation using a one-dimensional gradient descent search," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 10, pp. 608–616, June 2000.
- [31] L. G. Chen, W. T. Chen, Y. S. Jehng, and T. D. Chiuch, "An efficient parallel motion estimation algorithm for digital image processing," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 1, pp. 378–385, Dec. 1991.
- [32] A. Puri, H. M. Hang, and D. L. Schilling, "An efficient block matching algorithm for motion-compensated coding," in *ICASSP-87*, pp. 1063–1066, Apr. 1987.
- [33] L. M. Po and C. K. Cheung, "A new center-biased orthogonal search algorithm for fast block motion estimation," in *TENCON-96*, vol. 2, pp. 874–877, Nov. 1996.
- [34] X. Q. Binh and Y. P. Tan, "Adaptive dual-cross search algorithm for block-matching motion estimation," *IEEE Trans. Consumer Electronics*, vol. 50, pp. 766–775, May 2004.
- [35] J. Lu and M. L. Liou, "A simple and efficient search algorithm for block-matching motion estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 7, pp. 429–433, Apr. 1997.
- [36] F. H. Cheng and S. N. Sun, "New fast and efficient two-step search algorithm for block motion estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 9, pp. 977–983, Oct. 1999.
- [37] V. Christopoulos and J. Cornelis, "A center-biased adaptive search algorithm for block motion estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 10, pp. 423–426, Apr. 2000.
- [38] C. Zhu, X. Lin, L. P. Chau, H. A. Ang, and C. Y. Ong, "An optimized diamond search algorithm for block motion estimation," in *ISCAS-02*, vol. 2, pp. 488–491, May 2002.
- [39] C. H. Cheung and L. M. Po, "A novel cross-diamond search algorithm for fast block motion estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 12, pp. 1168–1177, Dec. 2002.

- [40] C. H. Cheung and L. M. Po, "A novel small-cross-diamond search algorithm for fast video coding and videoconferencing applications," in *ICIP-02*, vol. 1, pp. 681–684, Sept. 2002.
- [41] C. H. Cheung and L. M. Po, "A novel rood-diamond search algorithm for fast block motion estimation," in *ICASSP-02*, vol. 4, pp. 3397–3400, May 2002.
- [42] C. Zhu, X. Lin, L. P. Chau, H. A. Ang, and C. Y. Ong, "Efficient inner search for faster diamond search," *Signal Processing*, vol. 84, pp. 527–533, Mar. 2004.
- [43] C. J. Kuo, C. H. Yeh, and S. F. Odeh, "Polynomial search algorithms for motion estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 10, pp. 813–818, Aug. 2000.
- [44] K. K. Ma and G. Qiu, "Unequal-arm adaptive rood pattern search for fast block-matching motion estimation in the JVT/H. 26L," in *ICIP-03*, vol. 1, pp. 901–904, Sept. 2003.
- [45] K. K. Ma and G. Qiu, "An improved adaptive rood pattern search for fast block-matching motion estimation in JVT/H. 26L," in *ISCAS-03*, vol. 2, pp. 708–711, May 2003.
- [46] C. Zhu, X. Lin, and L. P. Chau, "An enhanced hexagonal search algorithm for block motion estimation," in *ISCAS-03*, vol. 2, pp. 392–395, May 2003.
- [47] C. Zhu, X. Lin, L. Chau, and L. M. Po, "Enhanced hexagonal search for fast block motion estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 14, pp. 1210–1214, Oct. 2004.
- [48] P. L. Tai, S. Y. Huang, C. T. Liu, and J. S. Wang, "Computation-aware scheme for software-based block motion estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 13, pp. 901–913, Sept. 2003.
- [49] P. L. Tai, C. T. Liu, and J. S. Wang, "Complexity-adaptive search algorithm for block motion estimation," in *ICIP-01*, vol. 2, pp. 969–972, Oct. 2001.
- [50] P. L. Tai, C. T. Liu, S. Y. Huang, and J. S. Wang, "Flexible fast block matching algorithm design based on complexity-distortion optimization," in *ICME-01*, pp. 689–692, Aug. 2001.
- [51] J. C. Tsai, C. H. Hsieh, S. K. Weng, and M. F. Lai, "Block-matching motion estimation using correlation search algorithm," *Signal Processing: Image Commun.*, vol. 13, pp. 119–133, 1998.
- [52] K. C. Lai and S. C. Wong, "A fast motion estimation using a three-dimensional reference motion vector," in *ICASSP-02*, vol. 4, pp. 3429–3432, May 2002.
- [53] Y. Li and S. Orintara, "A novel adaptive multi-mode search algorithm for fast block-matching motion estimation," in *ISCAS-04*, vol. 3, pp. 977–980, May 2004.
- [54] L. Luo, C. Zou, X. Gao, and Z. He, "A new prediction search algorithm for block motion estimation in video coding," *IEEE Trans. Consumer Electronics*, vol. 43, pp. 56–61, Feb. 1997.

- [55] D. S. Turaga and T. Chen, "Estimation and mode decision for spatially correlated motion sequences," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, pp. 1098–1107, Oct. 2001.
- [56] J. B. Xu, L. M. Po, and C. K. Cheung, "Adaptive motion tracking block matching algorithms for video coding," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 9, pp. 1025–1029, Oct. 1999.
- [57] K. L. Chung and L. C. Chang, "A new predictive search area approach for fast block motion estimation," *IEEE Trans. Image Processing*, vol. 12, pp. 648–652, June 2003.
- [58] J. M. Jou, P. Y. Chen, and J. M. Sun, "The gray prediction search algorithm for block motion estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 9, pp. 843–848, Sept. 1999.
- [59] C. H. Hsieh, P. C. Lu, and J. S. Shyn, "Motion estimation using interblock correlation," in *ISCAS-91*, vol. 2, pp. 995–998, May 1991.
- [60] H. Zhang and Z. He, "A new motion estimation algorithm," in *PACRIM-91*, vol. 2, pp. 619–622, May 1991.
- [61] D. W. Kim, J. S. Choi, and J. T. Kim, "Adaptive motion estimation based on spatio-temporal correlation," *Signal Processing: Image Commun.*, vol. 13, pp. 161–170, 1998.
- [62] J. Y. Nam, J. S. Seo, J. S. Kwak, M. H. Lee, and Y. H. Ha, "New fast-search algorithm for block matching motion estimation using temporal and spatial correlation of motion vector," *IEEE Trans. Consumer Electronics*, vol. 46, pp. 934–942, Nov. 2000.
- [63] M. G. Liu and C. H. Hou, "A fast block-matching motion estimation algorithm based on spatial-temporal motion vector correlation," in *ISIMP-01*, pp. 498–501, May 2001.
- [64] K. Ramkishor and S. Krishna, "Spatio-temporal correlation based fast motion estimation algorithm for MPEG-2," in *IEEE Conference on Signals, Systems and Computers*, vol. 1, pp. 220–224, Nov. 2001.
- [65] J. H. Lim and H. W. Choi, "Adaptive motion estimation algorithm using spatial and temporal correlation," in *PACRIM. 2001*, vol. 2, pp. 473–476, 2001.
- [66] H. Yoon and G. Lee, "Motion estimation based on spatio-temporal correlations," in *ICIP-03*, vol. 2, pp. 359–362, Sept. 2003.
- [67] A. M. Tourapis, O. C. Au, and M. L. Liou, "Highly efficient predictive zonal algorithms for fast block-matching motion estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 12, pp. 934–947, Oct. 2002.

- [68] A. Chimienti, C. Ferraris, and D. Pau, "A complexity-bounded motion estimation algorithm," *IEEE Trans. Image Processing*, vol. 11, pp. 387–392, Apr. 2002.
- [69] K. R. Namuduri, "Motion estimation using spatio-temporal contextual information," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 14, pp. 1111–1115, Aug. 2004.
- [70] M. Bierling, "Displacement estimation by hierarchical block matching," in *Proc. SPIE Visual Communications and Image Processing*, vol. 1001, 1988.
- [71] F. Dufaux and M. Kunt, "Multigrid block matching motion estimation with an adaptive local mesh refinement," in *Proc. SPIE Visual Communications and Image Processing*, vol. 1818, 1992.
- [72] F. Dufaux and F. Moscheni, "Motion estimation techniques for digital TV: A review and a new contribution," *Proceedings of the IEEE*, vol. 83, pp. 858–876, June 1995.
- [73] X. Xia and Y. Q. Shi, "A thresholding hierarchical block matching algorithm for motion estimation," in *ISCAS-96*, vol. 2, May 1996.
- [74] K. M. Uz, M. Vetterli, and D. J. LeGall, "Interpolative multiresolution coding of advance television with compatible subchannels," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 1, pp. 86–99, Mar. 1991.
- [75] S. Zafar, Y. Q. Zhang, and B. Jabbari, "Multiscale video representation using multiresolution motion compensation and wavelet decomposition," *IEEE Journal on Selected Areas in Communications*, vol. 11, pp. 24–25, Jan. 1993.
- [76] K. T. Lo and J. Feng, "Multiresolution block matching algorithm for motion estimation in video coding," in *ICCS-94*, vol. 3, Nov. 1994.
- [77] K. M. Nam, J.-S. Kim, R.-H. Park, and Y. S. Shim, "A fast hierarchical motion vector estimation algorithm using mean pyramid," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 5, pp. 344–351, Aug. 1995.
- [78] K. W. Cheng and S. C. Chan, "Fast block matching algorithms for motion estimation," in *ICASSP-96*, vol. 4, pp. 2311–2314, May 1996.
- [79] Y. Q. Shi and X. Xia, "A thresholding multiresolution block matching algorithm," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 7, pp. 437–440, Apr. 1997.
- [80] J. Chalidabhongse and C. C. J. Kuo, "Fast motion vector estimation using multiresolution-spatio-temporal correlations," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 7, pp. 477–488, June 1997.
- [81] T. H. Han and S. H. Hwang, "A novel hierarchical-search block matching algorithm and VLSI architecture considering the spatial complexity of the macroblock," *IEEE Trans. Consumer Electronics*, vol. 44, pp. 337–342, May 1998.

- [82] J. C. H. Ju, Y. K. Chen, and S. Y. Kung, "A fast rate-optimized motion estimation algorithm for low-bit-rate video coding," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 9, pp. 994–1002, Oct. 1999.
- [83] B. C. Song and J. B. Ra, "A fast multi-resolution block matching algorithm for motion estimation," *Signal Processing:Image Comm*, vol. 15, pp. 799–810, July 2000.
- [84] X. Song, T. Chiang, X. Lee, and Y. Q. Zhang, "New fast binary pyramid motion estimation for MPEG-2 and HDTV encoding," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 10, pp. 1015–1028, Oct. 2000.
- [85] J. H. Lee, K. W. Lim, B. C. Song, and J. B. Ra, "A fast multi-resolution block matching algorithm and its LSI architecture for low bit-rate video coding," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, pp. 1289–1301, Dec. 2001.
- [86] J. W. Zan, M. O. Ahmad, and M. N. S. Swamy, "New techniques for multi-resolution motion estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 12, pp. 793–802, Sept. 2002.
- [87] B. C. Song and K. W. Chun, "Multi-resolution block matching algorithm and its VLSI architecture for fast motion estimation in an MPEG-2 video encoder," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 14, pp. 1119–1137, Sept. 2004.
- [88] B. Liu and A. Zaccarin, "New fast algorithms for the estimation of block motion vectors," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 3, pp. 148–157, Apr. 1993.
- [89] Y. L. Chan and W. C. Siu, "A new block motion vector estimation using adaptive pixel decimation," in *ICASSP-95*, vol. 4, pp. 2257–2260, May 1995.
- [90] Y. Kim, C. S. Rim, and B. Min, "A block matching algorithm with 16:1 subsampling and its hardware design," in *ISCAS-95*, vol. 1, pp. 613–616, Apr. 1995.
- [91] K. T. Choi, S. C. Chan, and T. S. Ng, "A new fast motion estimation algorithm using hexagonal subsampling pattern and multiple candidates search," in *ICIP-96*, vol. 1, pp. 497–500, Sept. 1996.
- [92] Y. L. Chan and W. C. Siu, "New adaptive pixel decimation for block motion vector estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 6, pp. 113–118, Feb. 1996.
- [93] Y. L. Chan, W. L. Hui, and W. C. Siu, "A block motion vector estimation using pattern based pixel decimation," in *ISCAS-1997*, vol. 2, pp. 1153–1156, June 1997.
- [94] C. K. Ng and B. Zeng, "A new fast motion estimation algorithm based on search window subsampling and object boundary pixel decimation," in *ICIP-98*, pp. 605–608, Oct. 1998.
- [95] Y. Q. Wang, Y. Q. Wang, and H. Kuroda, "A globally adaptive pixel-decimation algorithm for block-motion estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 10, pp. 1006–1011, Sept. 2000.

- [96] H. H. Hsieh and Y. K. Lai, "A novel fast motion estimation algorithm using fixed subsampling pattern and multiple local winners search," in *ISCAS-01*, vol. 2, pp. 241–244, May 2001.
- [97] L. T. Chang, S. M. Phoong, and Y. P. Lin, "A multi-stage multi-candidate algorithm for motion estimation," in *ICASSP-01*, vol. 3, pp. 1613–1616, May 2001.
- [98] F. Moschetti and E. Debes, "About macroblock subsampling for motion estimation on IA-64," in *ICME-01*, pp. 1163–1166, Aug. 2001.
- [99] C. N. Wang, S. W. Yang, C. M. Liu, and T. Chiang, "A hierarchical decimation lattice based on N-queen with an application for motion estimation," *IEEE Signal Processing Letters*, vol. 10, pp. 228–231, Apr. 2003.
- [100] C. N. Wang, S. W. Yang, C. M. Liu, and T. Chiang, "A hierarchical N-queen decimation lattice and hardware architecture for motion estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 14, pp. 429–440, Apr. 2004.
- [101] C. D. Bei and R. Gray, "An improvement of the minimum distortion encoding algorithm for vector quantization," *IEEE Trans. Commun.*, vol. 33, pp. 1132–1133, Oct. 1985.
- [102] S. Eckart and C. Fogg, "ISO/IEC MPEG-2 software video codec," *Proc. SPIE*, vol. 2419, pp. 100–118, 1995.
- [103] "ITU-T recommendation H. 263 software implementation," 1995.
- [104] Y. L. Chan, W. C. Siu, and K. C. Hui, "Block motion estimation using adaptive partial distortion search," in *ICME-02*, vol. 1, pp. 477–480, Aug. 2002.
- [105] Y. L. Chan and W. C. Siu, "Search strategy for partial distortion elimination in motion estimation," *Electronics Letters*, vol. 38, pp. 1427–1428, Nov. 2002.
- [106] Y. L. Chan and W. C. Siu, "An adaptive partial distortion search for block motion estimation," in *ICASSP-03*, vol. 3, pp. 153–156, Apr. 2003.
- [107] J. N. Kim, S. C. Byun, Y. H. Kim, and B. H. Ahn, "Fast full search motion estimation algorithm using early detection of impossible candidate vectors," *IEEE Trans. Signal Processing*, vol. 50, pp. 2355–2365, Sept. 2002.
- [108] J. N. Kim and T. S. Choi, "A fast full-search motion-estimation algorithm using representative pixels and adaptive matching scan," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 10, pp. 1040–1048, Oct. 2000.
- [109] J. N. Kim, S. C. Byun, and B. H. Ahn, "Fast full search motion estimation algorithm using various matching scans in video coding," *IEEE Trans. Systems, Man and Cybernetics, Part C*, vol. 31, pp. 540–548, Nov. 2001.

- [110] J. N. Kim and T. S. Choi, "Adaptive matching scan algorithm based on gradient magnitude for fast full search in motion estimation," *IEEE Trans. Consumer Electronics*, vol. 45, pp. 762–772, Aug. 1999.
- [111] C. H. Cheung and L. M. Po, "Generalized partial distortion search algorithm for fast block motion estimation," in *ICASSP-01*, vol. 3, pp. 1601–1604, May 2001.
- [112] C. H. Cheung and L. M. Po, "A fast block motion estimation using progressive partial distortion search," in *International Symposium on Intelligent Multimedia, Video and Speech Processing*, pp. 506–509, May 2001.
- [113] C. H. Cheung and L. M. Po, "Adjustable partial distortion search algorithm for fast block motion estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 13, pp. 100–110, Jan. 2003.
- [114] K. Lengwehasatit and A. Ortega, "Computationally scalable partial distance based fast search motion estimation," in *ICIP-00*, vol. 1, pp. 824–827, Sept. 2000.
- [115] V. G. Moshnyaga, "A new computationally adaptive formulation of block-matching motion estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, pp. 118–124, Jan. 2001.
- [116] C. K. Cheung and L. M. Po, "A hierarchical block matching algorithm using partial distortion measure," in *ISCAS-97*, vol. 2, pp. 1237–1240, June 1997.
- [117] C. K. Cheung and L. M. Po, "A hierarchical block motion estimation algorithm using partial distortion measure," in *ICIP-97*, vol. 3, pp. 606–609, Oct. 1997.
- [118] T. M. Oh, Y. R. Kim, W. G. Hong, and S. J. Ko, "A fast full search motion estimation algorithm using the sum of partial norms," in *ICCE-00*, (Nagoya, Japan), pp. 236–237, Jan. 2000.
- [119] H. S. Wang and R. M. Mersereau, "Fast algorithms for the estimation of motion vectors," *IEEE Trans. Image Processing*, vol. 8, pp. 435–438, Mar. 1999.
- [120] M. Brnig and W. Niehsen, "Fast full-search block matching," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, pp. 241–247, Feb. 2001.
- [121] Y. S. Chen, Y. P. Hung, and C. S. Fuh, "Fast block matching algorithm based on the winner-update strategy," *IEEE Trans. Image Processing*, vol. 10, pp. 1212–1222, Aug. 2001.
- [122] S. M. Jung, S. C. Shin, H. Baik, and M. S. Park, "Nobel successive elimination algorithms for the estimation of motion vectors," in *MSE-00*, (Nagoya, Japan), pp. 332–335, Dec. 2000.
- [123] S. M. Jung, S. C. Shin, H. Baik, and M. S. Park, "New fast successive elimination algorithm," in *MWSCAS-00*, vol. 2, pp. 616–619, Aug. 2000.

- [124] Y. W. Huang, C. H. Tsai, and L. G. Chen, "Parallel global elimination algorithm and architecture design for fast block matching motion estimation," in *ICASSP-04*, vol. 5, pp. 153–156, May 2004.
- [125] J. Lu and Z. Lin, "Deliberation with "fast full-search block matching"," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 13, pp. 97–99, Jan. 2003.
- [126] Y. W. Huang, S. Y. Chien, B. Y. Hsieh, and L. G. Chen, "Global elimination algorithm and architecture design for fast block matching motion estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 14, pp. 898–907, June 2004.
- [127] C. Zhu, W. S. Qi, and W. Ser, "A new successive elimination algorithm for fast block matching in motion estimation," in *ISCAS-04*, vol. 3, pp. 733–736, May 2004.
- [128] L. C. Chang, K. L. Chung, and T. C. Yang, "An improved search algorithm for motion estimation using adaptive search order," *IEEE Signal Processing Letters*, vol. 8, pp. 129–130, May 2001.
- [129] J. N. Kim, D. K. Kang, S. C. Byun, I. L. Lee, and B. H. Ahn, "A fast full search motion estimation algorithm using sequential rejection of candidates from hierarchical decision structure," *IEEE Trans. Broadcasting*, vol. 48, pp. 43–46, Mar. 2002.
- [130] Y. Noguchi, J. Furukawa, and H. Kiya, "A fast full search block matching algorithm for MPEG-4 video," in *ICIP-99*, vol. 1, pp. 61–65, 1999.
- [131] T. M. Apostol, *Mathematical analysis*. Addison-Wesley Pub. Co, 1974.
- [132] T. Wiegand, X. Z. Zhang, and B. Girod, "Long-term memory motion-compensated prediction," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 1, pp. 70–84, Feb. 1999.
- [133] T. Wiegand, B. Lincoln, and B. Girod, "Fast search for long-term memory motion-compensated prediction," in *ICIP-98*, vol. 3, pp. 619–622, Oct. 1998.
- [134] J. Jeong and W. Ahn, "Subpixel-accuracy motion estimation using a model for motion compensated errors," in *Proc. PCS-93*, pp. 13. 4–13. 4–4, 1993.
- [135] Y. Senda, H. Harasaki, and M. Yano, "A simplified motion estimation using an approximation for the MPEG-2 real-time encoder," in *ICASSP-95*, pp. 2273–2276, 1995.
- [136] C. Du, Y. He, and J. L. Zheng, "PPHPS: a parabolic prediction-based, fast half-pixel search algorithm for very low bit-rate moving-picture coding," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 13, pp. 514–518, June 2003.
- [137] M. C. Chen and A. N. Jr., "Rate-distortion optimal motion estimation algorithms for motion-compensated transform video coding," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 8, pp. 147–158, Apr. 1998.