UNIVERSITY OF CALIFORNIA, SAN DIEGO

**Method and Architecture Design for Motion Compensated Frame Interpolation in High-Definition Video Processing**

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering (Electronic Circuits and Systems)

by

Yen-Lin Lee

Committee in charge:

Professor Truong Nguyen, Chair
Professor Yoav Freund
Professor Rajesh Gupta
Professor William Hodgkiss
Professor Nuno Vasconcelos

2009

The dissertation of Yen-Lin Lee is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

Chair

University of California, San Diego

2009

DEDICATION

To my family

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

The pursuit of a PhD education has been my memorable and challenging stage in my life. Without so much help, advice, and support from many people, I can hardly complete my PhD degree. I would like to acknowledge these important persons in my life.

First and foremost, I offer my sincerest gratitude to my advisor, Prof. Truong Nguyen, for his carefully considered advice, patience and inspirational enthusiasm for this research. I consider myself very fortunate to have studied at the Video Processing Laboratory and work with many brilliant people. I have a great deal of thanks to Prof. Truong Nguyen and his continuing support and encouragement to help me get through my most difficult times in research.

I would also like to thank my committee members, Prof. William Hodgkiss, Prof. Yoav Freund, Prof. Nuno Vasconcelos, and Prof. Rajesh Gupta, for providing me with valuable advice and comments to accomplish this work.

I wish to extend my appreciation to my nice and helpful labmates at the University of California, San Diego, for their companionship during these days spent at the lab, and for contributing to a friendly and motivational working environment. I would like to especially thank Natan Jacobson, Meng-Ping Kao, Wei-Hsin Chang, Shay Har-noy, Dung Vo, Carson Pun, Ai-Mei Huang, Jack Tzeng, Stanely Chan, and Nickolaus Mueller.

I owe a great deal of thanks to my parents, Chien-Erh Lee and Yueh-Chao Lee Hsu, my sisters, Patty Li and Yihsuan Lee, and my brother, Yen-Chi Lee, for encouraging me to set higher career goals and for showing endless love and support. I would also like to take this opportunity to acknowledge all my friends who always give me enormous support and encouragement.

Portions of Chapter 2 appear in "Method and Architecture Design for Motion Compensated Frame Interpolation in High-Definition Video Processing," Yen-Lin Lee and Truong Nguyen, in Proceedings of the *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2009. The dissertation author was the primary author of these publications, and the listed co-author directed and supervised the research that forms the basis for this chapter.

Portions of Chapter 3 appear in "Method and Architecture Design for Motion Compensated Frame Interpolation in High-Definition Video Processing," Yen-Lin Lee and Truong Nguyen, in Proceedings of the *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2009; "Fast One-pass Motion Compensated Frame Interpolation in High-Definition Video Processing," Yen-Lin Lee and Truong Nguyen, in Proceedings of the *IEEE International Conference on Image Processing (ICIP)*, Nov. 2009. The dissertation author was the primary author of these publications, and the listed co-author directed and supervised the research that forms the basis for this chapter.

Portions of Chapter 3 appear in "Method and Architecture Design for Motion Compensated Frame Interpolation in High-Definition Video Processing," Yen-Lin Lee and Truong Nguyen, in Proceedings of the *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2009; "Fast One-pass Motion Compensated Frame Interpolation in High-Definition Video Processing," Yen-Lin Lee and Truong Nguyen, in Proceedings of the *IEEE International Conference on Image Processing (ICIP)*, Nov. 2009. The dissertation author was the primary author of these publications, and the listed co-author directed and supervised the research that forms the basis for this chapter.

Portions of Chapter 4 appear in "Method and Architecture Design for Motion Compensated Frame Interpolation in High-Definition Video Processing," Yen-Lin Lee and Truong Nguyen, in Proceedings of the *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2009; "Fast One-pass Motion Compensated Frame Interpolation in High-Definition Video Processing," Yen-Lin Lee and Truong Nguyen, in Proceedings of the *IEEE International Conference on Image Processing (ICIP)*, Nov. 2009. The dissertation author was the primary author of these publications, and the listed co-author directed and supervised the research that forms the basis for this chapter.

Portions of Chapter 5 appear in "Method and Architecture Design for Motion Compensated Frame Interpolation in High-Definition Video Processing," Yen-Lin Lee and Truong Nguyen, in Proceedings of the *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2009; "Fast One-pass Motion Compen-

sated Frame Interpolation in High-Definition Video Processing," Yen-Lin Lee and Truong Nguyen, in Proceedings of the *IEEE International Conference on Image Processing (ICIP)*, Nov. 2009; "Novel Method and Architecture Design for Motion Compensated Frame Interpolation in High-Definition Video Processing," Yen-Lin Lee and Truong Nguyen, revised to *IEEE Trans. on Circuits and Systems for Video Technology*, 2009. The dissertation author was the primary author of these publications, and the listed co-author directed and supervised the research that forms the basis for this chapter.

Portions of Chapter 6 appear in "Novel Method and Architecture Design for Motion Compensated Frame Interpolation in High-Definition Video Processing," Yen-Lin Lee and Truong Nguyen, revised to *IEEE Trans. on Circuits and Systems for Video Technology*, 2009. The dissertation author was the primary author of these publications, and the listed co-author directed and supervised the research that forms the basis for this chapter.

Portions of Chapter 7 appear in "High Frame Rate Motion Compensated Frame Interpolation in High-Definition Video Processing," Yen-Lin Lee and Truong Nguyen, submitted to *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2010. The dissertation author was the primary author of these publications, and the listed co-author directed and supervised the research that forms the basis for this chapter.

## VITA

| | |
|---|---|
| 1999 | B. S., Electrical and Control Engineering, National Chiao Tung University, Hsinchu, Taiwan |
| 2001 | M. S., Electrical and Control Engineering, National Chiao Tung University, Hsinchu, Taiwan |
| 2009 | Ph. D, Electrical and Computer Engineering, University of California, San Diego |

## PUBLICATIONS

Yen-Lin Lee and T. Nguyen, "High Frame Rate Motion Compensated Frame Interpolation in High-Definition Video Processing," submitted to *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2010.

Yen-Lin Lee and T. Nguyen, "Fast One-pass Motion Compensated Frame Interpolation in High-Definition Video Processing," accepted in *IEEE International Conference on Image Processing (ICIP)*, Nov. 2009.

Yen-Lin Lee and T. Nguyen, "Method and Architecture Design for Motion Compensated Frame Interpolation in High-Definition Video Processing," *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1633-1636, May 2009.

Yen-Lin Lee and T. Nguyen, "Novel Method and Architecture Design for Motion Compensated Frame Interpolation in High-Definition Video Processing," revised to *IEEE Trans. on Circuits and Systems for Video Technology.*

Yen-Lin Lee and T. Nguyen, "Analysis and Efficient Architecture Design for VC-1 Overlap Smoothing and In-loop Deblocking Filter," *IEEE Trans. on Circuits and Systems for Video Technology*, Vol. 18, pp. 1786-1796, Dec. 2008.

Yen-Lin Lee and T. Nguyen, "Analysis and Integrated Architecture Design for Overlap Smooth and In-loop Deblocking Filter in VC-1," *IEEE International Conference on Image Processing (ICIP)*, pp. V161-V172, Sept. 2007.

ABSTRACT OF THE DISSERTATION

**Method and Architecture Design for Motion Compensated Frame Interpolation in High-Definition Video Processing**

by

Yen-Lin Lee
Doctor of Philosophy in Electrical Engineering (Electronic Circuits and Systems)
University of California San Diego, 2009
Professor Truong Nguyen, Chair

Digital displays such as Liquid Crystal Display (LCD) and plasma display televisions have become prevalent in recent years. Sports broadcasting and movies are two prime factors responsible for this popularity. However, motion blur and judder appear as objects move rapidly or color dramatically changes on a wide range of LCD devices because of slow response time and sample-and-hold drive nature. Frame Rate Up Conversion (FRUC) is a well-studied method that is used to minimize these detrimental effects.

A novel, fast, and efficient method with a well-designed architecture is proposed for Motion Compensated Frame Interpolation (MCFI) or Frame Rate Up Conversion. Unlike previous works involving high complexity, time-consuming iterations, and higher complex architecture, the proposed method adopts a one-pass, low-complexity approach without any iteration and is capable of dealing with High Definition (HD) video processing. Rather than a conventional motion estimation in MCFI, our method employs a unique true motion engine that explores at most nine motion candidates with different motion directions and then determines one true motion by referring to neighboring spatial and temporal information.

For the purpose of motion estimation, the proposed method introduces an adaptive overlapped block matching algorithm known as the Multi-Directional Enlarged Matching Algorithm (MDEMA), and considers different overlapped types

based on the direction of the current motion vector in order to enhance searching accuracy and visual quality. For practical issues and real-time HD requirements, the proposed architecture employs a modified Multi-level Successive Eliminate Algorithm (MSEA), which is a Fast Full-Search Block Matching Algorithm (FFS-BMA) and has the ability to reduce the heavy computation of full search while maintaining similar quality. According to analyzed temporal information, our method explores true motion candidates and refines the accuracy of true motions for blocks or sub-blocks. Experimental results show that the proposed algorithm provides better video quality than conventional methods and demonstrates excellent performance for 30fps HD1080p video (1920×1080 resolution) at 180MHz or 30fps 720p video (1280×720) at 83MHz.

# 1 Introduction

Digital displays, such as LCD and plasma, have become widely prevalent in recent years. Sports and movies have been prime factors in the popularity of large screen displays. Unfortunately, artifacts such as motion blur and judder appear due to rapid object motion or dramatic change in color on a wide range of LCD devices. These issues result from slow response time and sample-and-hold drive nature. Frame Rate Up Conversion (FRUC) is a well-studied method that is used to eliminate or reduce a device's physical disadvantages by inserting interpolated frames between any two successive original frames. Another practical application of FRUC is to enhance the reconstructed quality of a low bit rate decoded video as channel bandwidth is limited. Some conventional FRUC methods with low complexity, such as frame repetition and linear frame interpolation [1], yield blurred objects because these methods do not take motion information into account.

Motion Compensated Frame Interpolation (MCFI) [2][3][4] has been widely studied to enhance the quality of reconstructed video. This method explores block motions of the interpolated frames by processing using motion re-estimation or the received motion vectors from an encoded bitstream. MCFI can effectively reduce motion blur and improve visual quality by increasing the frame rate when it acquires true motion information. However, MCFI which directly processes motion vectors from a video encoder may suffer from blocking and ghost artifacts. This is because the encoder seeks to minimize prediction error rather than determining true motion. For this reason, many researchers have started to work on methods to accurately estimate true motion by considering spatial and temporal correlation

and searching with a Block Matching Algorithm (BMA) [5]. Fig. 1.1 shows an example for block-based MCFI.

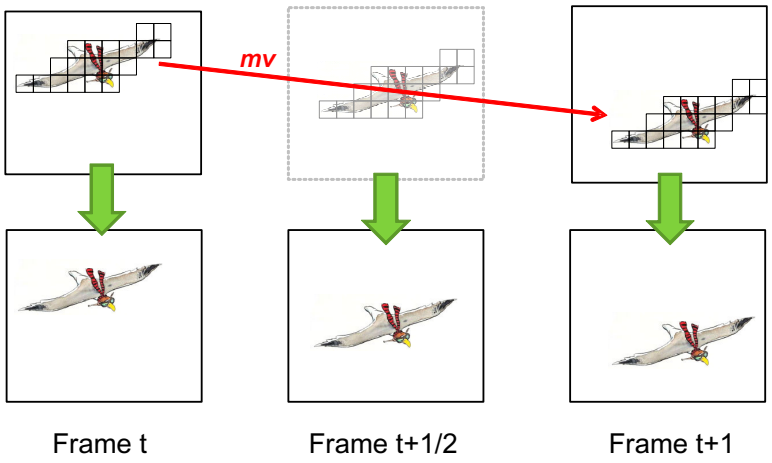



Figure 1.1: Motion Compensated Frame Interpolation with block-based motion search.

In the past ten years, numerous methods have been proposed to determine true motion. However, these methods involve complex computation, complicated time-consuming iterations, and are difficult to implement for real-time High-Definition (HD) videos. Considering these drawbacks, the proposed method and architecture adopts a one-pass, low-complexity design without any repeated iteration and is developed with consideration for fast hardware implementation for HD video processing. In addition, the proposed approach has no restriction on video compression format due to a low complexity motion re-estimation. The proposed method is composed of bidirectional estimation, Multi-Directional Enlarge Matching Algorithm, one-pass motion vector selection with multi-grids classification, temporal/spatial object information and localized global motion vectors, Motion Compensated Frame Interpolation, and deblock filtering.

For practical issues and the real-time HD requirement, the proposed architecture is designed with a true motion engine that employs a modified Multi-level Successive Eliminate Algorithm (MSEA) [9], Probable Sum of Absolute Difference (PSAD), true motion selection, and a refinement of motion vector processing. MSEA is a Fast Full-Search Block Matching Algorithm (FFSBMA) [10] for mo-

tion estimation following a Successive Elimination Algorithm (SEA) [11], which has gained popularity for the ability to reduce the heavy computation of Full-Search Block Matching Algorithm (FSBMA) [12] while maintaining similar quality. Although MSEA has initially been developed for video compression, the proposed architecture introduces and modifies this technique to enhance the accuracy of true motion search and reach a balance between complexity and performance.

In order to have a better understanding of Motion Compensated Frame Interpolation and the platform it operates on, we start the dissertation with an introduction to MCFI in Chapter 2. MCFI is a popular approach to generate interpolated frames which considers motion information in an attempt to reduce motion judder and blur effects. Conventional MCFI methods are also discussed in this chapter with illustrations on the techniques that improve visual quality for Frame Rate Up Conversion.

In Chapter 3, a dedicated processing method is proposed, which adopts one-pass and low-complexity design without need for iteration. The proposed techniques are developed based on a motion re-estimation in order to obtain accurate true motion, which has no restriction on specific video compression format. The methods and techniques adopted in our proposed MCFI are presented in this chapter.

In order to reduce the computational complexity, the proposed architecture first introduces a fast full search method, Multi-level Successive Elimination Algorithm (MSEA) in Chapter 4. In this chapter, a practical architecture is proposed to implement all methods proposed in Chapter 3. The proposed architecture has the capability to deal with Frame Rate Up Conversion for a 1080p sequence up to 60fps.

In Chapter 5, the performance of our proposed method and architectural design are examined. To verify the accuracy and the efficiency for the proposed method and architecture, the architecture is designed in VHDL (VHSIC (Very High Speed Integrated Circuits) Hardware Description Language) and implemented with TSMC (Taiwan Semiconductor Manufacturing Company Limited) 90-nm technology. The implemented architecture operates with our VHDL and MATLAB im-

plementation, and the result has been verified with our prototype in MATLAB. In addition to architecture implementation, PSNR tests are used to compare the performance of different MCFI methods including the proposed methods and conventional methods. We test visual quality by conducting subjective tests with human observers. The results demonstrate that the proposed method has better visual quality than conventional methods.

In Chapter 6, each technique of the aforementioned methods will be separately discussed via system analysis and profiling. In this chapter, we will examine each component's contribution and process loading in the proposed MCFI design. Due to the widespread popularity of digital displays, more and more high quality LCD devices with high frame rate have emerged in the market. However, current video sources, media storage, and decoding power are limited to 30- or 60-fps and make it impossible to transmit a 120- or 240-fps High Definition (HD) compressed video bitstream. Hence, the method for high Frame Rate Conversion will be proposed in Chapter 7. Finally, the conclusions will be summarized in Chapter 8.

# 2 Motion Compensated Frame Interpolation

## 2.1 Introduction of Motion Compensated Frame Interpolation

Motion blur occurs because of a disparity between Human eye tracking and the actual displayed data [13]. Fig. 2.1(a) depicts the case of an object moving in a horizontal direction at a constant velocity on an LCD with zero response time. Variable x denotes position and t shows frame time. Eye tracking causes Human observers to form a target trajectory for the object. However, due to the sample-and-hold characteristics of the LCD, the output at each pixel is held constant for the entire frame period. The difference between the eye tracking trajectory and the actual displayed data corresponds to the motion blur perceived by the human observer. Fig. 2.1(b) shows the case of reducing motion blur if data sample rate increases.

In contrast to conventional methods that linearly average [1] or temporally filter two or more successive frames, Motion Compensated Frame Interpolation (MCFI) is a popular approach to generate interpolated frames for a Frame Rate Up Conversion (FRUC) with higher quality and accuracy. MCFI considers motion information to reduce motion judder and blur effects. Although MCFI can enhance visual quality, it may still suffer from blockiness and ghost effects due to a failure of finding true motions. Contemporary proposals and methods in MCFI can be classified into two categories: motion re-estimation method and motion vector

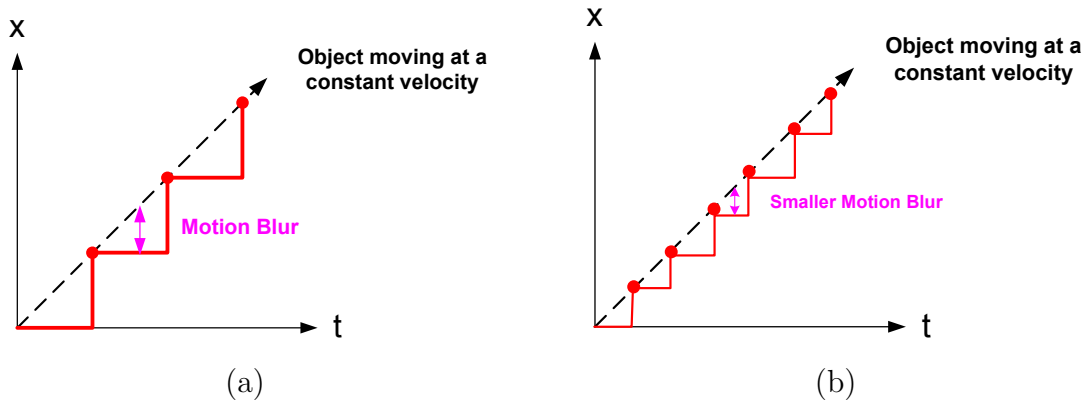



Figure 2.1: The difference between the eye tracking trajectory and the actual displayed data with zero response time. (a) Original frame rate. (b) Twice the original frame rate.

processing method. Fig. 2.2 shows block diagrams for these two approaches.

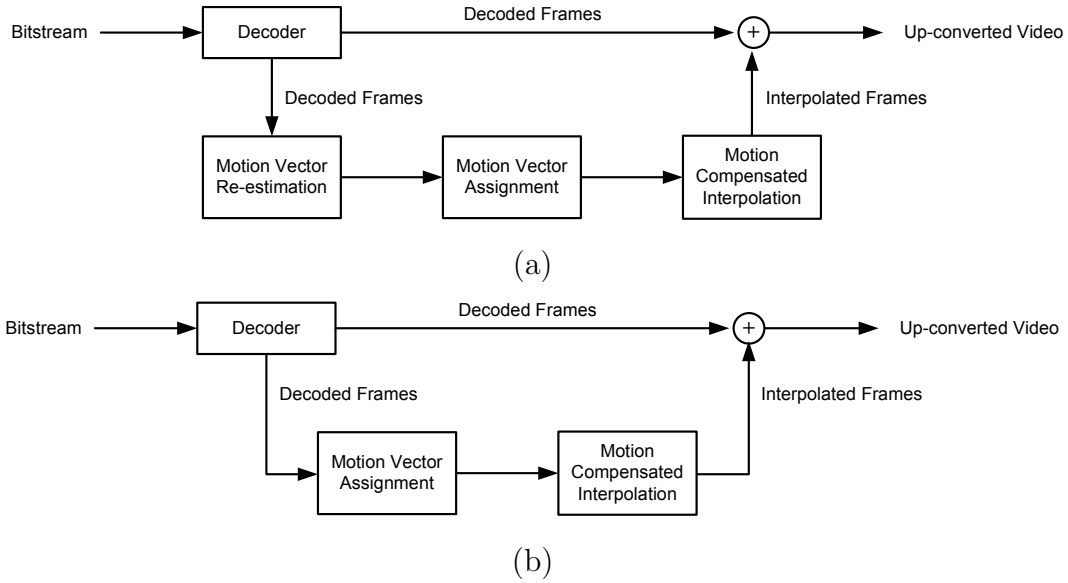



Figure 2.2: Two MCFI approaches. (a) Motion re-estimation method. (b) Motion vector processing method.

The motion re-estimation method in Fig. 2.2(a) is typically used for applications which demand high quality or multi-standard support. It is because it takes decoded frames from the decoder, and it is independent of the codec used. The disadvantage is the need to implement a motion estimator, which greatly increases computational complexity and memory bandwidth. In addition, lossy compressed

frames also increase the difficulty of true motion search. The concept of the motion vector processing method in Fig. 2.2(b) is to extract the motion vectors from the decoder and process this Motion Vector Field (MVF) so as to assign a true motion for each block. Although this method has lower complexity, the quality strongly depends on performance of an encoder's searching algorithm because block-based motion estimation at the encoder minimizes prediction errors rather than finding true motions. In other words, an unfavorable encoding may ruin the true motion vector field and create challenges in motion vector processing method. Here, we briefly review previous works.

In 1989, Soryani et al. proposed an approach [6] to coding of moving sequences that combines image segmentation with adaptive thresholding based on a priori knowledge about the scene and motion-adaptive frame interpolation techniques. 3-D Recursive Search (3DRS) [5] enhances video information evaluates candidate vectors of enhancement algorithms utilizing an error function biased towards spatio-temporal consistency with a penalty function. Another temporal frame interpolation technique is presented in [7], based on an object-based algorithm for 3-D motion estimation. This algorithm uses a joint estimation-segmentation scheme to minimize the displaced frame difference between a frame and its motion compensated prediction from the previous frame. The main novelty of the proposed method in [8] is the motion compensation algorithm which has been designed with low computational complexity as an important criterion. In [14], Chen proposed an adaptive temporal interpolation method with both forward and backward estimation conducted with a correction constraint. In [15], the method based on a pyramid structure and the motion compensation process is performed independently at each resolution level. A technique similar to Control Grid Interpolation (CGI) is employed to process hole regions generated at the top level of the pyramid.

Subsequently, a novel method that adopts Overlapped Block-based Motion Estimation (OBME) was proposed by Ha et al. in [16] to obtain higher accuracy motion trajectory. From the approach in [17], the motion vector from Phase Plane Correlarion (PPC) method is used that assigns multiple motion vectors to a block

thus achieving the result of object based approaches with much fewer operation counts. A method that examines the motion vectors by bi-directional motion estimation was proposed in [18] to remove and correct unreliable motion vectors from the bitstream. Kuo et al. proposed two add-on schemes to enhance a Fast Block-based Motion-Compensated Frame Interpolation (FMCI) [19] in their research. These are the Adaptive Frame Skip (AFS) scheme applied at the H.263 [20] encoder and the Hybrid Frame Interpolation (HFI) scheme which incorporates both frame repetition and FMCI at the decoder. Fujiwara et al. presented a technique [21] based on different block sizes in order to realize the clear interpolation of moving objects regardless of the object's size. Perspective transforms were introduced in [22] to reduce blocking artifacts in boundary blocks. A motion vector processing method was proposed by Dane et al. in [23] when an optimal temporal filter was obtained by minimizing the prediction error variance between original frame and interpolated frame. In [24], Choi et al. partitioned a frame into several object regions by clustering motion vectors and applied the Variable-Size Block Motion Compensation (VS-BMC) and Adaptive Overlapped Block Motion Compensation (AOBMC) to remove the limitations of conventional OBMC.

Yang et al. developed new criteria and coding scheme and adopted adaptive frame skip to guarantee the quality of interpolation [25]. A low complexity Motion Compensated Frame Interpolation method using compressed-domain information based on an H.264 decoder is presented in [26]. In this proposed method, the motion vectors are estimated using the constant acceleration motion model, and the interpolation algorithm is applied based on the macroblock coded types. Huang et al. proposed a motion vector processing algorithm that considers the reliability of motion vectors by analyzing the distribution of residual energies and merging blocks [27]. Yang et al. proposed to select the parameters and thresholds by analysing the statistical characterization of motion vectors and residual energy, thus thresholds can be changed adaptively during the decoding process [28]. The selected moving objects on each decoded frames, are meshed from quadrilateral blocks which are then deformed using specific warping functions in [29]. The meshed objects are reconstructed to the predicted nodes and integrated in

the H.264/AVC video coding standard. [31] addresses the problems of unreliable motion vectors that cause visual artifacts but cannot be detected by high residual energy or bidirectional prediction difference in motion-compensated frame interpolation. Song et al. use three frames to provide an algorithm in [30] to detect the occlusion area and differentiate the covering/uncovering area for Motion Compensated Frame Interpolation. This method aims at reducing artifact in occlusion areas. A correlation-based motion vector processing method is proposed in [31] to detect and correct those unreliable motion vectors by explicitly considering motion vector correlation in the motion vector reliability classification, motion vector correction, and frame interpolation stages.

Although there are many well-developed proposals on MCFI and FRUC, computational complexity is still a serious problem for a real-time processing to cope with the HD requirements due to multi-stage processing and complex searching procedures. Hence, we propose a novel, fast, and hardware-friendly method and its architecture design. The techniques of the proposed method will be discussed in the following chapters.

## 2.2 Acknowledgement

Portions of Chapter 2 appear in "Method and Architecture Design for Motion Compensated Frame Interpolation in High-Definition Video Processing," Yen-Lin Lee and Truong Nguyen, in Proceedings of the *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2009. The dissertation author was the primary author of these publications, and the listed co-author directed and supervised the research that forms the basis for this chapter.

# 3 The Proposed MCFI Method

As mentioned in Chapter 2, although many methods have been proposed to deal with the processing for temporal frame interpolation, they involve complex computation, time-consuming iterations, and are difficult to implement for real-time high definition systems. In this chapter, one dedicated processing method is proposed, which adopts one-pass and low-complexity design without repeated iteration. The proposed techniques are developed based on motion re-estimation in order to obtain accurate true motion with no restriction to specific video compression format.

## 3.1 Proposed Processing Flow

The proposed scheme adopts a motion-compensated approach to insert one or several interpolated frames between any two contiguous original frames. Accordingly, determining a true motion is the primary and most essential step to generate interpolated frames. Fig. 3.1 shows the entire processing flow of the proposed method including a true motion engine.

The entire processing procedure is composed of three major parts: a true motion engine, a block-based interpolator, and a deblocking filter. The true motion engine takes charge of motion estimation and true motion decision, and it also outputs true motion vectors to the block-based interpolator. The block-based interpolator obtains motion information and performs a motion compensated procedure with weighted values. At last, a deblocking filter smoothes the blocking artifacts where a neighboring or current interpolated block with a large Sum of

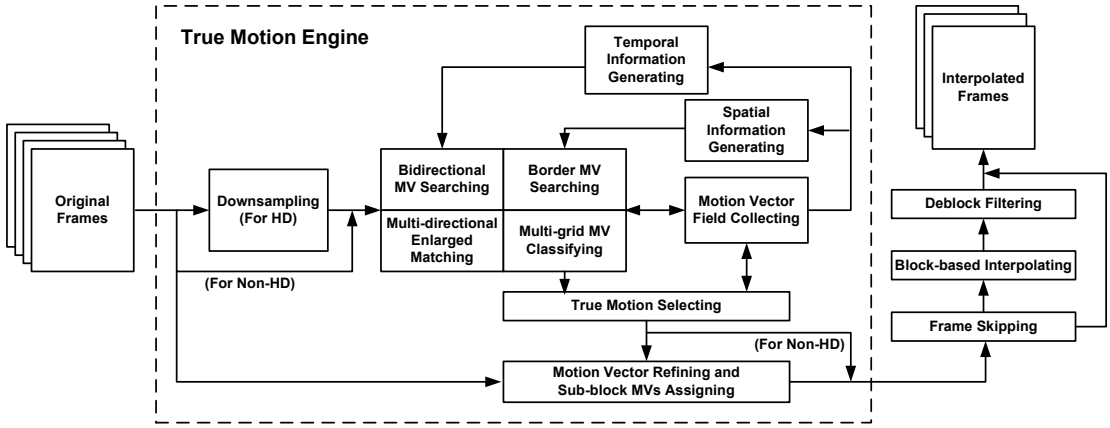



Figure 3.1: Processing flow of the proposed method.

Absolute Difference (SAD) shows up.

The purpose of the true motion engine is to assign a true motion vector for each individual block or sub-block, and this engine searches the motion with different processing steps. In our proposed method, a simple down-sampling procedure is performed to reduce computation when input images are of HD resolution, such as HD1080p or HD720p. If this down-sampling procedure is activated, a refinement is executed to increase accuracy for true motion vectors. Other functions include: bidirectional motion vector searching, multi-directional enlarged matching, border motion vector searching, and multi-grids motion vector classification. Most of these techniques refer to spatial and temporal information to acquire motion candidates for the purpose of true motion selection. The true motion selector chooses the best motion vector for compensation according to pre-defined conditions and neighboring and global motion information. After the true motion engine determines the true motions, a block-based interpolator generates each block image for an entire interpolated frame, and then a deblocking filter reduces blocking artifacts with a simple filtering operation before outputting the upconverted frames. Similar to the method in [25], the proposed method also introduces a skipping method to skip some difficult situations when the proposed method cannot generate good quality interpolated frames, especially for large moving motions, scene changes, and dramatic color changes.

## 3.2 Proposed True Motion Search

In this section, a dedicated true motion processing method is proposed, which adopts one-pass and low-complexity design without any repeated iteration. The proposed techniques are developed based on motion re-estimation in order to obtain accurate true motions, which has no restriction on specific video compression formats.

### 3.2.1 Bidirectional Search

For acquiring an accurate true motion, we employ motion re-estimation with bidirectional search on reconstructed frames from a decoder rather than processing and correcting a motion vector field extracted from a compressed bitstream. Naturally, a method without motion re-estimation should have lower complexity, but it is difficult to find true motions when the motion vector field is inaccurate and irregular. Furthermore, repeated iterations increase computation on high-definition video with high frame rate, such as 30fps or more. Fig. 3.2(a) shows a unidirectional searching method that is based on a block at the same location on the previous or subsequent frame and moves the motion vector along the trajectory. This is simple and compatible to any conventional encoder, but it might result in an inaccuracy as motions are inconsistent. Fig. 3.2(b) shows the proposed method using a bidirectional search. The proposed bi-directional search directly seeks two similar blocks from a zero distance of the interpolated block. $mv\_f$ and $mv\_b$ represent a forward motion and a backward motion respectively. For up-conversion ratios other than 2, $mv\_f$ and $mv\_b$ can be different distances to allow for interpolation of an arbitrary frame rate.

Two adjacent frames are denoted by *f(x,t)* and *f(x,t+1)*, where $x$ and $t$ are spatial and time domain indices. In the case that the up-conversion rate equals two, a motion vector, $\overrightarrow{V}$, of an N×N interpolated block is formulated by

$$\overrightarrow{V} = \arg\min_{\overrightarrow{v} \in S} \sum_{x \in B} |f(x - \overrightarrow{v}, t) - f(x + \overrightarrow{v}, t + 1)| \tag{3.1}$$

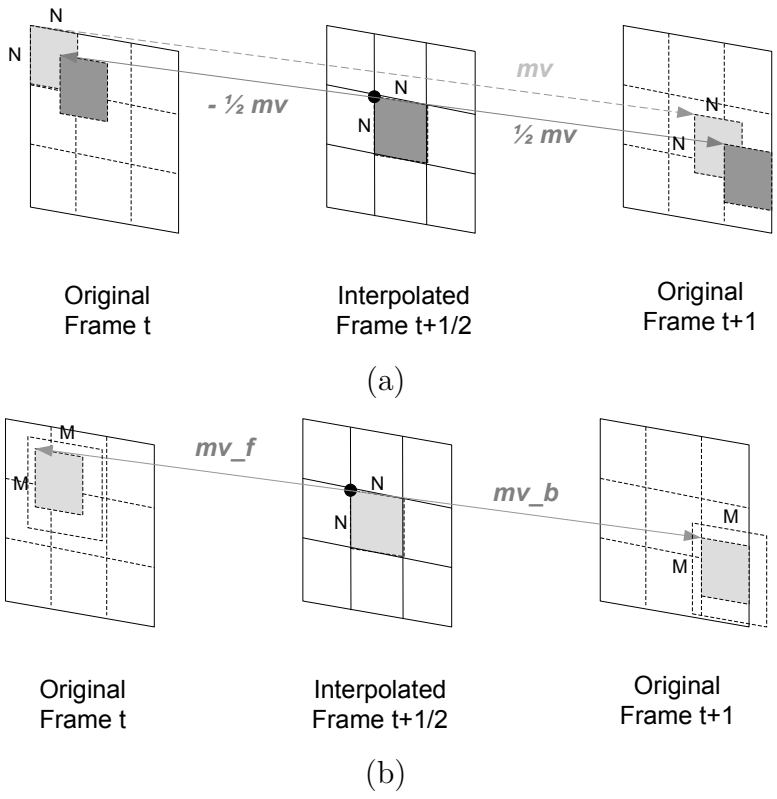



Figure 3.2: MCFI motion estimation method. (a) Unidirectional search. (b) Proposed bidirectional search.

where $B$ denotes a matching M×M block of the current interpolated position; $S$ is a set of motion vectors in our search range; $\overrightarrow{v}$ is the motion vector examined for the best matching. The interpolated block size of N×N is different than the matching block size of M×M for searching more accurate true motion where M is equal or larger than N. The concept of the proposed method is based on searching for the minimum SAD value using a Block Matching Algorithm. The proposed architecture also applies in adaptive overlapped block motion estimation algorithm, Multi-Directional Enlarged Matching Algorithm (MDEMA), to enhance the accuracy of true motion search and lower the presence of visual blurred artifacts. From our experiments, the engine using bidirectional search achieves better performance and botains a more precise motion vector field than unidirectional search.

### 3.2.2 Multi-Directional Enlarged Matching Algorithm

For searching motion candidates and true motion, the proposed method defines different block sizes for the interpolated block and the matching block. The current interpolated block size is defined by N×N pixels, and the matching block size is defined as M×M pixels. Compared to the conventional block matching method, larger matching block size improves the accuracy of true motion search because a larger part of a particular object is considered. However, it might lose some details surrounding the moving object. Based on our experiments, the results reveal that using smaller matching size is worse and often obtains the wrong motion with a smaller pixel difference instead of a true motion, especially for blocks with little texture. Therefore, how to take advantage of a larger block matching size to search a true motion for an interpolated block is a required topic. Although previous works employ an overlapped block searching method as well, the proposed method introduces a novel enlarged matching method, MDEMA. There are two searching modes for the MDEMA: (1) nine-directional enlarged matching and (2) direct multi-directional enlarged matching. The former is used with the MSEA engine to roughly search motion candidates. The MSEA is a fast full-search method, which will be further explained in the next chapter. The latter is used with the PSAD engine to exactly determine a motion vector for each divided search window from the proposed multi-grids classification, which defines nine different partitioned windows for motion candidates. The PSAD will also be explained in the next chapter. Here, we describe the operation of MDEMA.

In accordance with different motion directions, nine types of enlarged matching directions are defined and shown in Fig. 3.3. Small squares in the figure depict N×N block size, and this area will be used for interpolation as we find the most suitable true motion vector by matching with a block size of M×M. The proposed method determines the type by referring to the current motion vector and the distance from the position with zero motion. For example, consider the case with slow motion. When the current searching motion is small and near the zero motion, the approach chooses type 1 to enlarge the matching block. If the current searching motion goes to the top left side, the proposed approach chooses type

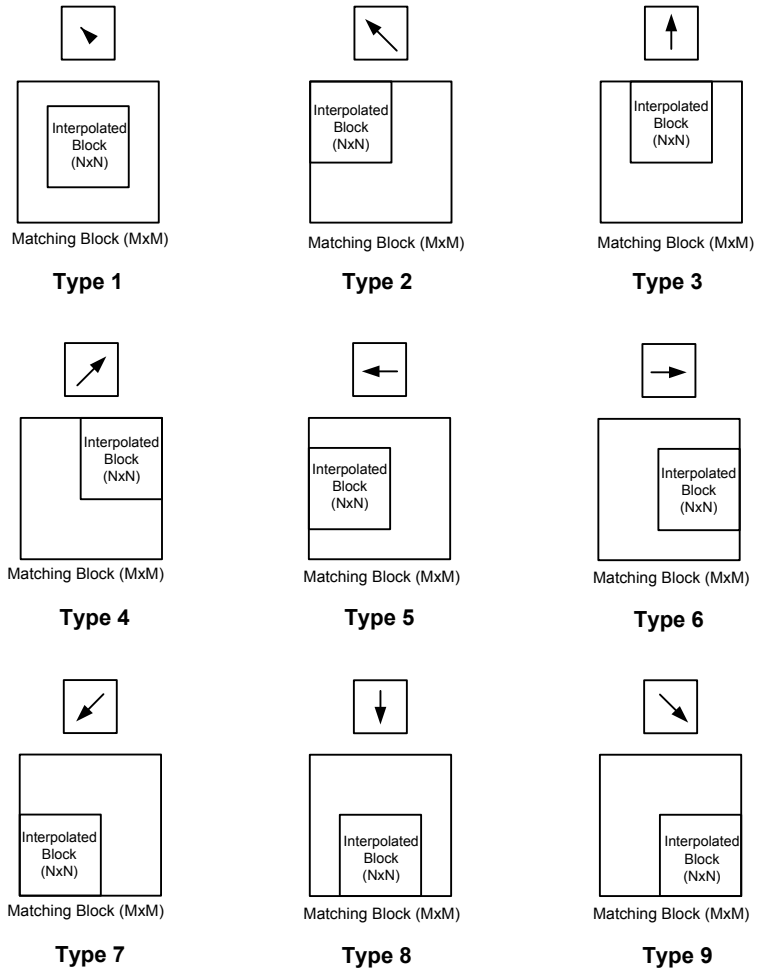



Figure 3.3: Nine-directional enlarged matching method.

2 to enlarge the matching block. If the current searching motion goes to upper side, the proposed approach chooses type 3 to enlarge the matching block, and so on. The reason for using different enlarged directions is to keep the completeness of the front edge of the moving object. From our observation, viewers are more sensitive to the artifacts and fractures of the moving object's front edge. Due to the visual quality, we aim to keep the front edge of the moving object intact by utilizing these enlarged matching types.

When generating and inserting a new frame between original frame $t$ and $t$+1, the proposed method processes the frame in raster scan order based on an interpolated block size, N×N. For each N×N interpolated block, motion candidates

are found by using bidirectional search mentioned above. These motion candidates will be immediately examined with motion information of neighboring blocks without iterations, which is called one-pass processing in the proposed method. In most situations, we choose a motion vector with surrounding consistent motion as true motion rather than with the minimal MSEA or PSAD (these techniques will be discussed in a later chapter). Take type 4 for example, as shown in Fig. 3.4. When processing the N×N interpolated block C, the M×M matching block O1 of frame $t$ and the M×M matching block O2 are our best matching pair. The interpolated block would be a weighted average of these two blocks.

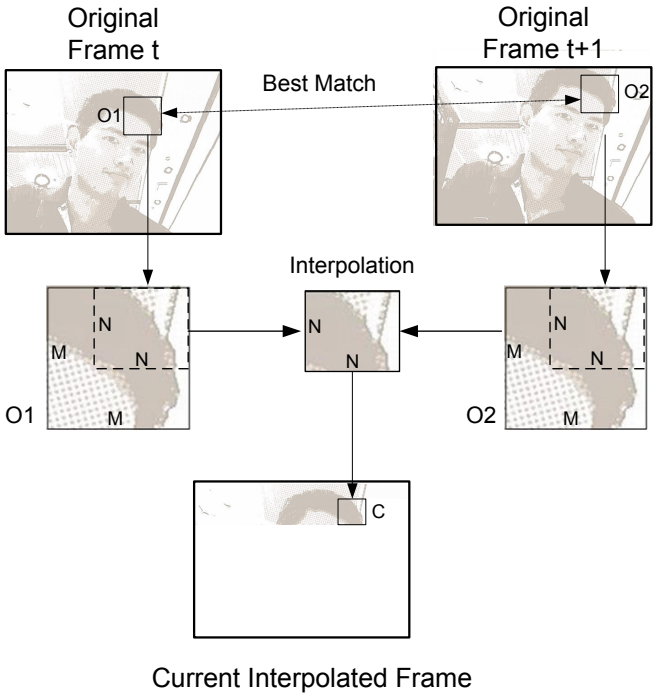



Figure 3.4: Nine-directional enlarged matching method with type 4.

The other multi-directional mode is the direct multi-directional enlarged matching method. Nine-directional enlarged matching method is derived from direct multi-directional enlarged matching method for fast computation and implementation during processing. Direct multi-directional enlarged method, shown in Fig. 3.5, adopts a more precise enlarged range and considers exact direction and magnitude of motion candidates. Because the MSEA engine already performs

a full search and then obtains several motion candidates for the PSAD, direct enlarged matching method is not applied on every motion vector within the entire searching window. There are four motion candidates for each divided searching window and thirty-six motion candidates for an entire searching window in the proposed implementation.

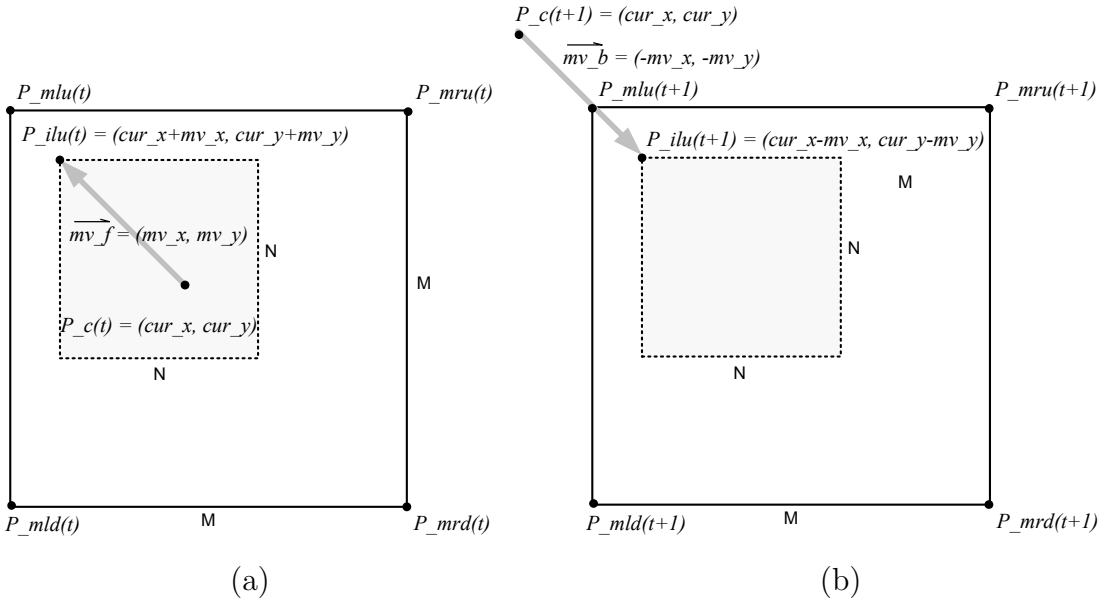



Figure 3.5: Direct multi-directional enlarged matching method (a) Matching window and interpolated block in a previous frame $t$. (b) Matching window and interpolated block in a successive frame $t$+1.

The proposed PSAD method with direct enlarged matching method defines M×M matching blocks on previous frame $t$ and successive frame $t$+1 when searching for a best matching pair. The PSAD is very similar to the SAD in that it finds the minimum sum of the pixel difference, but it only examines motion candidates from the MSEA. The corners of these two blocks in Fig. 3.5 are defined by

$$Corner\_positions(t) = \begin{cases} P\_mlu(t) = (mlu\_x(t), mlu\_y(t)) \\ P\_mru(t) = (mlu\_x(t) + M, mlu\_y(t)) \\ P\_mld(t) = (mlu\_x(t), mlu\_y(t) + M) \\ P\_mrd(t) = (mlu\_x(t) + M, mlu\_y(t) + M) \end{cases} \tag{3.2}$$

$$where \begin{cases} mlu\_x(t) = cur\_x + mv\_x + \left(\frac{\max\_mv\_x - |mv\_x|}{\max\_mv\_x}\right) \times sign(mv\_x) \times \frac{M}{4} - \frac{M}{4} \\ mlu\_y(t) = cur\_y + mv\_y + \left(\frac{\max\_mv\_y - |mv\_y|}{\max\_mv\_y}\right) \times sign(mv\_y) \times \frac{M}{4} - \frac{M}{4} \end{cases}$$

$$Corner\_positions(t+1) = \begin{cases} P\_mlu(t+1) = (mlu\_x(t+1), mlu\_y(t+1)) \\ P\_mru(t+1) = (mlu\_x(t+1) + M, mlu\_y(t+1)) \\ P\_mld(t+1) = (mlu\_x(t+1), mlu\_y(t+1) + M) \\ P\_mrd(t+1) = (mlu\_x(t+1) + M, mlu\_y(t+1) + M) \end{cases}$$

$$where \begin{cases} mlu\_x(t+1) = cur\_x - mv\_x + \left(\frac{\max\_mv\_x - |mv\_x|}{\max\_mv\_x}\right) \times sign(mv\_x) \times \frac{M}{4} - \frac{M}{4} \\ mlu\_y(t+1) = cur\_y - mv\_y + \left(\frac{\max\_mv\_y - |mv\_y|}{\max\_mv\_y}\right) \times sign(mv\_y) \times \frac{M}{4} - \frac{M}{4} \end{cases} \tag{3.3}$$

### 3.2.3 One-pass Motion Selection with Multi-grids Classification

The proposed approach adopts a one-pass processing method, which means that no iteration is needed when determining a true motion vector field. This allows faster processing time and reduces memory requirements. From this viewpoint, it does not rely on information of blocks subsequent to the current interpolated block (except searching result for the next block). In other words, this method only counts on information of motion vectors from previous processed blocks, which is similar to advanced techniques in video compression standards, such as H.264 [32] and VC-1 [33].

Effective neighboring blocks are shown in Fig. 3.6 and grouped under two levels: N1, N2, N3, N4 and N5 belong to the first level, and N6, N7, N8, and N9 belong to the second level. Blocks in the second level are used to determine how reliable block information is in the first level. C represents the current interpolated block and will be assigned a true motion after the process. The effect of the

neighboring blocks in the first level is more significant than those in the second level, and two levels are considered together: N1 and N6 are regarded as a group; N2 and N7 are regarded as a group; N3 and N8 are regarded as a group; N4 and N9 are regarded as a group. The information of these neighboring blocks or groups is used when determining a true motion.

Block N5 is processed in a unique way during this procedure. As mentioned above, the spatial information comes from the blocks processed prior to the current block. However, N5 does not belong to the processed blocks. According to the experiment, a decision considering motion from N5 greatly increases the correctness when choosing a true motion from nine motion candidates. Hence, the true motion selection for C should be done subsequent to motion search of N5. This searching procedure does not include the decision of the final true motion but obtain true motion candidates for reference. It is acceptable to search one more block in raster scan order because N5 is the subsequent block in this order. Necessary pixels are stored in local memory so as to process the true motion selection and interpolation. However, although the proposed method benefits from one-pass processing and less external memory access by ignoring information from following neighboring blocks, it also loses some advantage from missing information and cannot realize motions from the right or below. To compensate for this shortcoming, the proposed method utilizes temporal information, temporal object information and global motion vectors to enhance the search quality.

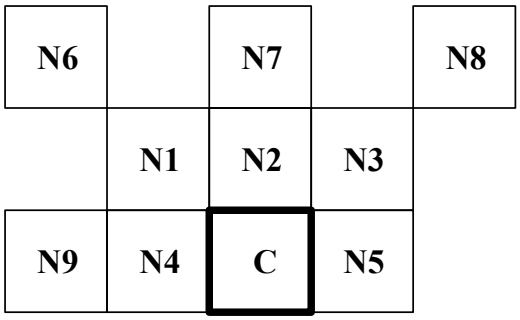



Figure 3.6: Effective neighboring blocks for spatial information.

After defining which neighboring block would influence the searching strat-

egy, the proposed method introduces a multi-grids classification to simplify processing and reduce the storage requirement for the motion vector field. It divides a search window into multiple areas shown in Fig. 3.7 (9 grids), and each area has one motion candidate after processing the MSEA and the PSAD. After a motion vector is determined with pre-defined conditions, this motion vector is refined and used for generating an interpolated block of the inserted frame. Then, the procedure will classify this true motion into one of these nine classes and record the classification instead of the motion vector value. This approach simplifies our processing procedure and also lowers memory access bandwidth.

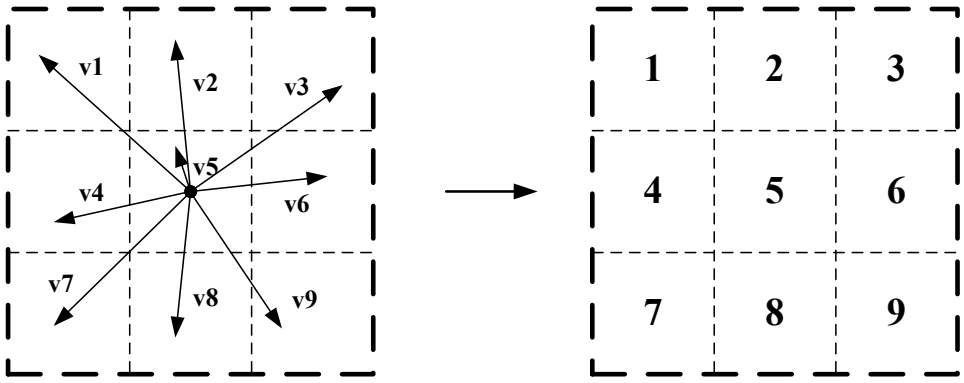



Figure 3.7: Multi-grids Classification in a searching window.

### 3.2.4 Temporal and Spatial Object Information

Although one-pass processing does not provide complete neighboring information, such as motion types from the right of or below neighboring blocks relative to the current interpolated frame, it can acquire temporal information from the previous processing result to realize object motion. Here, we assume that an object at a location in the previous frame has high probability of appearing near this location in most cases. Since all motion information is known for previous frames, true motion fields from previous results help us search for current true motion. Fig. 3.8 shows an example for multi-directional enlarged bidirectional search algorithm with temporal information.

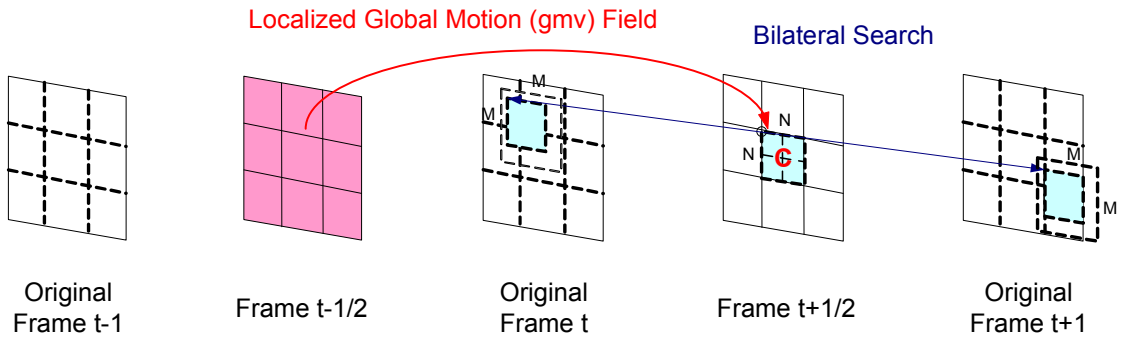



Figure 3.8: Multi-directional enlarged bidirectional search algorithm with temporal information.

Temporal information is a method to get an object's motion which has a constant speed, especially for panning scenes. When the camera pans slowly across a scene, most interpolated blocks move consistently except foreground objects. However, not all information from previously interpolated blocks is considered for global motion in our proposed method. Only interpolated blocks with a large pixel variance are considered, which means that the large part of an object will be examined. This method removes inaccurate motion vectors from being taken into account. However, a simple and uniform pattern easily matches a similar block with a false true motion, such as the pattern on a white board, and will provide inaccurate, irregular, and unreliable motion information. These inaccurate motion vectors lower the accuracy of global motion, and therefore are not considered. The Sum of Absolute Variance (SAV), is formulated by

$$SAV(B) = \sum_{k=1}^{N-1} \sum_{l=1}^{N-1} (|p(k,l) - p(k,l+1)| + |p(k,l) - p(k+1,l)|) \qquad (3.4)$$

where the block size of an interpolated block B is N×N. A pixel value within this interpolated block is denoted by $p(x,y)$, where $x$ and $y$ are spatial indices relative to the pixel's position at the top-left corner of a block.

When the SAV of an interpolated block is larger than a threshold, the motion vector of this interpolated block is considered as a global motion vector for the next interpolated frame. In our approach, the global motion, $\overrightarrow{mv_{global}}$ is processed and we also consider about the localized global motion vector, $\overrightarrow{mv_{local}}$,

because every current interpolated block has its own localized global motion vector. These localized global motion vectors are useful for deciding which motion vectors are accurate, and are therefore considered with higher probability for true motion. Spatial information from the processed blocks is also considered when searching, and motion vectors from neighboring blocks have higher probability of being the true motion. The proposed method combined with the PSAD is formulated by

$$PSAD(\overrightarrow{v}) = \begin{cases} \sum\limits_{x \in B} |f(x - \overrightarrow{v}, t) - f(x + \overrightarrow{v}, t + 1)| - T_{global} & when \quad \overrightarrow{v} = \overrightarrow{mv_{global}} \\ \sum\limits_{x \in B} |f(x - \overrightarrow{v}, t) - f(x + \overrightarrow{v}, t + 1)| - T_{local} & when \quad \overrightarrow{v} = \overrightarrow{mv_{local}} \\ \sum\limits_{x \in B} |f(x - \overrightarrow{v}, t) - f(x + \overrightarrow{v}, t + 1)| & otherwise \end{cases} \tag{3.5}$$

The proposed method adopts an SAD-like method, PSAD, to address the complexity issue. The PSAD of $\overrightarrow{mv_{global}}$ or $\overrightarrow{mv_{local}}$ is subtracted by the threshold value, $T_{global}$ or $T_{local}$, and compared with the PSAD values of other motion vectors. The advantage of this proposed method is prevention of the bidirectional searching method from falling into a *background trap*. Background trap indicates a situation where an incorrect motion vector is determined from two very similar background blocks. Although unidirectional searching may also find an incorrect motion vector from two similar blocks, the problem in the bidirectional search is more serious. The proposed method introduces these two reference vectors, $\overrightarrow{mv_{global}}$ and $\overrightarrow{mv_{local}}$, and two threshold values, $T_{global}$ and $T_{local}$, to efficiently prevent this trap.

### 3.2.5 Sub-block Motion Assignment and Motion Refinement

According to analyzed temporal information, our method explores true motion candidates and then refines the accuracy of true motions for blocks or sub-blocks. Not only does this temporal information determine which motion candidate has higher probability, but it also provides criteria to assign true motion for sub-blocks in order to enhance visual quality and reduce occlusion problems and halo artifacts.

To reduce computation, our proposed method adopts block-based motion

assignment and HD downsampling. Block motion assignment will be used for a 2N×2N block (by default N=8) after upsampling recovery. However, when a moving objects is smaller than one block or an object boundary crosses a block, occlusion problems and halo artifacts obviously appear. To enhance motion precision and reduce computation, each block is assigned up to four different motions to deal with object boundary issues. Fig. 3.9 shows that the proposed sub-block assignment method examines motions from the surrounding motions of the previous interpolated frame. At first, all sub-block motions, $\overrightarrow{C\_SMV_i}$ when $i = 0..3$, are assigned the same motion as the original block motion, $\overrightarrow{C\_MV}$, for the default motions. These default motions, $\overrightarrow{P\_MV_i}$ when $i$ equals 0 to 8, are checked with the SAD function by N×N matching blocks. If the SAD of preceding motion is smaller than the SAD of the current assigned motion, a new motion will be assigned to this sub-block as the new true motion. The sub-block motion assignment method is formulated by

$$\overrightarrow{C\_SMV_i} = \arg\min_{\overrightarrow{v} \in S'} \sum_{x \in B} |f(x - \overrightarrow{v}, t) - f(x + \overrightarrow{v}, t + 1)| \tag{3.6}$$

where $B$ denotes a matching N×N block of the current interpolated position; $S'$ is a set of motion candidates, including $\overrightarrow{C\_MV}$ and $\overrightarrow{P\_MV_i}$ when $i$ equals 0 to 8; $\overrightarrow{C\_SMV_i}$ is the sub-block motion vector examined for the best matching. Two adjacent frames are denoted by *f(x,t)* and *f(x,t+1)*, where $x$ and $t$ are spatial and time domain indices.

After determining the true motion, the proposed method performs the final refinement procedure. This procedure works on the original frames. The refining window is [-1, +1] shown in Fig. 3.10. After refining, this motion vector will be used for the interpolation.

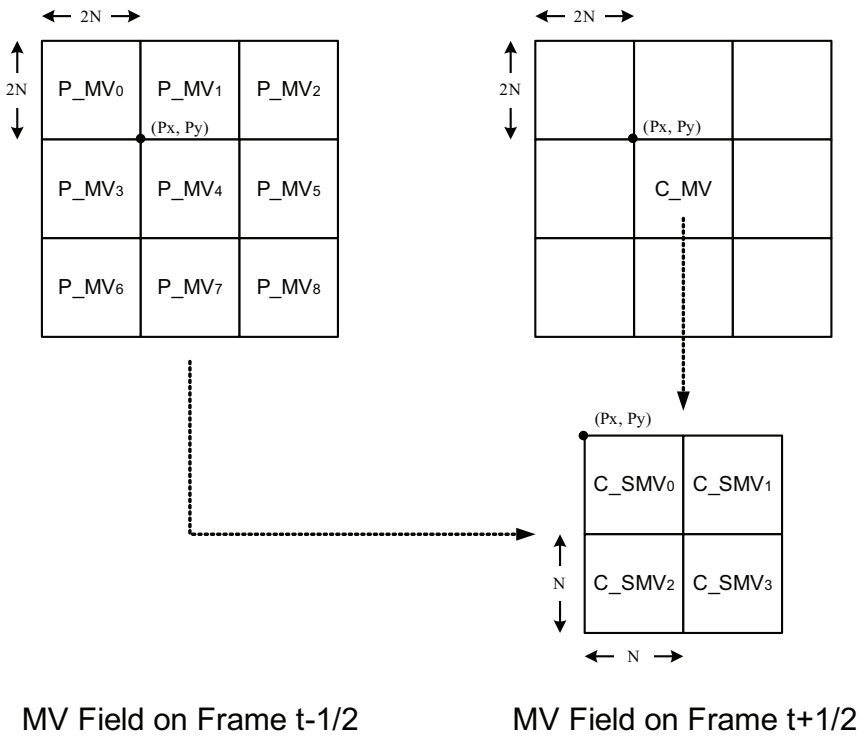



Figure 3.9: Motions on the previous frame and the current block motion are employed on sub-block assignment.

## 3.3 Frame Interpolation

### 3.3.1 Frame Motion Skip

Motion Compensated Frame Interpolation is an efficient method to improve the visual quality of video by increasing the frame rate. However, the performance of true motion estimation varies with the image quality, the size of moving objects, and motion velocity. When the true motion engine cannot correctly find the true motion, irregular motions can cause blocking artifacts and provide incorrect spatial and temporal information for the following frame. If the frame interpolator generates an interpolated frame based on these incorrect motions, visual quality might be much worse than the case where MCFI is not applied.

Similar to [25], the proposed method employs an algorithm for the adaptive frame motion skipper. The proposed skipping method considers the level of block matching difference and motion consistency. The frame skip ratio (*Skip_Ratio*) is

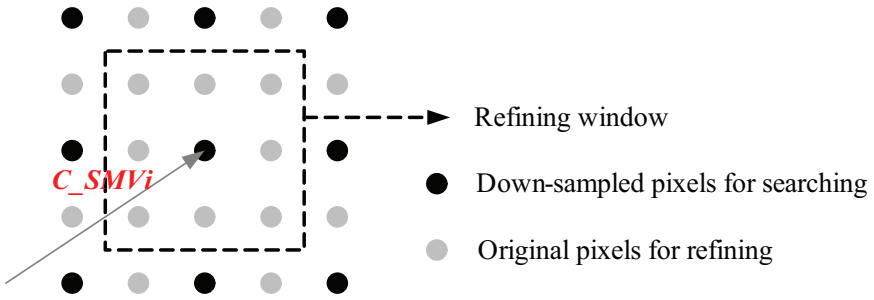



Figure 3.10: Motion vector refining window

formulated by

$$Skip\_Ratio(t) = \frac{\sum_{i=1}^{BH}\sum_{j=1}^{BW} Skip\_Factor(i,j)}{BH \times BW} \tag{3.7}$$

where the skip ratio of the current frame $t$ is denoted by *Skip_Ratio(t)*. BH and BW denote the total block number on the height and the width of the current frame $t$. If *Skip_Ratio(t)* is larger than a threshold value, $T_{skip}$, all motion information will be discarded, and the interpolator will repeat the previous frame $t$ or average frame $t$ and frame $t+1$. The skip factor of each block (*Skip_Factor*) is formulated by

$$Skip\_Factor(i,j) = \begin{cases} 1 & if \quad SAD(i,j) < SAD_{skip} \\ 1 & if \quad \overrightarrow{MV_{i,j}} \neq any \quad \overrightarrow{MV_{neighboring}} \\ 0 & otherwise \end{cases} \tag{3.8}$$

### 3.3.2 Interpolation

After determining the true motion vector for a current interpolated block, interpolation should be performed if frame skipping is disabled. Two adjacent frames are denoted by *f(x,t)* and *f(x,t+1)*, where $x$ and $t$ are spatial and time domain indices respectively. The motion vector field of a frame is denoted by $\overrightarrow{V}$. A motion vector is denoted by $\overrightarrow{v}$ for an N×N interpolated block. Motion-compensated averaging including boundary cases can be used as follows:

$$\begin{cases} f(x,t+\frac{1}{2}) = \frac{1}{2}\left[f(x-v,t) + f(x+v,t+1)\right] & if \quad (x-v) \in V \quad and \quad (x+v) \in V \\ f(x,t+\frac{1}{2}) = \frac{1}{2}\left[f(x,t) + f(x,t+1)\right] & if \quad (x-v) \notin V \quad and \quad (x+v) \notin V \\ f(x,t+\frac{1}{2}) = f(x-v,t) & if \quad (x-v) \notin V \quad and \quad (x+v) \in V \\ f(x,t+\frac{1}{2}) = f(x+v,t+1) & otherwise \end{cases} \tag{3.9}$$

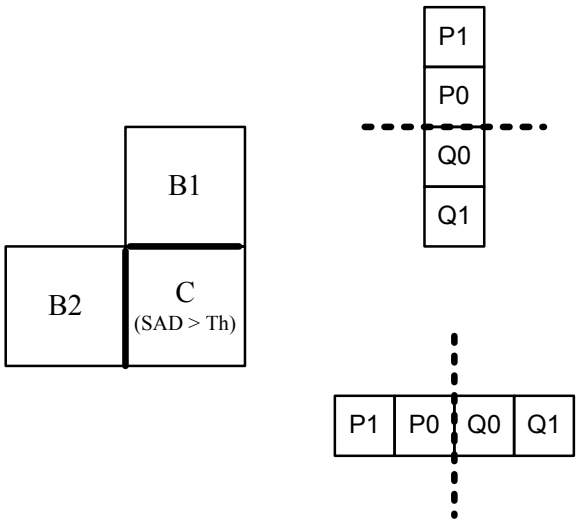



Figure 3.11: Two edges of the current interpolated block C are filtered.

### 3.3.3 Deblocking Filter with Block Difference

A deblocking filter is used subsequent to interpolation. Whether an edge should be filtered depends on the SAD of the current interpolated block and the top or left neighboring interpolated block. If the difference is larger than a threshold, the edges of this block should be filtered. In the proposed method, a simple filter is applied, which only considers one pixel on each side of a filtered edge, as shown Fig. 3.11. The SAD of the current interpolated block C is larger than a predetermined threshold, *Th*, so there are two edges of this block which should be filtered with a deblocking operation. If a pixel needs to be filtered twice, a vertical filtering procedure should be performed and a horizontal filtering procedure follows. This operation is performed as follows:

$$
\begin{aligned}
P0' &= (P1 + 2P0 + Q0) >> 2 \\
Q0' &= (Q1 + 2Q0 + P0) >> 2 \\
P1' &= P1 \\
Q1' &= Q
\end{aligned}
\tag{3.10}
$$

## 3.4 Acknowledgement

Portions of Chapter 3 appear in "Method and Architecture Design for Motion Compensated Frame Interpolation in High-Definition Video Processing," Yen-Lin Lee and Truong Nguyen, in Proceedings of the *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2009; "Fast One-pass Motion Compensated Frame Interpolation in High-Definition Video Processing," Yen-Lin Lee and Truong Nguyen, in Proceedings of the *IEEE International Conference on Image Processing (ICIP)*, Nov. 2009. The dissertation author was the primary author of these publications, and the listed co-author directed and supervised the research that forms the basis for this chapter.

# 4 The Proposed MCFI Architecture

For real-time Frame Rate Up Conversion with motion re-estimation, searching the proper motion candidates and determining one true motion is a demanding task. In order to lower the computational complexity, the proposed architecture introduces a fast full search method, Multi-level Successive Elimination Algorithm (MSEA). This method is neither a conventional full search nor a fast search with lower matching accuracy. MCFI is more sensitive to motion accuracy than is motion estimation for video compression. In this section, a practical architecture is proposed to implement all proposed methods mentioned in previous sections. The proposed architecture can process Frame Rate Up Conversion for a 1080p sequence from 30fps to 60fps.

## 4.1 System Architecture

The proposed system architecture contains several modules, shown in Fig. 4.1. This architecture implements and simplifies the proposed processing flow shown in Fig. 3.1. In this figure, there are two kinds of elements: the white blocks are functional elements, including motion estimator, motion compensator, spatial parameter generator, temporal parameter generator, deblocking filter, and system memory controller. The gray blocks are storage elements, including system parameters, reference buffer, temporal parameter buffer, motion vector field buffer, and filtering buffer.

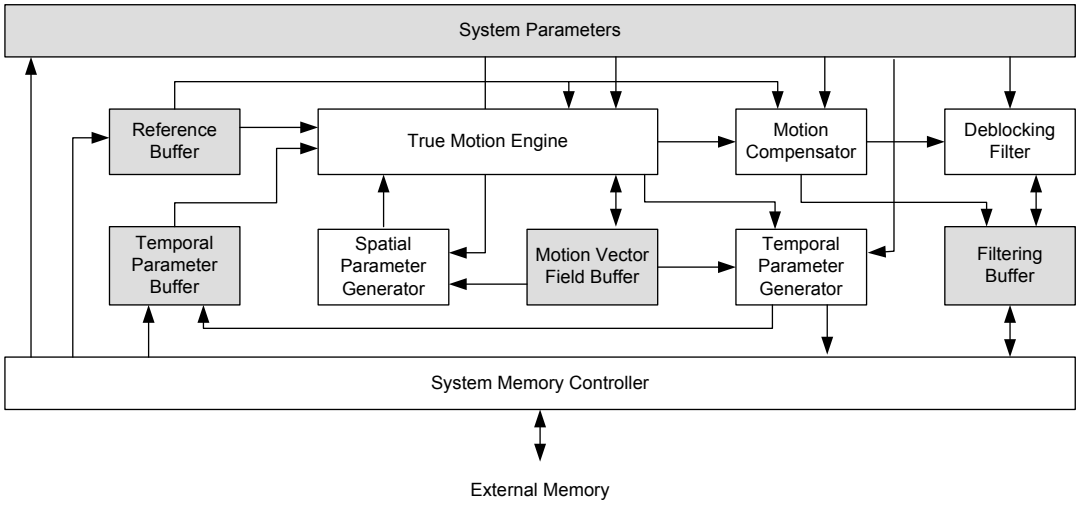



Figure 4.1: System block diagram of the proposed architecture.

Considering the memory usage, there are three levels of memory access strategy: system memory access, on-chip memory access, and local memory access in Fig. 4.2. The System Memory Controller pre-fetches data from the external memory for MCFI processing procedures. This data contains: system parameters, reference data, temporal information parameters, and filtering pixels. System parameters include the settings for the current input sequence, conditional threshold values, starting addresses of reference data, and the MCFI processing parameters. The reference buffer temporarily stores the reference pixels for the true motion engine, which searches true motion vectors for the current interpolated frame, and the motion compensator, which constructs interpolated images. The temporal parameter buffer stores information for global motion vector and localized motion vectors from the previous interpolated frame. Global motion vectors are generated by the Temporal Parameter Generator and are delivered to the temporal parameter buffer when an interpolated frame is finished. Each localized global motion vector for an interpolated block is generated by the Temporal Parameter Generator when sufficient motion information is available. These localized vectors are stored in external memory and are read back when the true motion engine searches for the next interpolated frame. The filtering buffer stores the neighboring pixels to the left of and above the current block. The line buffer for pixels above the

current block can be designed with on-chip memory so as to reduce external memory loading, thereby enhancing performance. Otherwise, the neighboring pixels will be pre-fetched from the external memory for the filtering procedure. The motion vector field buffer stores motion vector information in the spatial domain. In the proposed architecture, the motion vectors for two rows of blocks are stored in this buffer in order to keep the information for immediately determining the true motion.

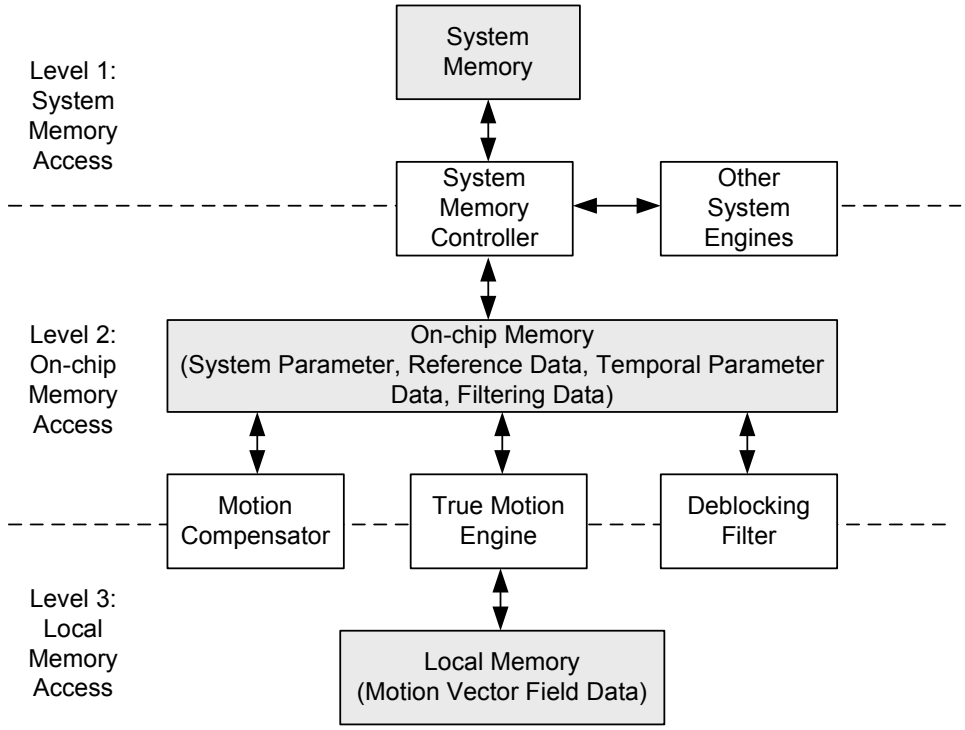



Figure 4.2: Three levels of memory access for the proposed architecture.

## 4.2 True Motion Engine

The true motion engine is the core of the MCFI algorithm. It searches for a true motion for motion compensation based on two reference frames and motion information in the temporal and spatial domains. Temporal information is used to access information from more than two successive reference frames. Fig. 4.3

shows a block diagram of the proposed true motion engine.

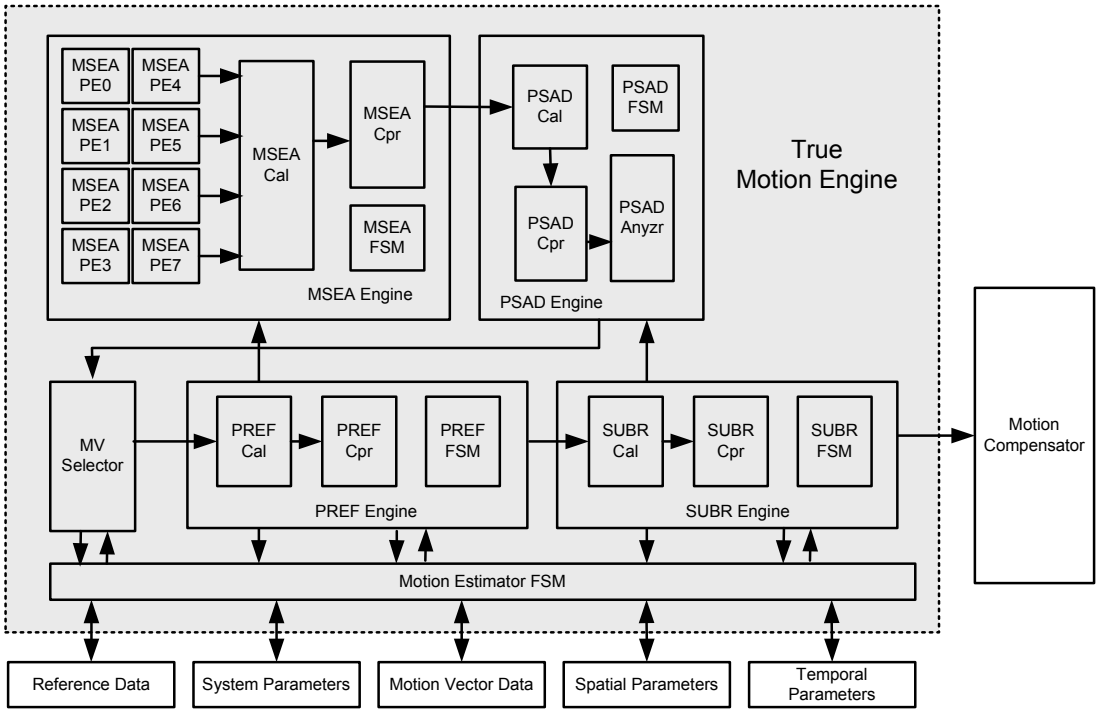



Figure 4.3: Block diagram of the proposed true motion engine.

Five processing steps are performed in the true motion engine. In the implementation, nine-grids classification is used with a search window from -7 to +7 pixels (15×15) for a down-sampled frame, 8×8 block size, 16×16 enlarged block size, and four candidates for each MSEA sub-block (36 candidates for an entire block are considered). The processing flow of the proposed true motion engine is shown in Fig. 4.4. HD down-sampling is the first step, which drops all odd pixels in two dimensions, as shown in Fig. 4.5. This is performed for high resolution video, such as 720p and 1080p. There is no low-pass filter applied when downsampling because there are several search stages in the proposed architecture, and it will perform a refinement later on non-downsampled images. Hence, this is a simple and fast method to select all even pixels for two dimensions when performing a rough search. In the implementation phase, all pixels of the reference frames are stored in the reference buffer, and MSEA reads all even pixels when searching. Moreover, all reference pixels are used when performing refinement.

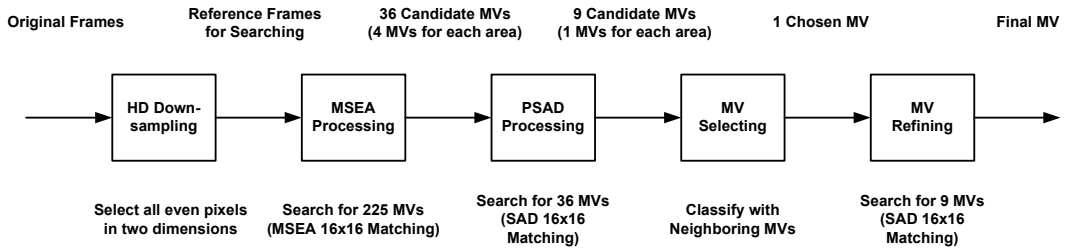



Figure 4.4: Processing flow of the proposed true motion engine.

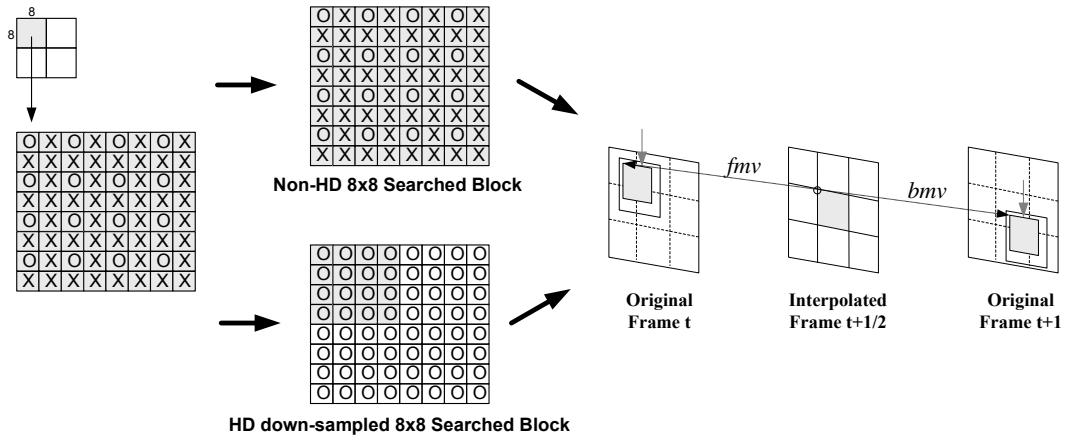



Figure 4.5: HD downsampling - drop all odd pixels in two dimensions.

The second step of the proposed architecture is the Multi-level Successive Elimination Algorithm (MSEA), which is a fast full search method. Although full search achieves the most accurate result, it is more computationally expensive than a fast search. Fast search is an accepted implementation method for video compression applications because it dramatically reduces the computational complexity while slightly increasing the bitrate for approximate quality. However, the goal of Motion Compensated Frame Interpolation is to find the true motion distribution and not the best matching result with lowest bitrate. Based on our experiments, fast search methods such as three-step search [34] and diamond search [35] are all unsuitable for MCFI due to the local trap problem. MSEA is a compromise between the full search and the fast search. An MSEA example is shown in Fig. 4.6, and the equations for two 16×16 matching blocks of SAD, MSEA, and SEA are formulated by

$$
\begin{aligned}
SAD(m,n) &= \sum_{i=0}^{15}\sum_{j=0}^{15} |f_{t+1}(i-m, j-n) - f_t(i+m, j+m)| \\
&\geq \sum_{i'=0}^{3}\sum_{j'=0}^{3} \left| \sum_{i=0}^{3}\sum_{j=0}^{3} f_{t+1}(i+4i'-m, j+4j'-n) - \sum_{i=0}^{3}\sum_{j=0}^{3} f_t(i+4i'+m, j+4j'+m) \right| \\
&\equiv \sum_{i'=0}^{3}\sum_{j'=0}^{3} \left| SSB_{t+1(i',j')}(-m,-n) - SSB_{t(i',j')}(m,n) \right| \equiv MSEA(m,n) \\
&\geq \left| \sum_{i=0}^{15}\sum_{j=0}^{15} f_{t+1}(i-m, j-n) - \sum_{i=0}^{15}\sum_{j=0}^{15} f_t(i+m, j+m) \right| \\
&\equiv |SB_{t+1}(-m,-n) - SB_t(m,n)| \\
&\equiv SEA(m,n)
\end{aligned}
\tag{4.1}
$$

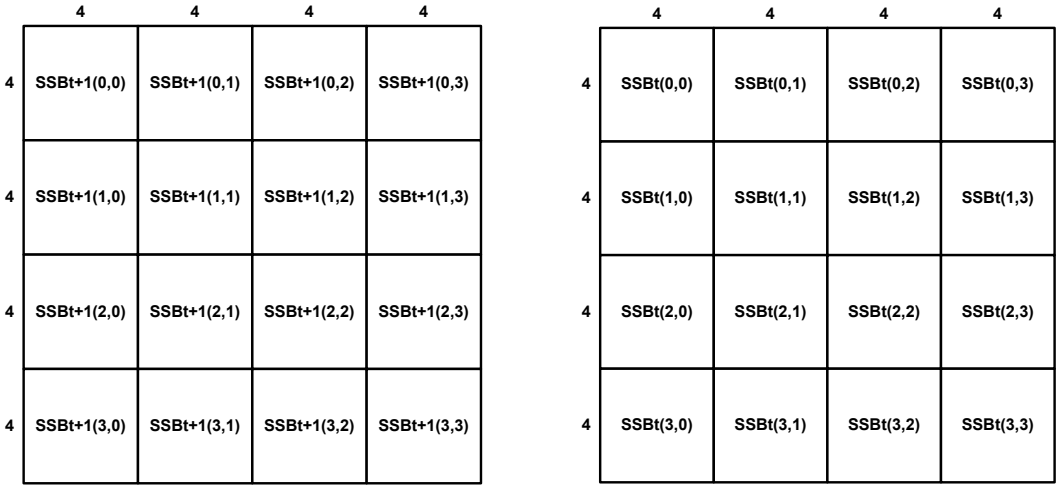



Figure 4.6: Multi-level Successive Elimination Algorithm (MSEA).

Due to nine-grids classification, the MSEA engine separately searches four motion candidates with small MSEA values from the complete twenty-five (5×5) vectors for each area, which is shown in Fig. 4.7. There are a total of thirty-six motion candidates after a search by the MSEA engine, and each matching block is divided into sixteen 4×4 sub-blocks for the MSEA calculations. A nine-directional enlarged matching method, shown in Fig. 3.3, is used by the MSEA

engine. Fig. 4.8 demonstrates the architecture of the proposed MSEA engine, which employs eight Processing Elements (PEs) to deal with the MSEA search in an HD application. MSEA searching patterns for a previous frame $t$ and successive frame $t$+1 are shown in Fig. 4.9(a) and (b). Fig. 4.9(c) demonstrates how to add and remove pixels to get a new Sum of Sub-Block (SSB) when the search engine shifts vector positions. Based on this shifting technique, the proposed engine is faster and less complex than a full search.

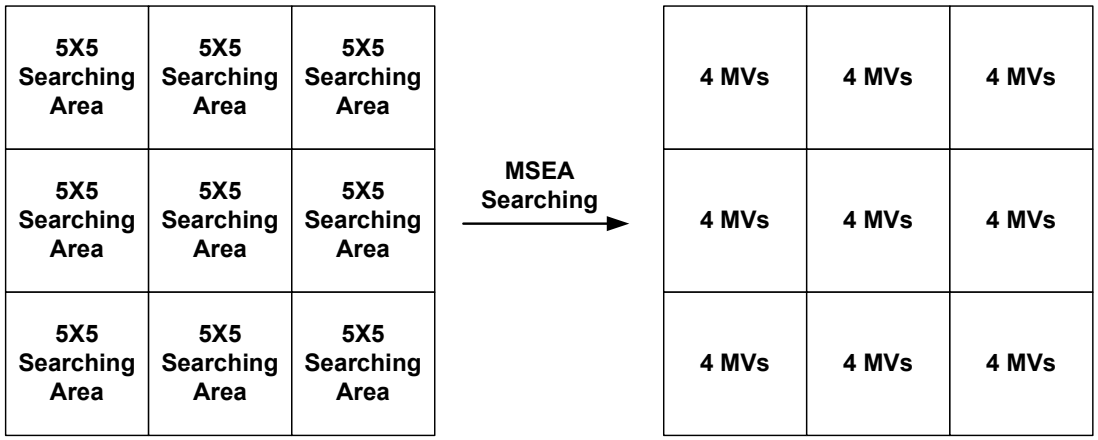



Figure 4.7: MSEA searches four true motion candidates for each area.

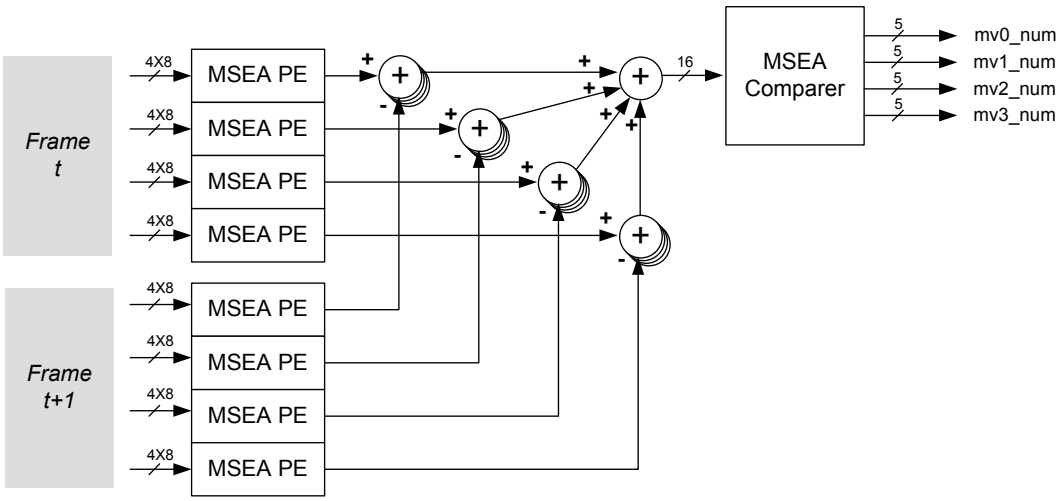



Figure 4.8: Architecture of the proposed MSEA engine.

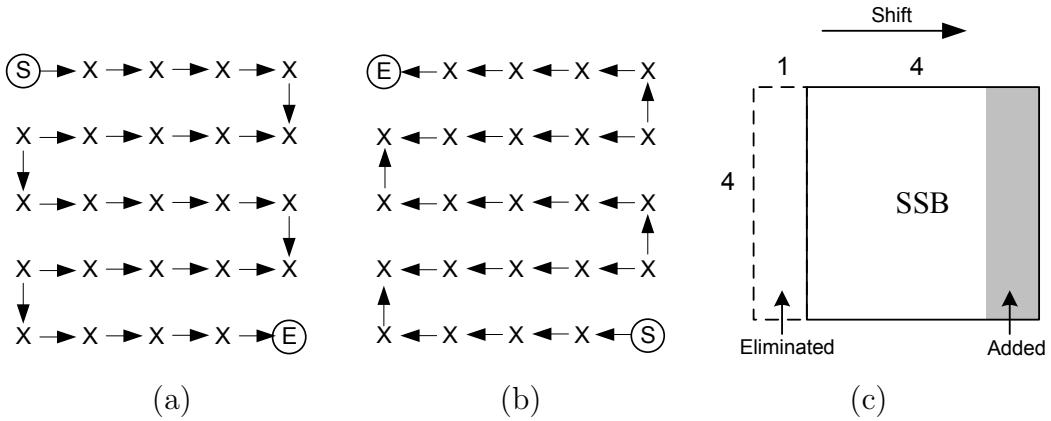



Figure 4.9: (a) Searching order on an area for a previous frame $t$. (b) Searching order on an area for a successive frame $t$+1. (c) Added and eliminated areas when shifting a MSEA vector position.

The third step of the true motion engine is to search by the Probable Sum of Absolute Difference (PSAD) mentioned in Eqn. (3.5). Since the MSEA engine decides four highly probable candidates for each directional area, the PSAD engine only needs to check the PSAD values for these four vectors as opposed to a full search with the SAD. The direct multi-directional enlarged matching method, shown in Fig. 3.5, is used by the PSAD engine. As a result of this technique, each area produces a most probable true motion candidate for a search window. That is, there are in total nine motion candidates with different directions, which are shown in Fig. 3.7. The fourth step is a true motion selection from these nine motion candidates. Effective neighboring blocks for spatial information are shown in Fig. 3.6. The proposed method pre-defines conditions to determine a true motion vector for each block from all available information. At the last stage, the proposed architecture performs a refining procedure by the Probable Refining Engine (PREF) if the input video sequence has high resolution and is down-sampled before searching. The refining window is shown in Fig. 3.10. This refinement enhances the accuracy of motion vectors due to down-sampling the searching pixels. A SAD method is used for the PREF.

The proposed pipeline scheduling is demonstrated in Fig. 4.10. The Fetch command obtains necessary information from external memory at the first stage.

MSEA and PSAD operate at the same pipeline stage following the Fetch stage. The motion selection (MV_Sel) stage is postponed until the information from N6 (shown in Fig. 3.6) is available. A refining procedure by PREF is performed at the same stage as true motion selection. The interpolating (Interpolation) and deblocking filtering (Deblock) procedures are performed at the same stage after the refinement. Finally, there is a write-back (Write Back) procedure to write interpolated block and temporal information to the external memory for display and successive interpolation procedures.

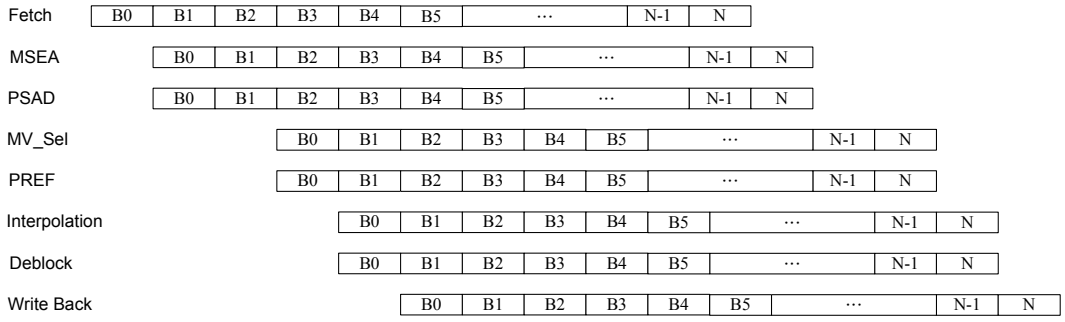



Figure 4.10: Pipeline scheduling for the proposed architecture.

## 4.3 Acknowledgement

Portions of Chapter 4 appear in "Method and Architecture Design for Motion Compensated Frame Interpolation in High-Definition Video Processing," Yen-Lin Lee and Truong Nguyen, in Proceedings of the *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2009; "Fast One-pass Motion Compensated Frame Interpolation in High-Definition Video Processing," Yen-Lin Lee and Truong Nguyen, in Proceedings of the *IEEE International Conference on Image Processing (ICIP)*, Nov. 2009. The dissertation author was the primary author of these publications, and the listed co-author directed and supervised the research that forms the basis for this chapter.

# 5 MCFI Implementation and Experimental Results

## 5.1 Implementation

The specification for the proposed architecture is capable of doubling the frame rate for an HD1080p 30fps video at 180MHz or a 720p 30fps video at 83MHz. To verify the accuracy and efficiency of the proposed method and architecture, the architecture is designed in VHDL ((VHSIC (Very High Speed Integrated Circuits) Hardware Description Language)) and implemented with TSMC (Taiwan Semiconductor Manufacturing Company) 90-nm technology. The implemented architecture operates with our VHDL and MATLAB-model, and the result has been verified with our prototype in MATLAB-model. TABLE 5.1 shows logic gate count including all functional blocks of the proposed architecture synthesized with the Cadence RTL compiler at 180 MHz. The total logic gate count without on-chip memory is about 28K. TABLE 5.2 summarizes the distribution of the on-chip memory resources. The proposed architecture requires 22.67 KB of on-chip memory during operation. The Reference Buffer occupies 62.88% of memory because there are up to five overlapped searching window areas stored in this buffer from previous frame $t$ and successive frame $t$+1. The operation of this buffer is explained in Fig. 4.10. When the Motion Compensator and Deblocking Filter process Block 0, the Memory Controller starts to fetch reference data and parameters for Block 4. Hence, reference data buffered for five consecutive blocks is necessary to reduce the external memory bandwidth and the processing period. The Motion Vector

Field Buffer keeps two rows plus two blocks of motion information in a current interpolated frame. This buffer has to store motion information for 3,842 blocks. The effective neighboring block for spatial information is shown in Fig. 3.6.

Table 5.1: Front-end Hardware Cost of the Proposed True Motion Engine with TSMC 90-nm Technology

| Functional Block | Gate Count |
|---|---|
| MSEA Engine | 19,559 |
| PSAD Engine | 4,055 |
| True Motion Selector | 2,125 |
| PREF Engine | 1,466 |
| Glue Logic | 792 |
| Total | 27,997 |

Table 5.2: On-chip Memory Buffer Size of the Proposed Architecture for a 1080p video

| Buffer | Size (KB) |
|---|---|
| Reference Buffer | 14.256 |
| Temporal Parameter Buffer | 0.026 |
| Motion Vector Field Buffer | 7.684 |
| Filtering Buffer | 0.704 |
| Total | 22.670 |

Table 5.3: Processing Time of the Proposed Architecture

| Type | Cycles | Frequency (MHz) | Time (ms) | Require (ms) |
|---|---|---|---|---|
| 1 16×16 Block | 683 | 180/83 | - | - |
| 1 Frames @ 1080p | 5,581,795 | 180 | 30.69 | 33.33 |
| 30 Frames @ 1080p | 167,453,850 | 180 | 921.99 | 1000 |
| 1 Frames @ 720p | 2,462,755 | 83 | 29.55 | 33.33 |
| 30 Frames @ 720p | 73,882,650 | 83 | 886.59 | 1000 |

Based on our implementation, we simulate this architecture with real video sequences so as to calculate the cycle counts and verify the processing time that meets timing requirement for HD1080p 30 fps. Since the processing cycle count is fixed for the proposed design, the cycles in TABLE 5.3 can be analyzed for any sequence with the same resolution. From TABLE 5.3, the proposed architecture

Table 5.4: External Memory Bandwidth Requirement for the Proposed Architecture

| Type | Reference Data Fetch (MB) | Parameter Fetch (KB) | Data Write-back (MB) | Total (MB) |
|---|---|---|---|---|
| 1 Frames @ 1080p | 17.19 | 16.37 | 3.13 | 20.34 |
| 30 Frames @ 1080p | 515.89 | 491.10 | 94.00 | 610.38 |
| 1 Frames @ 720p | 8.01 | 7.25 | 1.38 | 9.40 |
| 30 Frames @ 720p | 240.16 | 217.50 | 41.472 | 281.85 |

Table 5.5: Performance Comparison - PSNR

| Method | CITY@720p | CREW@720p |
|---|---|---|
| Bilinear | 28.95 dB | 28.64 dB |
| 3DRS | 29.44 dB | 28.77 dB |
| PPC | 29.55 dB | 28.40 dB |
| Proposed | 32.42 dB | 28.95 dB |
| Proposed_SubMV | 32.50 dB | 28.97 dB |
| Proposed_FS | 32.49 dB | 28.97 dB |
| Proposed_DS | 29.22 dB | 28.94 dB |

demonstrates its capability for a 1080p 30fps video at 180 MHz or a 720p 30fps video at 83 MHz. TABLE 5.4 shows the requirement of the external memory bandwidth. The proposed architecture requires 610.38 MB/s for a 1080p video or 281.85 MB/s for a 720p video so as to double the frame rate from 30fps.

## 5.2 Performance Result

Here, we compare the performance of the proposed algorithm by testing high-resolution test sequences with three other methods: bilinear interpolation, 3-D Recursive Search (3DRS) [5], and Phase-Plane Correlation (PPC) [17] with the same searching block size of 16×16. We also compare the proposed methods with different search strategies including MSEA (Proposed), Full Search (Proposed_FS), and Diamond Fast Search (Proposed_DS). For 60fps test sequences, such as CREW@720p and CITY@720p, the test drops all even frames and generates these missing frames by four different methods. These interpolated frames are compared with the original frames so as to calculate PSNR results. Exper-

imental results show that the proposed algorithm provides better video quality in terms of PSNR than conventional methods. The results also show that the PSNR of the proposed method with MSEA is similar to Full-Search and superior to Diamond Fast-Search. In addition to PSNR comparisons, we also show visual quality comparisons with HD sequences including PRODUCERS(1440×960) and FLIGHT(1080p) at 24fps. We test visual quality by conducting subjective tests with human observers. The results demonstrate that the proposed method has better visual quality than other conventional methods.


Figure 5.1: PSNR comparison 1 - CREW 1280×720 60fps

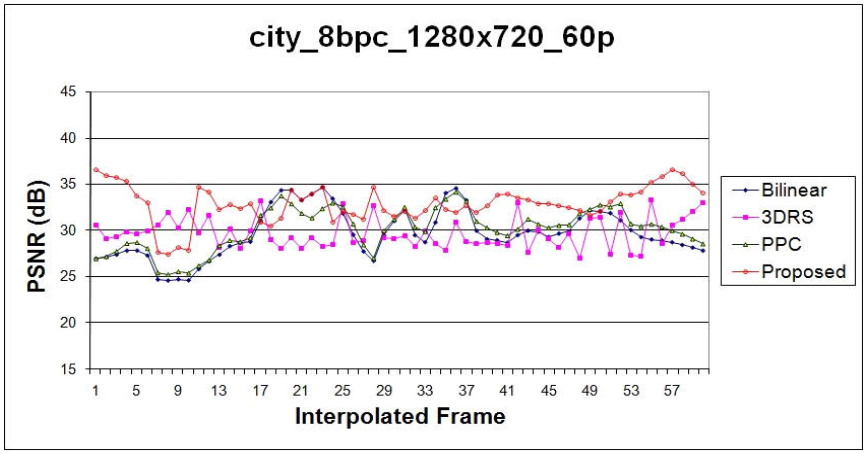


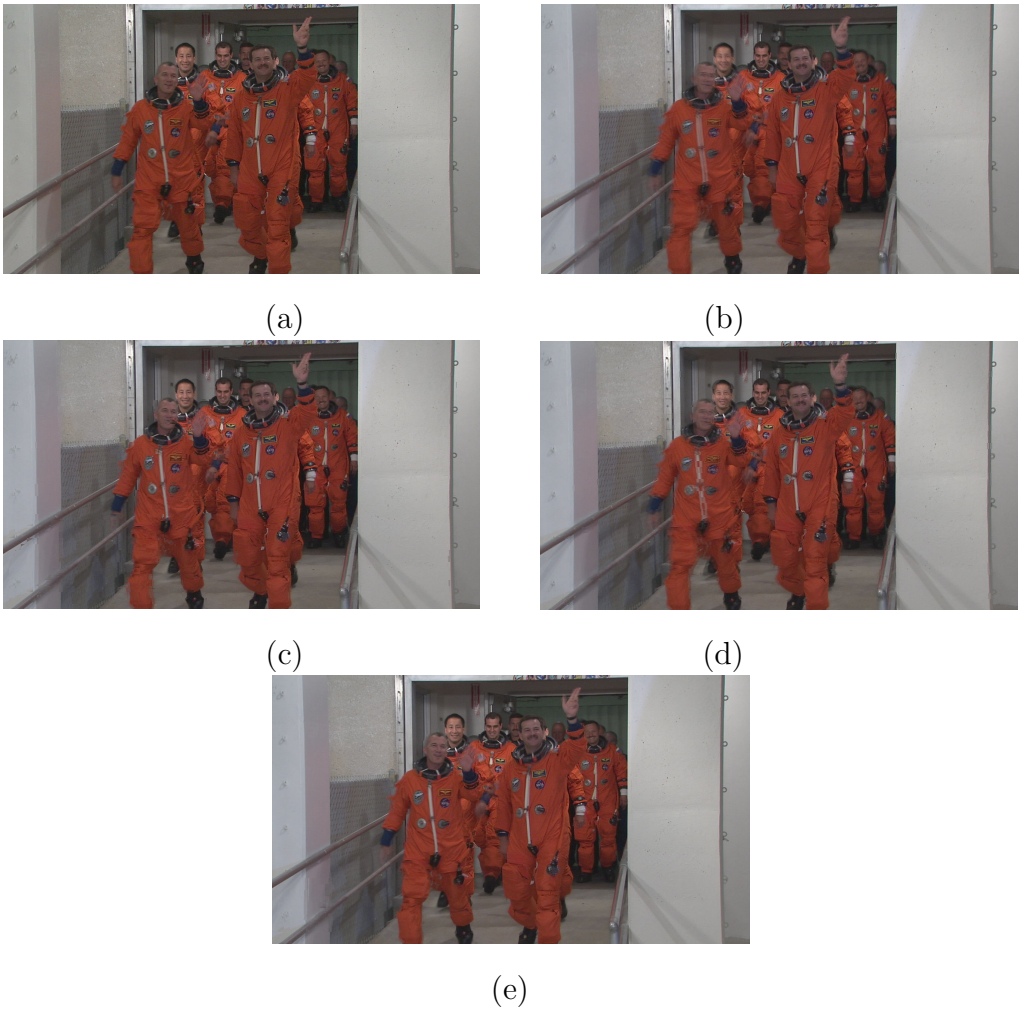

Figure 5.2: PSNR comparison 2 - CITY 1280×720 60fps

The PSNR results are organized in TABLE 5.5 and also shown in Fig.

5.1 and 5.2. In TABLE 5.5, we compare PSNR results with three conventional methods: bilinear, 3DRS, and PPC. The average PSNR of the proposed method is superior to competing methods. Our proposed method has better visual quality compared to other conventional methods even in frames with similar PSNR. In TABLE 5.5, we compare the PSNR with our proposed algorithm with block mv assignment (Proposed) or sub-block mv assignment (Proposed_SubMV) using two different search strategies including full search (Proposed_FS) and diamond fast search (Proposed_DS). The PSNR results between Proposed and Proposed_FS are very close, with Proposed_FS performing slightly better. However, although the PSNR of Proposed_DS is similar to the other two proposed method in CREW, it is obviously worse in CITY. In our experiments, Proposed_DS has more broken objects and annoying artifacts because of searching two similar matching blocks at an early stage and falling into a local trap.

A visual comparison example for CREW is shown in Fig. 5.3 and 5.4. Interpolated frame 185 is generated with different methods, and the PSNR results are very close. They are 21.2462 dB (bilinear), 21.3463 dB (3DRS), 21.2293 dB (PPC), and 21.3307 dB (PPC), respectively. The PSNR value dramatically decreases in this interpolated frame due to the flashlight. Extreme color change increases the difficulty of true motion search, so an accurate searching algorithm is very important in order to keep visual quality with lower PSNR. We zoom in on the face of one team member in Fig. 5.4 so as to clearly compare the difference. The bilinear method results in a blurred face and ghost artifacts. 3DRS causes a broken face due to large color change. PPC generates a blurred and broken face because it is difficult to find the correlation between two temporally dissimilar blocks. In Fig. 5.4(e), the result of the proposed method keeps the face intact and clear although there are still some artifacts surrounding the face. This is because the proposed method adopts a block matching method to calculate motion vector and meets an occlusion situation here. Our method attempts to keep the foreground intact by using the multi-directional enlarged matching algorithm, but the quality of the background suffers. From the experiments, viewers are more sensitive to a moving object than a still background. Hence, the proposed method provides a better

visual quality.



(a) (b)

(c) (d)

(e)

Figure 5.3: Interpolated frame 185 in CREW (a) original frame, (b) bilinear interpolation (21.2462 dB), (c) 3D Recursive Search (21.3463 dB), (d) Phase Plane Correlation (21.2293 dB dB), (e) proposed method (21.3307 dB).

Another visual comparison example for CREW is shown in Fig. 5.5. Interpolated frame 117 is generated with different methods, yielding similar PSNR results. They are 34.287 dB (bilinear), 37.756 dB (3DRS), 33.418 dB (PPC), 38.237 dB (proposed method without sub-block assignment), and 38.301 dB (proposed method with sub-block assignment), respectively. We zoom in for frame 117 on the hand in Fig. 5.6 in order to clearly compare the difference.

Fig. 5.7 and 5.8 show another example for visual comparison. A tower

(a) (b) (c) (d) (e)

Figure 5.4: Zoom in for interpolated frame 185 in CREW (a) original frame, (b) bilinear interpolation, (c) 3D Recursive Search, (d) Phase Plane Correlation, (e) proposed method.

slowly rotates in the middle of the scene while the background quickly translates. This causes several occlusions to occur. In Fig. 5.7, the proposed method yields better results than the other methods, especially for the consistency of the background. In Fig. 5.7(c)-(d), some buildings are broken because similar background regions create matching error. The proposed method introduces localized global motions and assigns each block a global motion vector to promote motion consistency and also different global motion in each area. We zoom in on a part of the tower in Fig. 5.8 so as to clearly compare the difference. The bilinear method generates a blurred image and ghost artifacts. 3DRS and PPC result in broken structures for windows of a building near the main tower. Here, the proposed method provides better visual quality as well as achieving higher PSNR.

We perform more visual tests on HD video sequences. PRODUCERS is a clip which pans from left to right. The difficulty of this sequence is that it does not have one global motion: objects near the camera move faster and objects farther from the camera move more slowly. There are also significant occlusions in this sequence. This sequence tests our localized global motion and motion accuracy. FLIGHT is a clip that shows a scene going through a river like a flying bird. Two sides of the river move in different directions, and some seabirds appear in the scene. Fig. 5.9 and 5.10 show a visual comparison for PRODUCERS, and Fig. 5.11 and 5.12 show a visual comparison for FLIGHT. From our experiments, the proposed method demonstrates superior performance to conventional methods.

For comparing our proposed method with other conventional algorithms

in visual quality, we conducted a perceptual test using human observers. This subjective test was conducted in a double blind manner according to the single stimulus non-categorical judgment method described in [36]. Each viewer watches both the original version with original frame rate (24 fps or 30fps) and up-converted version with double frame rate using different methods. Viewers are asked to rate which one they preferred and select a score on a continuous scale between [-3, 3] where positive responses indicate a preference for the up-converted clip. Twenty observers viewed five tests each on a Samsung 2443BW LCD with 5 ms response time. Each test includes four different up-converted clips with different methods including bilinear, 3DRS, PPC, and the proposed method.

TABLE 5.6 shows the average scores and standard deviations for each test and each method across all twenty viewers. The results suggest that there were indeed perceptual improvements for the proposed method in NATIONAL TREASURE (1440×960), FLIGHT, and PRODUCERS, all of which contained fast global motions or fast large moving objects. For CREW and CITY, the results show that there were only slight improvements because there are too many details in the screen and reviewers were distracted by these complicated images. However, human observers are sensitive to obvious artifacts and broken objects. If artifacts exist in a proposed clip, people prefer the original clips without processing. Consequently, the results demonstrate that the proposed method has better visual quality than other conventional methods in our subjective experiment.

## 5.3 Acknowledgement

Portions of Chapter 5 appear in "Method and Architecture Design for Motion Compensated Frame Interpolation in High-Definition Video Processing," Yen-Lin Lee and Truong Nguyen, in Proceedings of the *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2009; "Fast One-pass Motion Compensated Frame Interpolation in High-Definition Video Processing," Yen-Lin Lee and Truong Nguyen, in Proceedings of the *IEEE International Conference on Image Processing (ICIP)*, Nov. 2009; "Novel Method and Architecture Design for Motion

Table 5.6: Results of the perceptual experiment

| Test 1 - CITY | Average Score | Standard Deviation |
|---|---|---|
| Original 60fps | 1.81 | 0.65 |
| Bilinear | -0.20 | 0.95 |
| 3DRS | -1.29 | 1.20 |
| PPC | -1.23 | 1.11 |
| Proposed | 0.75 | 1.16 |
| Test 2 - CREW | Average Score | Standard Deviation |
| Original 60fps | 1.18 | 0.79 |
| Bilinear | -0.43 | 0.82 |
| 3DRS | -1.02 | 0.82 |
| PPC | -1.21 | 0.85 |
| Proposed | 0.74 | 0.88 |
| Test 3 - NATIONAL TREASURE | Average Score | Standard Deviation |
| Bilinear | -0.45 | 1.02 |
| 3DRS | 0.87 | 0.88 |
| PPC | -1.58 | 0.96 |
| Proposed | 2.00 | 0.73 |
| Test 4 - FLIGHT | Average Score | Standard Deviation |
| Bilinear | 0.28 | 1.06 |
| 3DRS | -0.63 | 1.48 |
| PPC | -0.40 | 1.00 |
| Proposed | 1.52 | 0.79 |
| Test 5 - PRODUCERS | Average Score | Standard Deviation |
| Bilinear | -0.49 | 0.97 |
| 3DRS | -0.08 | 1.50 |
| PPC | -1.68 | 1.12 |
| Proposed | 2.13 | 0.63 |

Compensated Frame Interpolation in High-Definition Video Processing," Yen-Lin Lee and Truong Nguyen, revised to *IEEE Trans. on Circuits and Systems for Video Technology*, 2009. The dissertation author was the primary author of these publications, and the listed co-author directed and supervised the research that forms the basis for this chapter.

(a) (b)

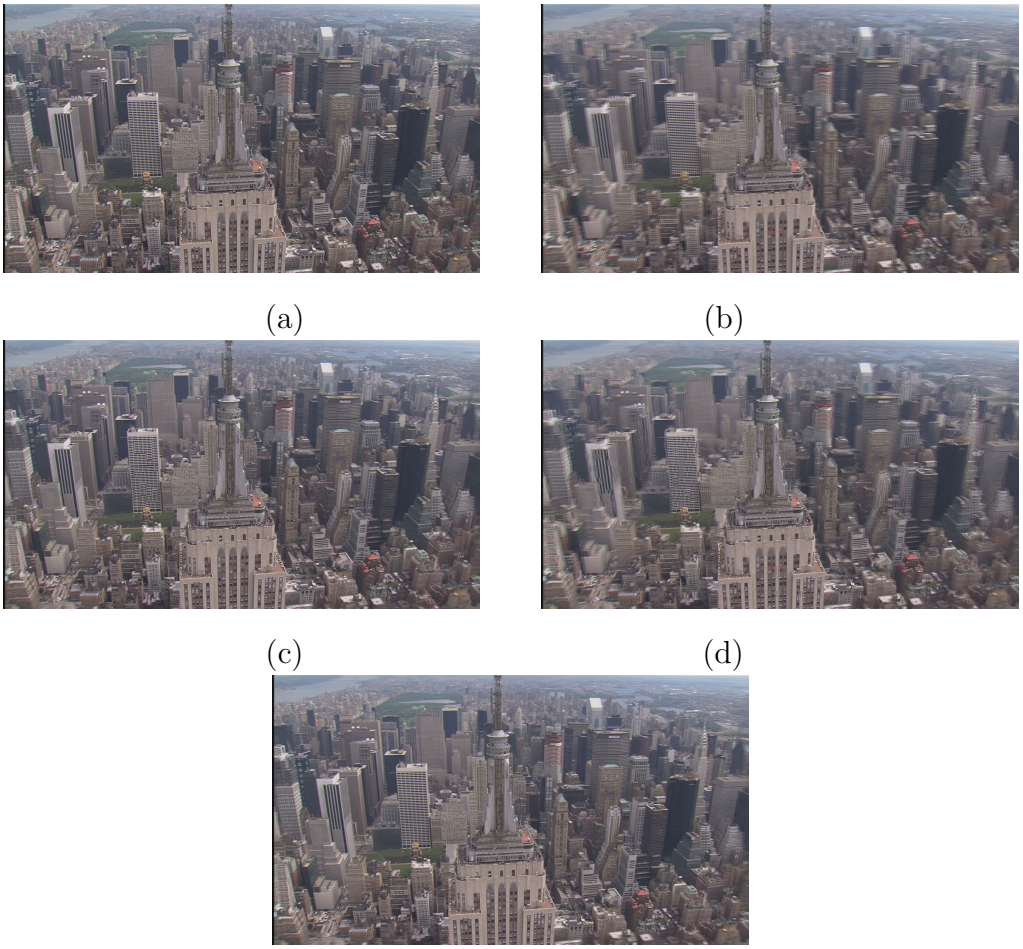
(c) (d)

(e) (f)

Figure 5.5: Interpolated frame 117 in CREW (a) original frame, (b) bilinear interpolation, (34.287 dB) (c) 3D Recursive Search (37.756 dB), (d) Phase Plane Correlation (33.418 dB), (e) proposed method (38.237 dB), (f) proposed method with sub_mv assignment (38.301 dB).

(a) (b) (c) (d) (e) (f)

Figure 5.6: Zoom in for interpolated frame 117 in CREW (a) original frame, (b) bilinear interpolation, (c) 3D Recursive Search, (d) Phase Plane Correlation, (e) proposed method, (f) proposed method with sub_mv assignment.



(a) (b)

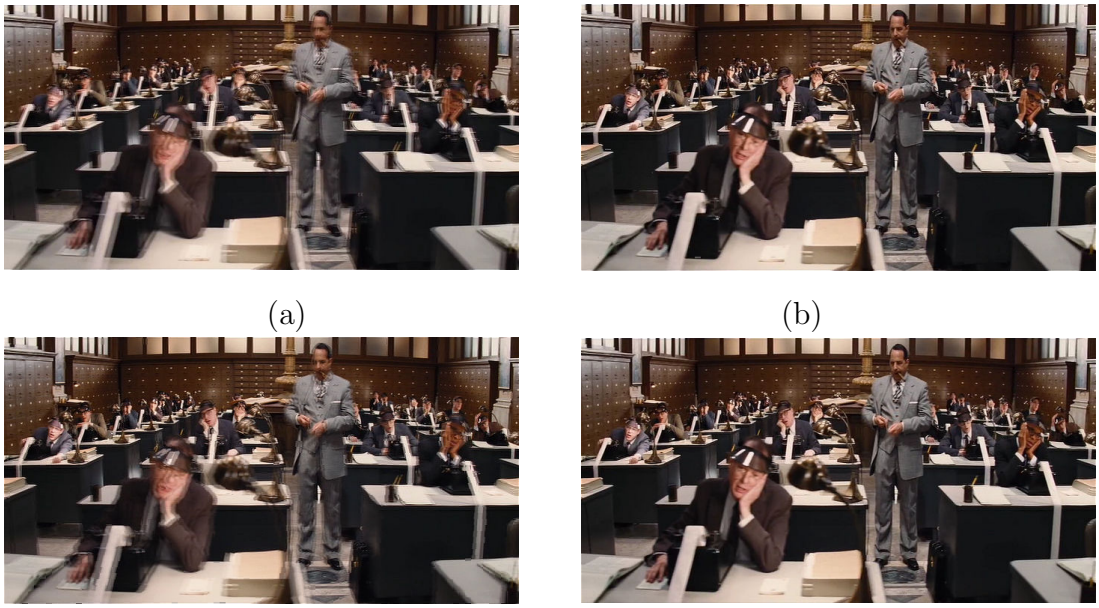
(c) (d)

(e)

Figure 5.7: Interpolated frame 2 in CITY (a) original frame, (b) bilinear interpolation (27.1726 dB), (c) 3D Recursive Search (29.0897 dB), (d) Phase Plane Correlation (27.2179 dB), (e) proposed method (35.9651 dB).

(a) (b) (c) (d) (e)

Figure 5.8: Zoom in for interpolated frame 2 in CITY (a) original frame, (b) bilinear interpolation, (c) 3D Recursive Search, (d) Phase Plane Correlation, (e) proposed method.



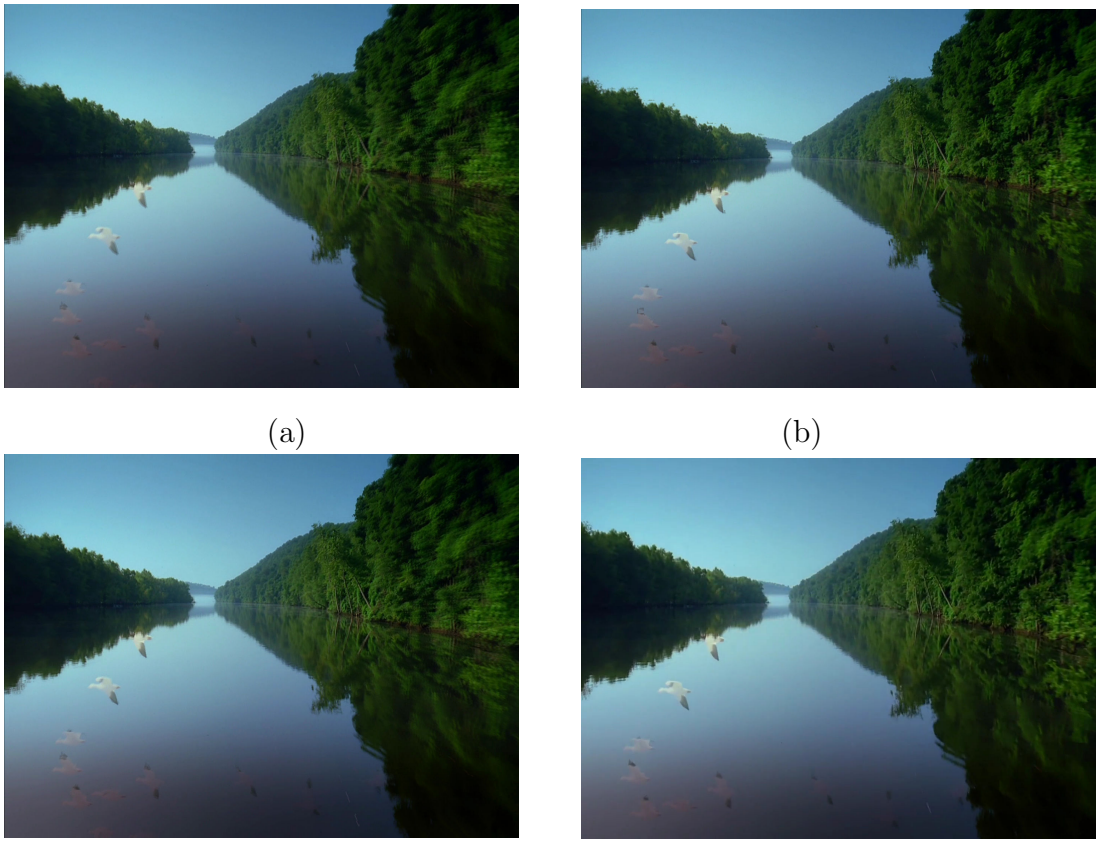
(a) (b)

(c) (d)

Figure 5.9: Interpolated frame 267 in PRODUCERS (a) bilinear interpolation, (b) 3D Recursive Search, (c) Phase Plane Correlation, (d) proposed method.

(a) (b) (c) (d)

Figure 5.10: Zoom in for interpolated frame 267 in PRODUCERS (a) bilinear interpolation, (b) 3D Recursive Search, (c) Phase Plane Correlation, (d) proposed method.



(a) (b)

(c) (d)

Figure 5.11: Interpolated frame 353 in FLIGHT (a) bilinear interpolation, (b) 3D Recursive Search, (c) Phase Plane Correlation, (d) proposed method.

(a) (b) (c) (d)

Figure 5.12: Zoom in for interpolated frame 353 in FLIGHT (a) bilinear interpolation, (b) 3D Recursive Search, (c) Phase Plane Correlation, (d) proposed method.

# 6 MCFI System Analysis

## 6.1 Analysis of the Proposed MCFI System

In this chapter, a detailed system loading analysis is discussed for each processing module. To profile the system loading time and analyze the architectural performance, the proposed method is implemented with a C implementation and profiled under Microsoft Visual Studio 2008. All analysis results are averaged over four test sequences, PRODUCERS (1440×960), NATIONAL_TREASURE (1440×960), LIVING_SEA (1440×1080), and FLIGHT (1440×1080). The proposed method operates on every other frame and compares the MCFI frames with the original frames to calculate the PSNR values.

Table 6.1: System Computation

| Module | Sub-Module | Sub-Loading | Loading |
|---|---|---|---|
| HD_Downsampling | - | - | 0.12% |
| MV_Search | MSEA_Search<br>PSAD_Search<br>MV_Selection | 43.01%<br>42.77%<br>0.11% | 85.89% |
| MV_Refining | - | - | 7.77% |
| Frame_Skipping | - | - | 0.01% |
| Interpolating | - | - | 3.98% |
| Deblocking | - | - | 0.02% |
| Temp_Generating | - | - | 1.79% |
| Others | - | - | 0.42% |
| Total | - | - | 100% |

TABLE 6.1 shows computational expense for each processing module under our software C implementation, and the percentage in this table represents what

percentage these individual modules take of the entire system loading time. There is little complexity for HD downsampling (0.12%) because we drop every other pixel without applying a real downsampling filter. To obtain more accurate true motion, the proposed method employs motion re-estimation without acquiring the motion vector field from the bitstream. Hence, true motion search (MV_Search) takes most of our processing time, with 85.89% of the overall processing time. In MV_Search, MSEA_Search takes 43.01% and PSAD_Search takes 42.77% of the overall processing time. Compared with the architecture design of the MSEA and the PSAD module in TABLE 5.1, we need more hardware resources than other modules to implement our search engine in real time. After MV_Search, MV_Refining is the second most expensive module in our system. It takes 7.77% of the overall processing time. The purpose of this module is to refine the true motion accuracy which become lower due to HD downsampling while also reducing ghost artifacts. Interpolation takes 3.98% of the overall processing time after we find true motions for each block. Deblocking helps us to eliminate the blocking artifacts because of dissimilarity of the motion field. In the proposed method, it is only applied to obvious blocking artifacts. Compared to the expensive deblocking method in the H.264/AVC codec (requiring up to 33% of processing power), the debocking process takes less than 1% of computing power. Finally, Temp_Generating produces temporal information for future processing, and it takes 1.79% of the overall processing time.

Table 6.2: Search Strategy Comparison

| Search Strategy | Loading | PSNR (dB) | PSNR (%) |
|---|---|---|---|
| Proposed Fast-Full Search | 100% | 33.74 | - |
| Conventional Full Search | 254% | 33.79 | +0.15% |
| Conventional Fast Search | 140% | 31.89 | -5.48% |

In the second part, we compare the proposed method with conventional full search and fast search methods. We determine performance by replacing the proposed search with other methods. TABLE 6.2 shows the comparison of the system loading and the PSNR error. The proposed method only takes 39% of the computing power of the conventional full search, but the average PSNR can ap-

proach 99.85% of the performance of the conventional full search. When compared with conventioanl fast search (three-step search), the proposed method spends less processing time and produces a better results.

Table 6.3: Module Performance Comparison

| Disable Options | PSNR (dB) | PSNR (%) |
|---|---|---|
| No MV_Selection | 33.59 | -0.44% |
| No MV_Refining | 33.68 | -0.18% |
| No Deblocking | 33.71 | -0.09% |

Finally, we disable specific options and observe the effect on PSNR. In TABLE 6.3, PSNR values change slightly. This is because these options primarily process the edge of moving objects and improve the visual quality of broken objects. However, when calculating the PSNR value on an entire frames, the PSNR values are improved only slightly.

## 6.2 Acknowledgement

Portions of Chapter 6 appear in "Novel Method and Architecture Design for Motion Compensated Frame Interpolation in High-Definition Video Processing," Yen-Lin Lee and Truong Nguyen, revised to *IEEE Trans. on Circuits and Systems for Video Technology*, 2009. The dissertation author was the primary author of these publications, and the listed co-author directed and supervised the research that forms the basis for this chapter.

# 7 High Frame Rate Up Conversion Processing

Frame Rate Up Conversion (FRUC) or Motion Compensated Frame Interpolation (MCFI) is an effective method to reduce judder for digital displays, especially for low response time devices, such as LCD HDTVs. New frames are interpolated and inserted between original or decoded frames to smooth motion blur and enhance the visual quality. Due to the widespread popularity of digital displays, more and more high quality LCD devices with high frame rate have emerged in the market. However, current video sources in the market are usually limited to 30- or 60-fps. In addition, the bandwidth of current broadcast channels or home theater media devices make it impossible to transmit a 120- or 240-fps high definition compressed video bitstream. Media storage and decoding power are also a problem for this system. Hence, high Frame Rate Up Conversion, such as 240-Hz or higher, has become an indispensable research topic stemming from current double frame rate technology. Fig. 7.1 shows an example of inserting three interpolated frames in order to generate a 4x frame rate video.

## 7.1 High Frame Rate Technology

To reduce blurring, most 120Hz LCD displays use a system called MEMC (Motion Estimation and Motion Compensation) or MCFI to insert in a new frame between each of the original frames and employ pull-down technology for display. The end result is one extra frame for every true frame. Although MEMC is the most

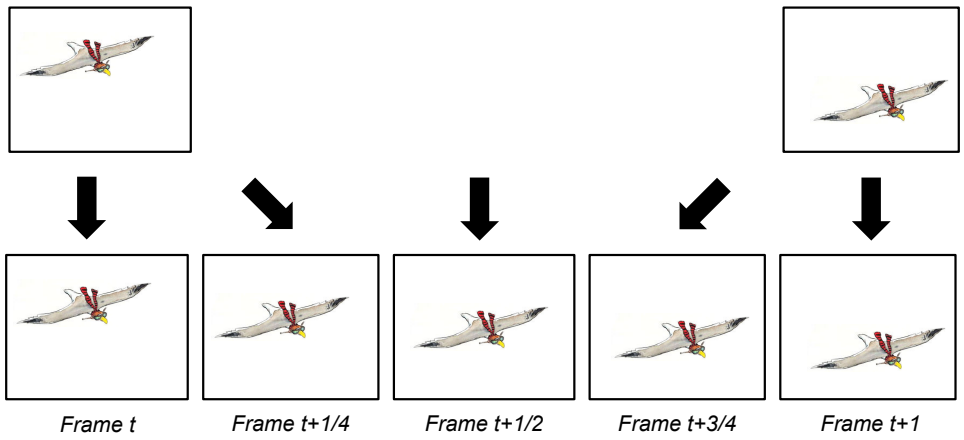


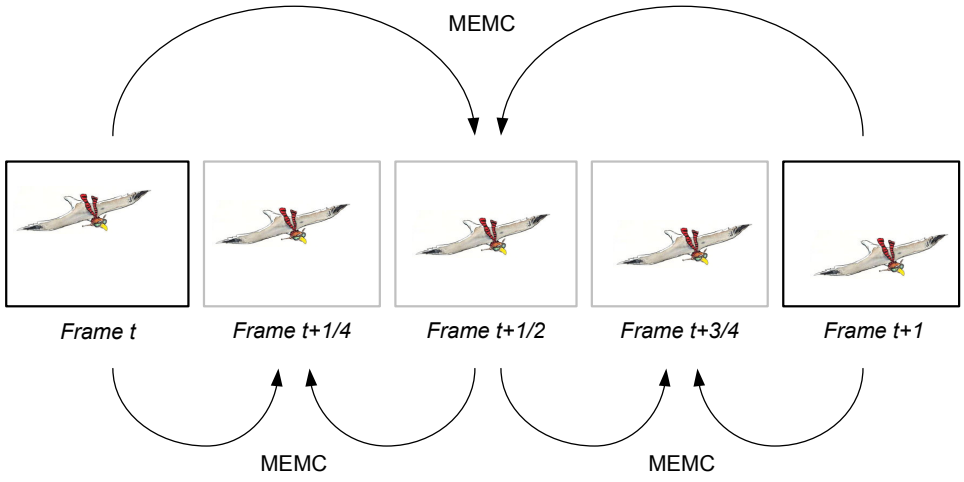

Figure 7.1: An example of 4x Frame Rate Up Conversion.

popular approach to increase the original frame rate, there are several different methods to achieve higher frame rate up conversion after performing MEMC.

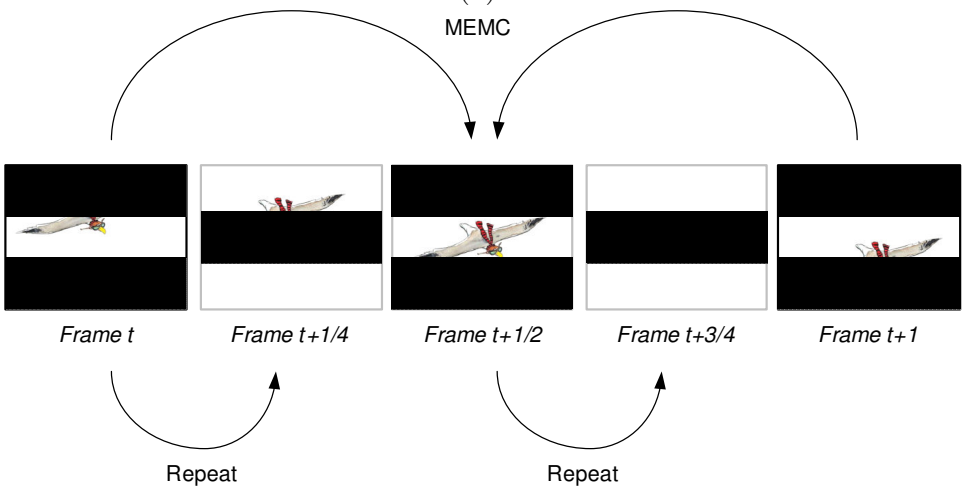
The first method is the pure MEMC method shown in Fig. 7.2(a). This is an easily modified method based on existing MEMC or MCFI modules. This process adopts the MEMC method to generate the 2x frame rate interpolated frame $t$+1/2 between any two original frames, frame $t$ and $t$+1, and then it reapplies the MEMC method to generate 4x frame rate interpolated frame $t$+1/4 and frame $t$+3/4. This method can easily apply to the existing 2x Frame Rate Up Conversion system, but the complexity dramatically increases. Not only will the computational complexity of one 4x pure MEMC method be three times higher than that of a 2x MEMC approach, but memory requirement increases three times. This method is used for small resolution video or lower frame rate video for practical reasons.

The second method is MEMC with backlight scanning as shown in Fig. 7.2(b). This process adopts MEMC to generate the 2x frame rate interpolated frame $t$+1/2 between any two original frames, frame $t$ and $t$+1, and then it repeats every 2x frames with backlight scanning. This technique that synchronizes the display's pixel updates to a cycling pattern of illumination generated by fluorescent tube or LED array backlight modules [37]. Developing an effective method to insert black data between image frames is one of the most viable ways to shorten the spatio-temporal integration time. In the case in Fig. 7.2(b), 50% of the data

frame is blanked by black data, and this 50% luminance loss will dim the visual display. In order to reduce luminance loss, a sharpened or alternate gamma frames should be driven in order to maintain the luminance. Although this method can provide much lower complexity than pure MEMC, visual quality is lower than pure MEMC, especially on lower frame rate videos.




(a)




(b)

Figure 7.2: Two methods to achieve 4x Frame Rate Up Conversion. (a) Pure MEMC method. (b) MEMC method with backlight scanning.

The third method is MEMC with motion trajectory, shown in Fig. 7.3, to achieve 4x frame rate up conversion. Motion trajectory method is a common method for MCFI in a low complexity system, especially for a MCFI system which

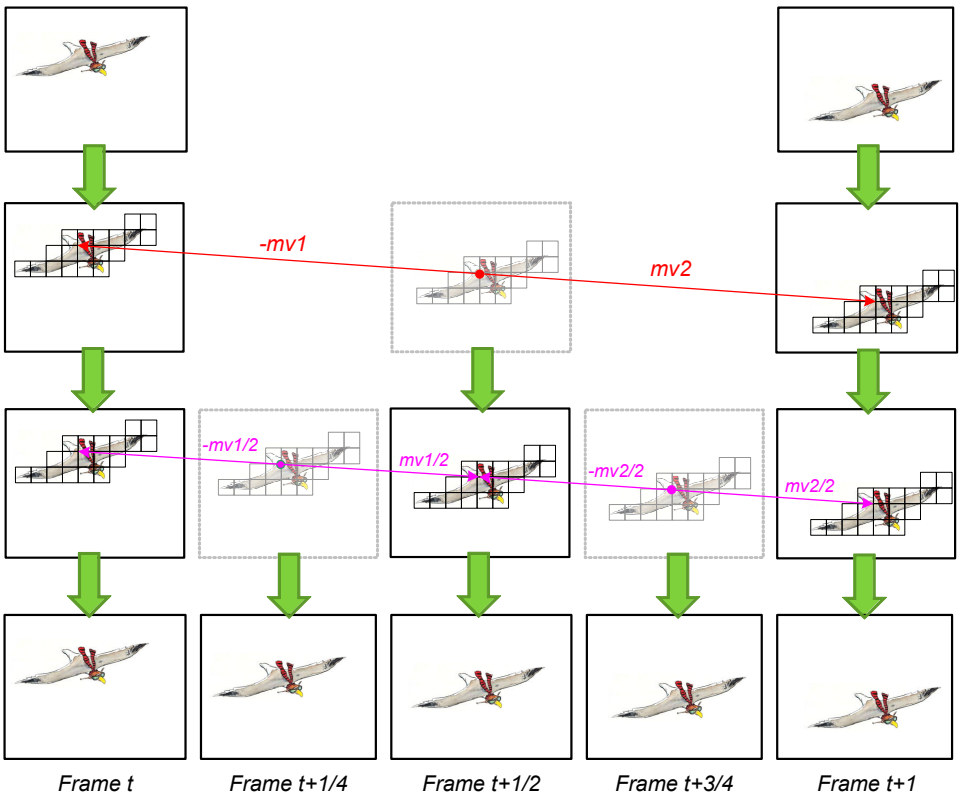



Figure 7.3: MEMC method with motion trajectory to achieve 4x Frame Rate Up Conversion.

extracts the motion vectors from the decoder and post-processes the Motion Vector Field (MVF). This process adopts MEMC to generate the 2x frame rate interpolated frame $t$+1/2 between any two original frames, frame $t$ and $t$+1, and then it takes one half of the motion vector field from 2x video processing and projects to the same location of 4x images, such as frame $t$+1/4 and frame $t$+3/4. The benefit of this method is that there is no need to repeatedly perform motion estimation while still utilizing the advantage of the motion compensated interpolation. However, the quality suffers from faster moving objects because motion trajectory results in less precision. Fig. 7.4 demonstrates an example of the trajectory method damaging the shape of the moving object if 2x motion vector cannot perfectly map to 4x motion motion vectors.

In Fig. 7.4, the first step is to search true motions (1) for 2x frame rate images via MEMC. Based on the Motion Vector Field (MVF) generated by the

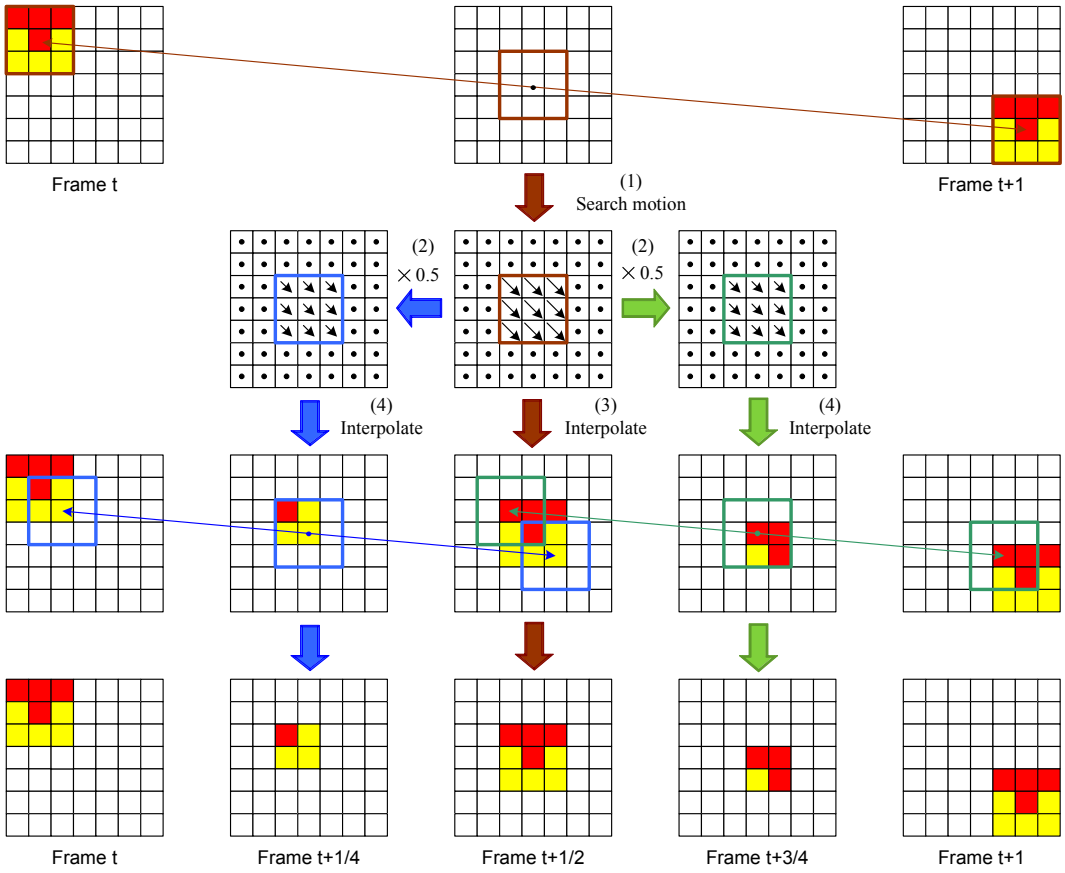



Figure 7.4: MEMC method with motion trajectory to achieve 4x Frame Rate Up Conversion.

first step, these motion vectors move along the trajectory and assign new motion vectors for 4x frame rate images (2) without motion re-estimation. In the second step, MVF for 4x frame rate image will be half of motion vectors from 2x frame rate images. The third step is to interpolate 2x frame rate images with 2x true MVF (3) based on two original images before generating 4x frame rate images. The final step is to interpolate 4x frame rate images with 4x projected true motion field (4) based on one original image and one 2x frame rate interpolated frame. Fig. 7.4 shows that the video will be flickering and broken because the imprecise 4x MVF result in obvious artifacts and produces incomplete objects on frame $t$+1/4 and frame $t$+3/4.

The following sections will discuss the proposed high Frame Rate Up Conversion. One low complexity MEMC method and its architecture will be proposed.

This proposed method is derived from the third approach shown in Fig. 7.3, but it can provide highly precise true MVF as the pure MEMC method does. The proposed method generates a better visual quality video than pure MEMC at much lower complexity.

## 7.2 Proposed High Frame Rate Processing

There are several different approaches to achieve high Frame Rate Up Conversion, as the previous section mentioned. Here, we will modify our previously proposed MCFI scheme, slightly increase complexity, and achieve 4x Frame Rate Up Conversion. The proposed scheme in Chapter 3, shown in Fig. 3.1, adopts a motion-compensated approach to double the frame rate by inserting one interpolated frames between any two contiguous original frames. A new processing scheme is proposed and shown in Fig. 7.5, and the blocks enclosed by the red box are additional functional blocks to process 4x Frame Rate Up Conversion. Motion Vector Refinement for 4x in Fig. 7.5 is the major processing procedure to assign proper motion vectors for two additional interpolated frames. This module also belongs to the proposed true motion engine. Frame Skipping and Block-based Interpolating for 4x MEMC processing are similar to those in the previous proposed 2x MCFI scheme. Based on this proposed scheme, one interpolated frame is generated by 2x MCFI processing flow and two interpolated frames generated by 4x MCFI or MEMC processing flow.

The module of Motion Vector Refinement for 4x employs a similar method to sub-block motion assignment in Chapter 3. Because we can obtain the true MVF after motion vector refinement or sub-block motion assignment from 2x MCFI processing flow, the assigned motion vectors at the same position of current block and eight neighboring blocks from the 2x interpolated frame will be considered as motion candidates for 4x MCFI processing flow. By succeeding the MVF from the prior 2x process, it can reduce the computational complexity in true motion searching. Based on the experiments, if 2x MCFI processing cannot find true motion vectors for 2x interpolated frames, it is impossible to search true motions

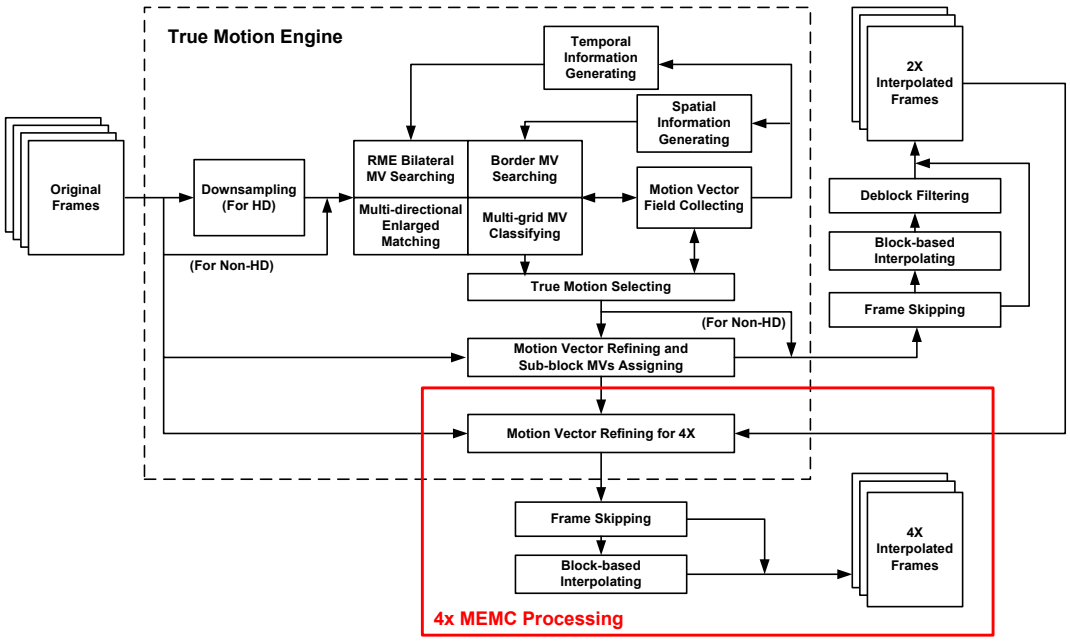



Figure 7.5: Processing flow of the proposed method on 4x Frame Rate Up Conversion.

for 4x interpolated frames from the incorrect 2x interpolated frames. In this case, the system needs to skip MCFI so as not to generate incorrect interpolated frames because of the inaccurate MVF. Hence, making use of the 2x true MVF as motion candidates is an effective method to improve the accuracy of 4x true motion vectors and also take the computational complexity into consideration.

In the proposed 4x MCFI scheme, true motion vector assignment from 2x motion candidates will be used for an N×N block (by default N=8 in the proposed architecture). Fig. 7.6 shows that the proposed block assignment method examines motions from the surrounding motions of the current 2x interpolated frame. True motions, $\overrightarrow{MV_i}$ when $i$ equals 0 to 8, are checked with the SAD function from one original frame (frame $t$ or frame $t$+1) and one 2x interpolated frame (frame $t$+1/2). This 4x motion assignment is formulated by

$$\begin{aligned}\overrightarrow{CMV\_4X_0} &= \arg\min_{\vec{v}\in S'} \sum_{x\in B} \left| f(x-\tfrac{1}{2}\vec{v},t) - f(x+\tfrac{1}{2}\vec{v},t+\tfrac{1}{2}) \right| \\ \overrightarrow{CMV\_4X_1} &= \arg\min_{\vec{v}\in S'} \sum_{x\in B} \left| f(x-\tfrac{1}{2}\vec{v},t+\tfrac{1}{2}) - f(x+\tfrac{1}{2}\vec{v},t+1) \right|\end{aligned} \tag{7.1}$$

where $B$ denotes a matching N×N block of the current interpolated position; $S'$ is a set of motion candidates, including $\overrightarrow{MV_i}$ when $i$ equals 0 to 8; $\overrightarrow{CMV\_4X_i}$ is the assigned motion vector for 4x images examined for the best matching when $i$ = 0 for the first 4x interpolated frame and $i$ = 1 for the second 4x interpolated frame. Two adjacent frames are denoted by $f(x,t)$ and $f(x,t+1)$, where $x$ and $t$ are spatial and time domain indices. The current 2x interpolated frame is denoted by $f(x,t+\frac{1}{2})$.

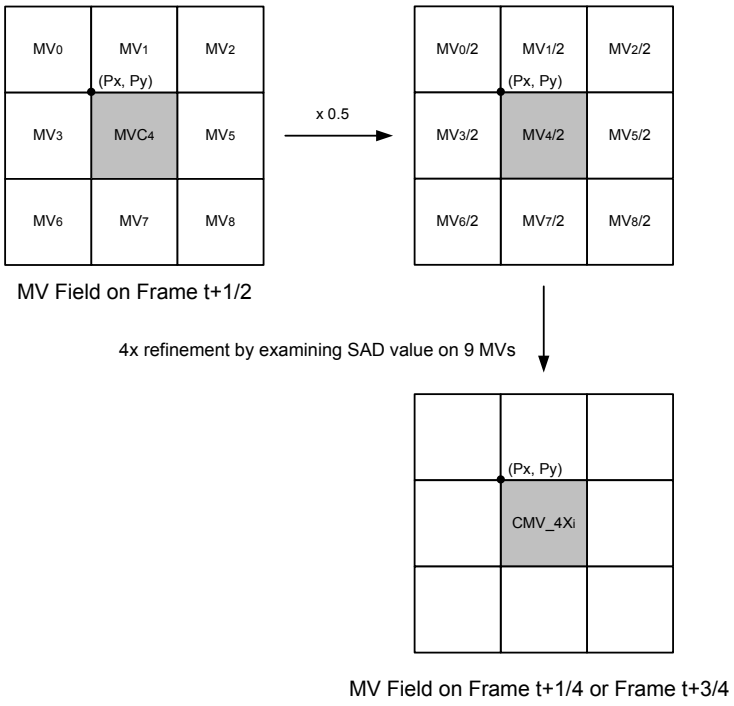



Figure 7.6: Proposed MEMC method with motion refinement to achieve 4x Frame Rate Up Conversion.

Fig. 7.7 demonstrates an example of how 4x true motion refinement can reduce artifacts from the broken object in Fig. 7.4. The first step is to search true motions (1) for 2x frame rate images by MEMC method. Secondly, the MVF for 4x frame rate images (frame t+1/4 and frame t+3/4) take half of motion vectors from 2x frame rate images (2). The third step is to interpolate the 2x frame rate images with 2x true MVF (3) based on the original images before generating 4x frame rate images. The fourth step is to refine 4x true MVF with Eqn. (7.1) and assign a motion vector for each block (4). The final step is to interpolate 4x frame

rate images with 4x assigned true motion field (5) based on one original image and one 2x frame rate interpolated image. Comparing Fig. 7.7 with 7.4, MEMC with the proposed 4x refinement can approach a more accurate 4x MVF and generate more precise 4x interpolated images because the proposed refinement method takes more neighboring motion vector candidates into account and thereby improving the visual quality when compared with the motion trajectory method.

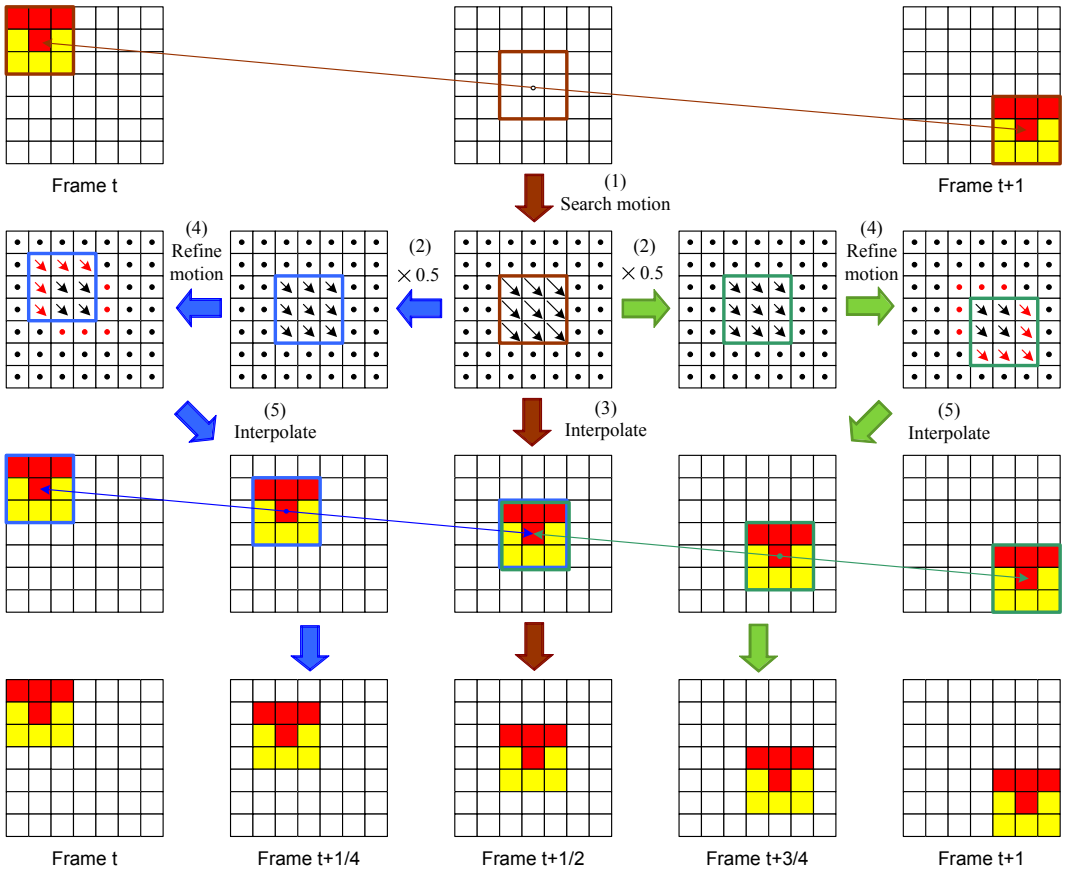



Figure 7.7: Proposed MEMC method with motion refinement to achieve 4x frame rate up conversion.

There is another technique, Relative Motion Estimation (RME), included in the 4x proposed method. The problem is shown in Fig. 7.8(a) when the true motion vector is out of the searching window $w$(t). There are several methods that can solve this problem. The first one is to increase the search range, but the computational complexity also become dramatically higher. Additionally, a larger searching window reduces the accuracy of true motion search. The second method

is to repeat or average the block when the true motion search engine cannot find a reliable motion vector. It avoids broken images or objects when the selected motion vector is incorrect, but the visual quality will still be blurred. The third method or our proposed searching method, Relative Motion Estimation (RME), is shown in Fig. 7.8(b). We assume that any object should have a smaller acceleration than its velocity, and RME is a good method to track the motion if the acceleration is smaller than the RME search range, especially for a rapidly panning scene. Fig. 7.8(b) shows how a larger motion vector can be found by using RME. RME will check the PSAD value when searching, and the motion vector will revert to zero motion (average the block) if the PSAD value is larger than the error threshold value, T_RME_Error. RME algorithm for updating motion vector can be stated by

$$
\begin{aligned}
&do \\
&\{ \\
&\quad t = 0; \\
&\quad if \quad PSAD(t) < T\_RME\_Error \\
&\qquad \overrightarrow{mv}(t) = \overrightarrow{mv'}(t) + \overrightarrow{mv}(t-1); \\
&\quad else \\
&\qquad \overrightarrow{mv}(t) = (0,0); \\
&\quad t++ \\
&\} \\
&while(t! = last)
\end{aligned}
$$

The proposed RME method searches true motion vectors based on previous motion at the same block position from the previous 2x interpolated frame processing. Hence, if the new motion vector of the current interpolated block is the same as that of the previous block, the current motion vector, $\overrightarrow{mv'}$(t), will be set as (0,0), and $\overrightarrow{mv}$(t) will be $\overrightarrow{mv'}$(t)+$\overrightarrow{mv}$(t-1)=$\overrightarrow{mv}$(t-1). If the RME engine finds a reliable $\overrightarrow{mv'}$(t) within the searching window, the current $\overrightarrow{mv}$(t) will be $\overrightarrow{mv'}$(t)+$\overrightarrow{mv}$(t-1). If the RME engine cannot find any reliable motion vector within the searching range,

the current $\overrightarrow{mv}$(t) will be set (0,0).

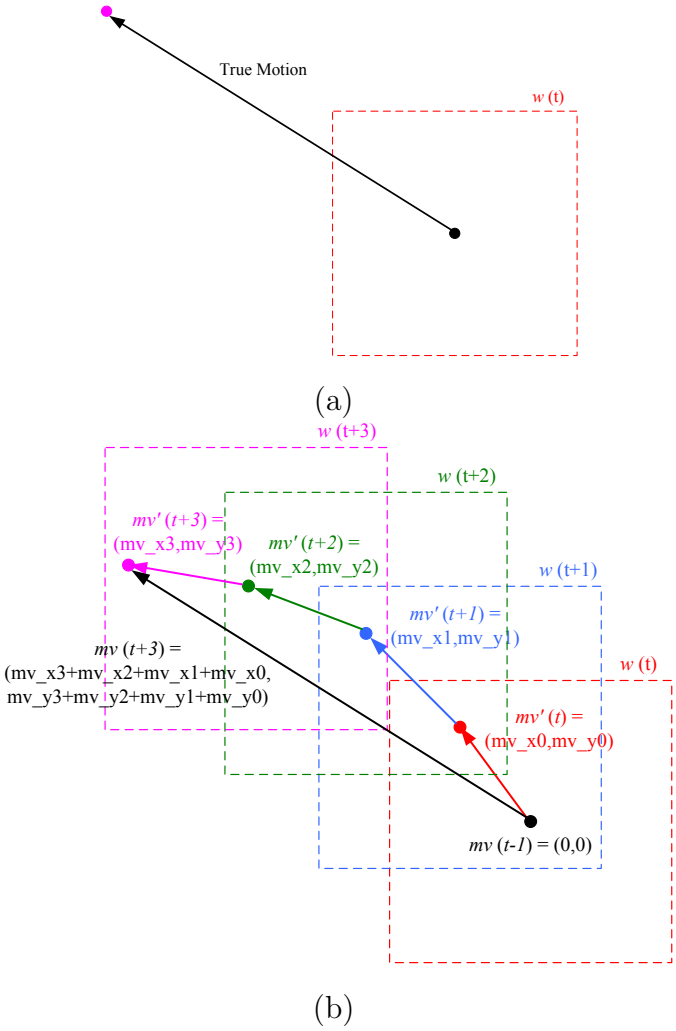



Figure 7.8: (a) Normal true motion vector search. (b) Relative true motion vector search.

## 7.3 Proposed Architecture Design for High Frame Rate Processing

Several modules are included in the proposed 4x system architecture which is shown in Fig. 7.9. This architecture implements and simplifies the proposed processing flow shown in Fig. 7.5. The additional module in the system archi-

tecture is 4X Motion Compensator, which generate each interpolated block for 4x inserted frames and write back to the external memory through the system memory controller. Due to 4x MCFI processing, the size of some internal buffers, including Reference Buffer, Temporal Parameter Buffer, and Motion Vector Field Buffer, should be enlarged for appropriate operations. The Filtering Buffer has no need to be enlarged because it does not change in the proposed 4x MCFI process.

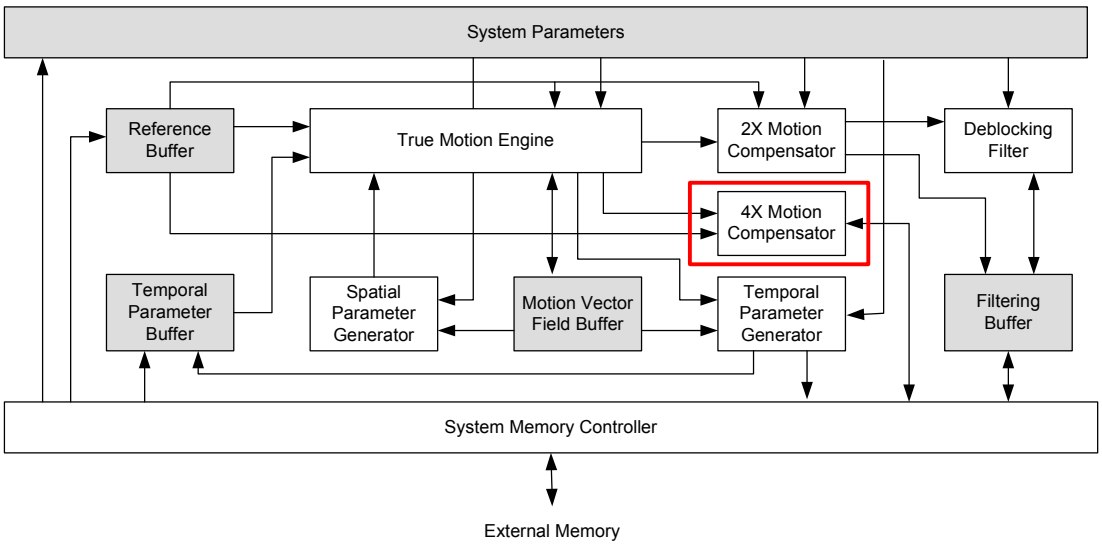



Figure 7.9: System block diagram of the proposed 4x MCFI architecture.

The true motion engine, shown in Fig. 7.10, is the core of the proposed MCFI algorithm. It searches for true motions for motion compensation based on two original reference frames and motion information from the temporal and spatial domains. After determining true motion for current 2x interpolated frames, the 4X Refine Engine takes these selected motion vectors and refines them as motion vectors for 4x interpolated frames. Because there are two 4x interpolated frames required for processing, two pairs of 4XR Cal (4x Refining Calculator) and 4XR Cpr (4x Refining Comparer) are designed in the proposed architecture in Fig. 7.10. 4XR FSM (4x Refining Finite State Machine) controls the pipeline flow and fetches the data for processing. The blocks enclosed by the red border are additional functional blocks to process 4x Frame Rate Up Conversion.

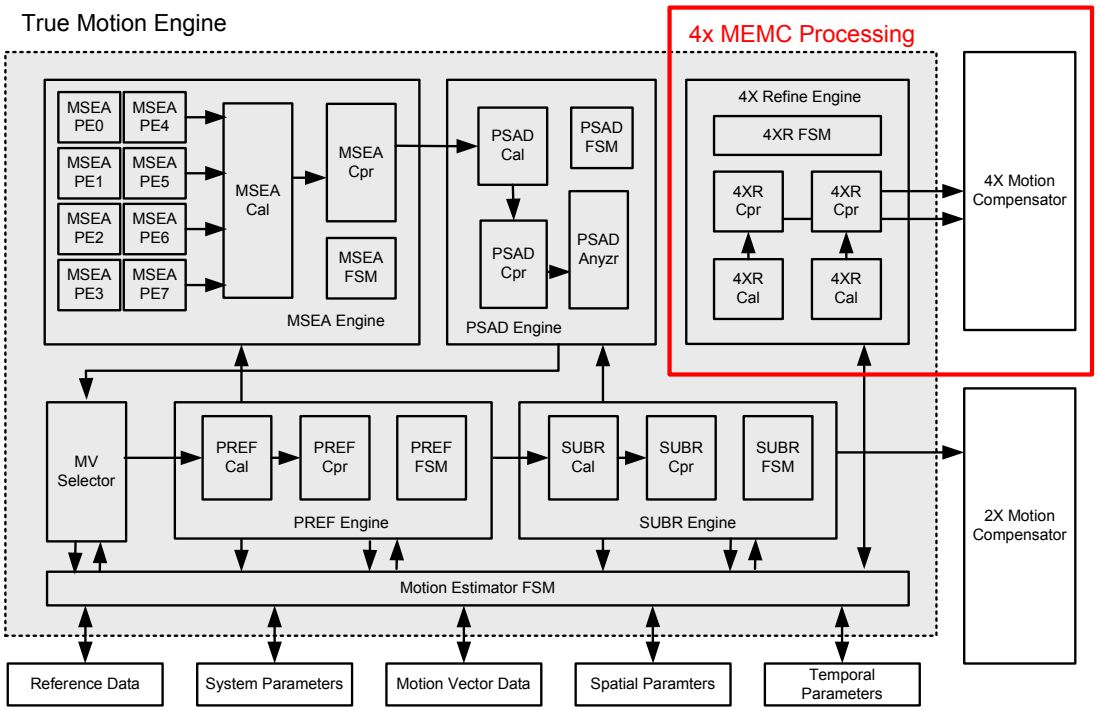



Figure 7.10: Block diagram of the proposed 4x true motion engine.

## 7.4 Experimental Results

In this section, several test sequences, FLIGHT (1080p), PRODUCERS (1440×960), CAMCUT (VGA), and STATE (1080p), are processed with the proposed 4x MCFI method. Visual quality of these test sequences will be shown with two successive original frames, one 2x interpolated frame, and two 4x interpolated frames. We will zoom in on a specific area of each visual quality result so as to clearly observe the change over time. We also compare the proposed RME 4x methods with the original processing method when the motion vectors are larger than the search range from (0,0). By comparing with the results of normal searching strategy without skipping or skipping with averaging the images, the proposed method using RME can provide better visual quality when RME method can track the acceleration change within the RME search window.

Fig. 7.11 shows an example of the proposed 4x MCFI processing on FLIGHT (1080p). Fig. 7.11(a) and (e) are two original and successive frames from the test

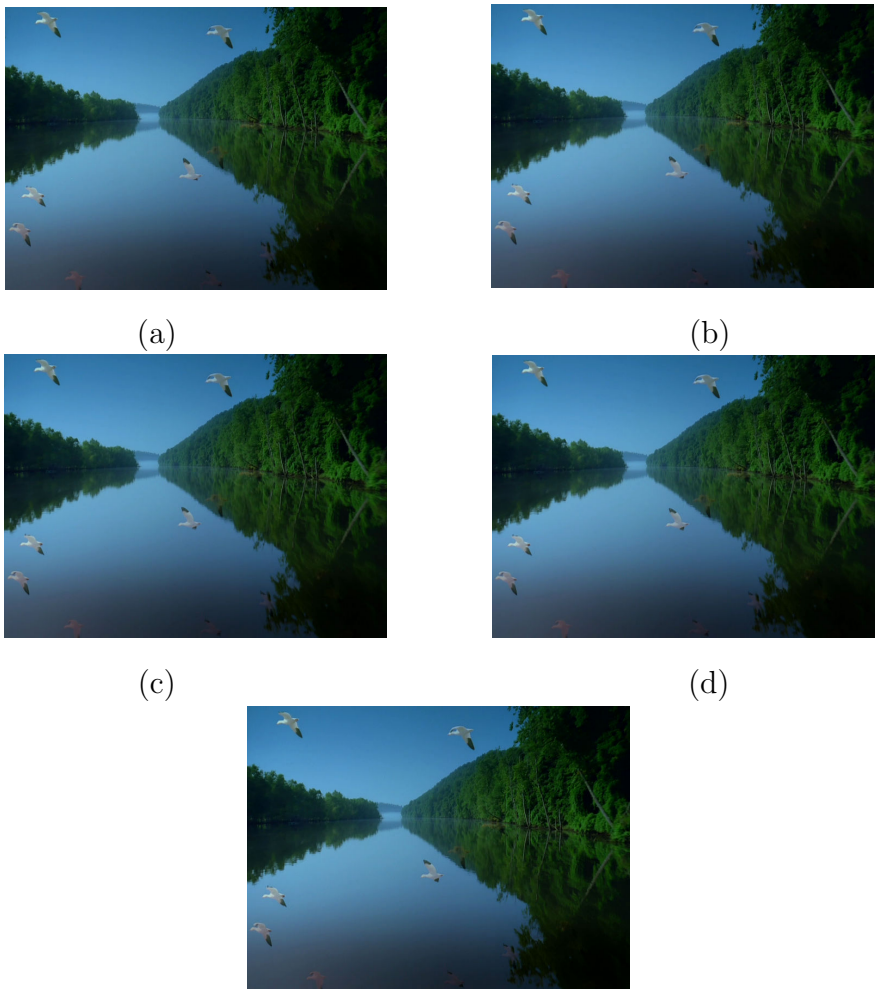
sequence, and Fig. 7.12(a) and (e) zoom in on the specific area of a seabird in order to observe the shape change. From these images, both wings spread and move from 7.12(a) to (e). Fig. 7.11(c) and 7.12(c) show the result of the 2x interpolated frame, and Fig. 7.11(b)(d) and 7.12(b)(d) show the results of the 4x interpolated frames. These demonstrate that the proposed 2x and 4x MCFI methods can successfully track the motion and generate the intermediate stages from Fig. 7.11(a) to (e). Fig. 7.13-7.18 show the processed results of the proposed 4x MCFI processing on PRODUCERS (1440×960),CAMCUT (VGA), and STATE (1080p).

Fig. 7.19 and 7.20 show the artifacts when processing on the same original frames in Fig. 7.17 if the proposed method does not perform Relative Motion Estimation (RME). Fig. 7.19(b) and 7.20(b) present the results of the normal motion search window without RME on the 2x interpolated frame. Because the motions from Fig. 7.17(a) to (e) are larger than our predefined searching window [-30,+30] based on zero motion [0,0], it is impossible to find true motions in this case. Hence, Fig. 7.19(b) shows a lot of broken blocks and artifacts with incorrect motions, and the same situation occurs on Fig. 7.19(a) and (c). When the proposed method fails to find reliable true motion vectors, it may either repeat the previous images or average from two original successive frames for 2x interpolated frames and then average from one original frame and one averaged 2x interpolated frame to generate 4x interpolated frames. The latter results are demonstrated in Fig. 7.19(d)(e)(f). No matter which interpolation method the proposed method adopts for these incorrect motions, it cannot recover a clear interpolated frame. Therefore, this example demonstrates that RME can find the true motion when its acceleration is smaller than the RME searching window, which allows it to generate unblurred images.

## 7.5 Acknowledgement

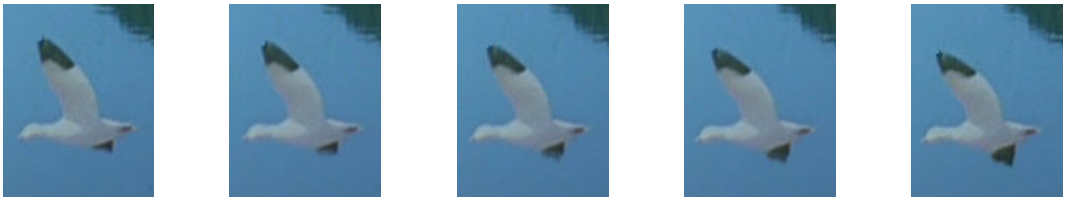
Portions of Chapter 7 appear in “High Frame Rate Motion Compensated Frame Interpolation in High-Definition Video Processing,” Yen-Lin Lee and Truong

Nguyen, submitted to *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2010. The dissertation author was the primary author of these publications, and the listed co-author directed and supervised the research that forms the basis for this chapter.



(a) (b) (c) (d) (e)

Figure 7.11: The proposed 4x MCFI Processing in FLIGHT (a) original frame 924, (b) 4x interpolated frame 925, (c) 2x interpolated frame 926, (d) 4x interpolated frame 927, (e) original frame 928.



(a) (b) (c) (d) (e)

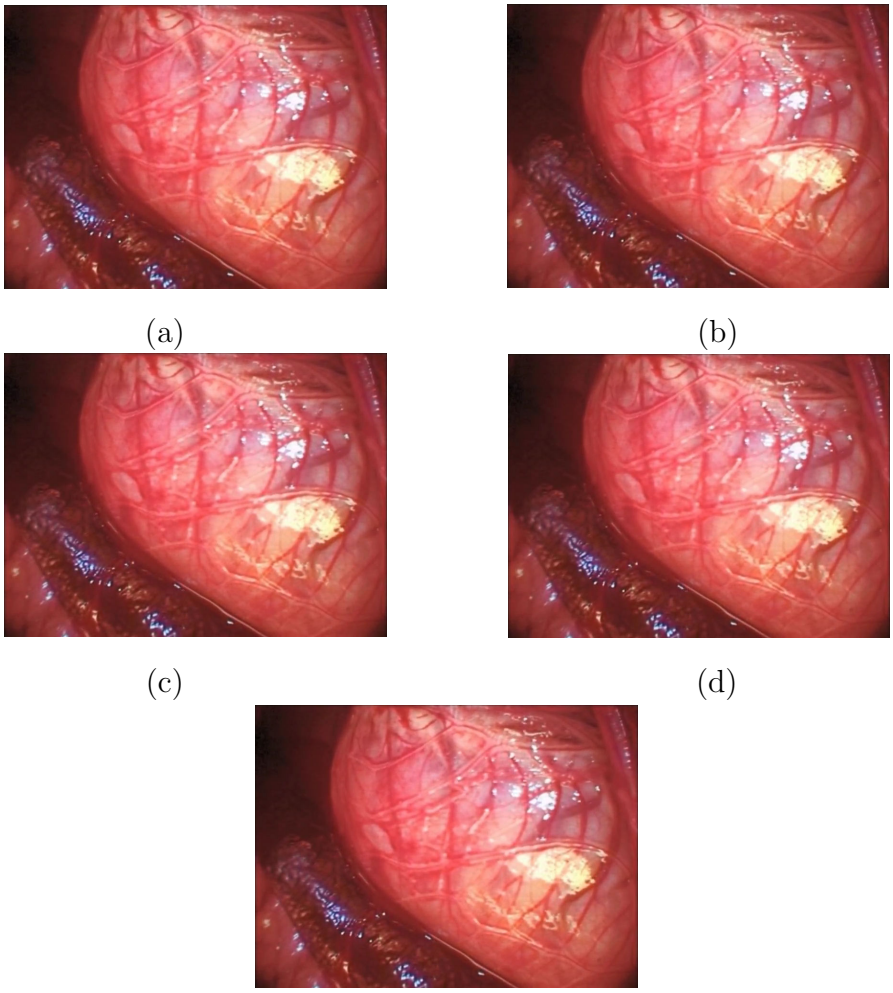
Figure 7.12: Zoom in on the proposed 4x MCFI Processing in FLIGHT (a) original frame 924, (b) 4x interpolated frame 925, (c) 2x interpolated frame 926, (d) 4x interpolated frame 927, (e) original frame 928.

(a) (b) (c) (d) (e)

Figure 7.13: The proposed 4x MCFI Processing in PRODUCERS (a) original frame 400, (b) 4x interpolated frame 401, (c) 2x interpolated frame 402, (d) 4x interpolated frame 403, (e) original frame 404.

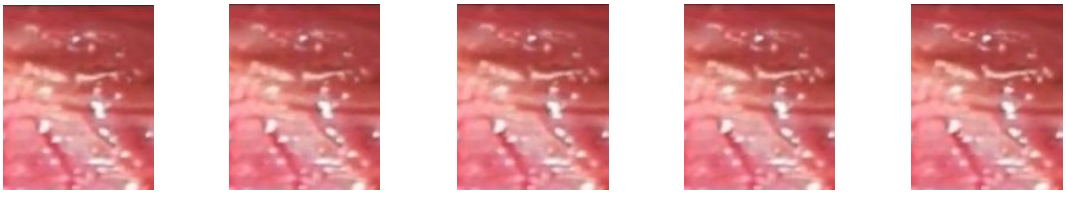
(a) (b) (c) (d) (e)

Figure 7.14: Zoom in on the proposed 4x MCFI Processing in PRODUCERS (a) original frame 400, (b) 4x interpolated frame 401, (c) 2x interpolated frame 402, (d) 4x interpolated frame 403, (e) original frame 404.



(a) (b) (c) (d) (e)

Figure 7.15: The proposed 4x MCFI Processing in CAMCUT (a) original frame 132, (b) 4x interpolated frame 133, (c) 2x interpolated frame 134, (d) 4x interpolated frame 135, (e) original frame 136.



(a) (b) (c) (d) (e)

Figure 7.16: Zoom in on the proposed 4x MCFI Processing in CAMCUT (a) original frame 132, (b) 4x interpolated frame 133, (c) 2x interpolated frame 134, (d) 4x interpolated frame 135, (e) original frame 136.

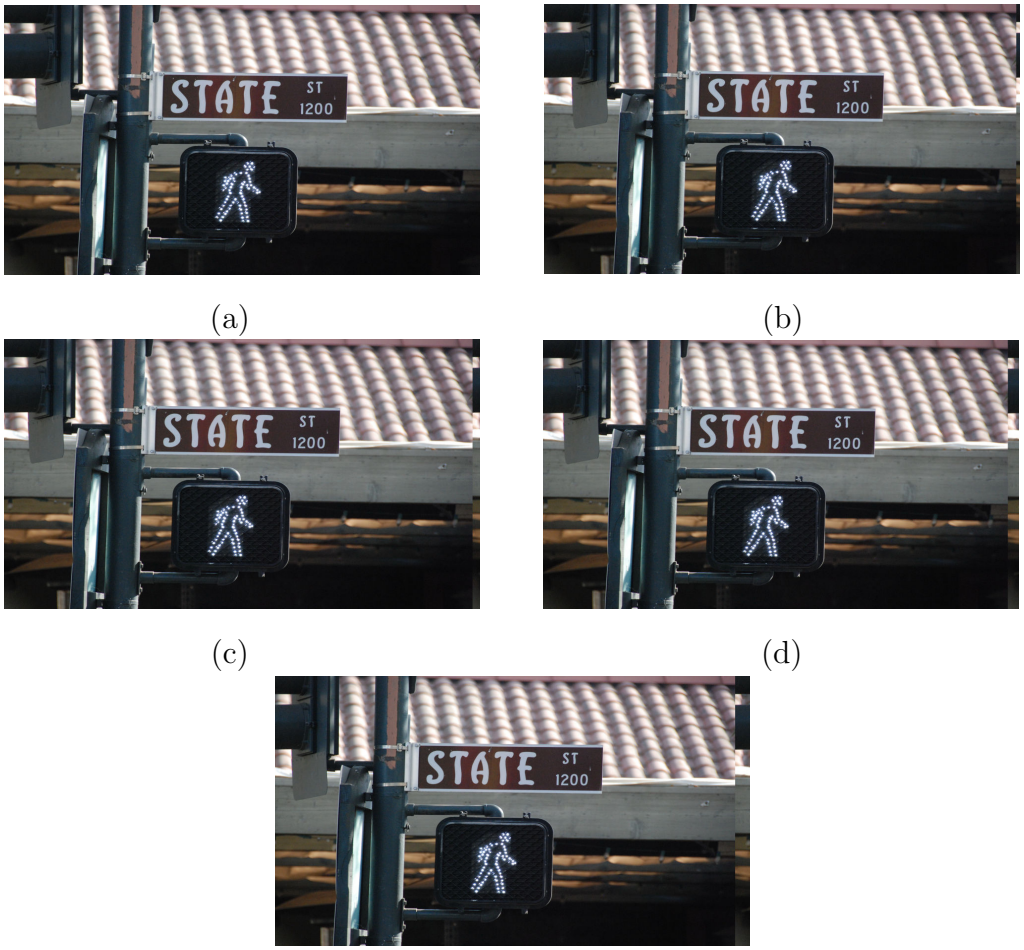
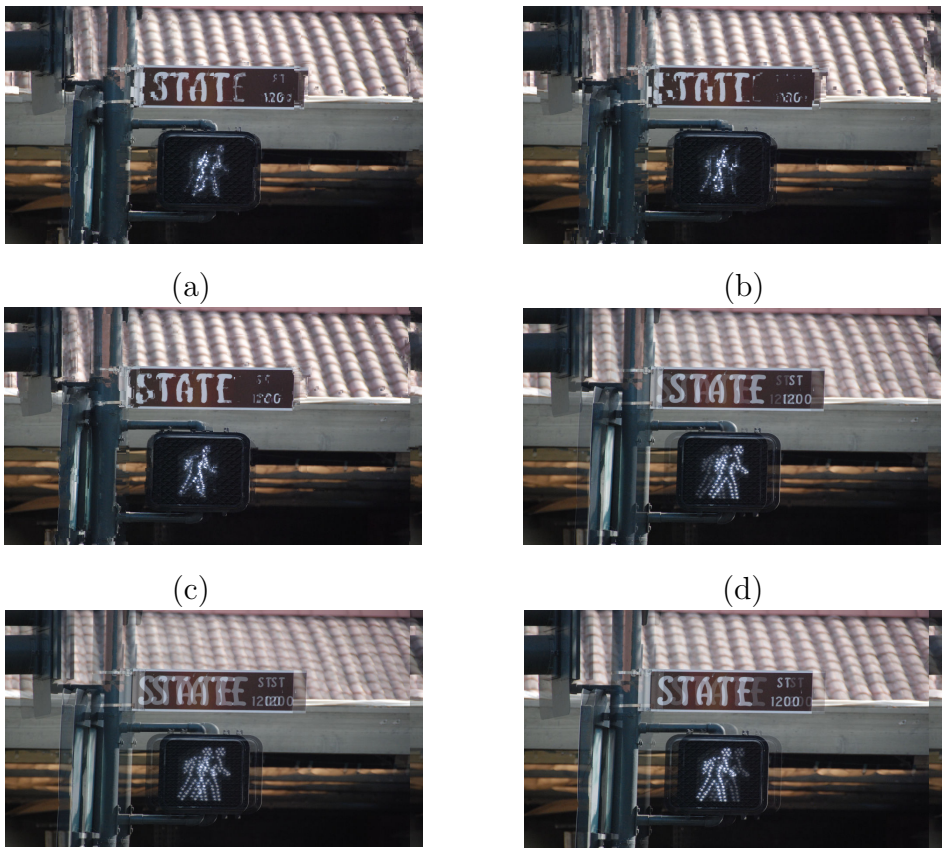


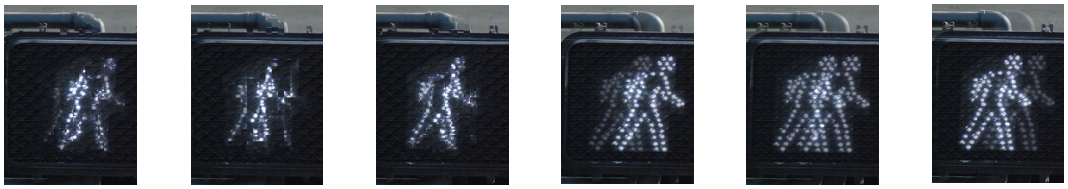

Figure 7.17: The proposed 4x MCFI Processing in STATE (a) original frame 1580, (b) 4x interpolated frame 1581, (c) 2x interpolated frame 1582, (d) 4x interpolated frame 1583, (e) original frame 1584.


Figure 7.18: Zoom in the proposed 4x MCFI Processing in STATE (a) original frame 1580, (b) 4x interpolated frame 1581, (c) 2x interpolated frame 1582, (d) 4x interpolated frame 1583, (e) original frame 1584.




(a) (b) (c) (d) (e) (f)

Figure 7.19: 4x MCFI comparisons with incorrect motions and image averaging in STATE (a) 4x interpolated frame 1581 with incorrect motions, (b) 2x interpolated frame 1582 with incorrect motions, (c) 4x interpolated frame 1583 with incorrect motions, (d) 4x interpolated frame 1581 with averaging, (e) 2x interpolated frame 1582 with averaging, (f) 4x interpolated frame 1583 with averaging.



(a) (b) (c) (d) (e) (f)

Figure 7.20: Zoom in on the 4x MCFI comparisons in STATE (a) 4x interpolated frame 1581 with incorrect motions, (b) 2x interpolated frame 1582 with incorrect motions, (c) 4x interpolated frame 1583 with incorrect motions, (d) 4x interpolated frame 1581 with averaging, (e) 2x interpolated frame 1582 with averaging, (f) 4x interpolated frame 1583 with averaging.

# 8 Conclusions

In this dissertation, a novel, fast, and efficient method with a well-designed architecture is proposed for Motion Compensated Frame Interpolation (MCFI). Our method employs a unique true motion engine with an adaptive Overlapped Block Matching Algorithm (OBMA), Multi-Directional Enlarged Matching Algorithm (MDEMA), and one-pass processing. The proposed architecture employs a modified Multi-level Successive Eliminate Algorithm (MSEA), which greatly reduces computational expense.

Each technique of the proposed methods is separately discussed via system analysis and profiling. We examine each component's contribution and process loading in the proposed MCFI design. In addition to normal double Frame Rate Up Conversion, one high Frame Rate up Conversion method is proposed. Experimental results show that the proposed algorithm provides better video quality than conventional methods and shows satisfying performance to cope with 30fps HD1080p video at 180 MHz or 30fps 720p video at 83MHz.

# A Analysis and Efficient Architecture Design for VC-1 Overlap Smoothing and In-loop Deblocking Filter

## A.1 Introduction

VC-1 [33] is a video codec specification that has been standardized by the Society of Motion Picture and Television Engineers (SMPTE) and adopted in next-generation optical media formats, such as HD-DVD and Blu-ray. Although the basic functionality of VC-1 adopts a block-based motion compensation and spatial transform method similar to that used in other video compression techniques, VC-1 introduces several techniques and optimization that make it distinct in coding efficiency, frame quality, and computational complexity from the basic compression scheme. Comparing to H.264/AVC [32], the design approach of VC-1 lowers computational complexity without significant performance loss since H.264/AVC uses many complex techniques to improve visual quality. Lower complexity leads to practical architecture, lower cost, and less power consumption and heat dissipation. Although VC-1 operates at lower complexity, it still approaches a compression ratio similar to H.264/AVC [38]. Fig. A.1 shows the encoding loop of a VC-1 codec.

An in-loop deblocking filter is frequently used to increase coding efficiency

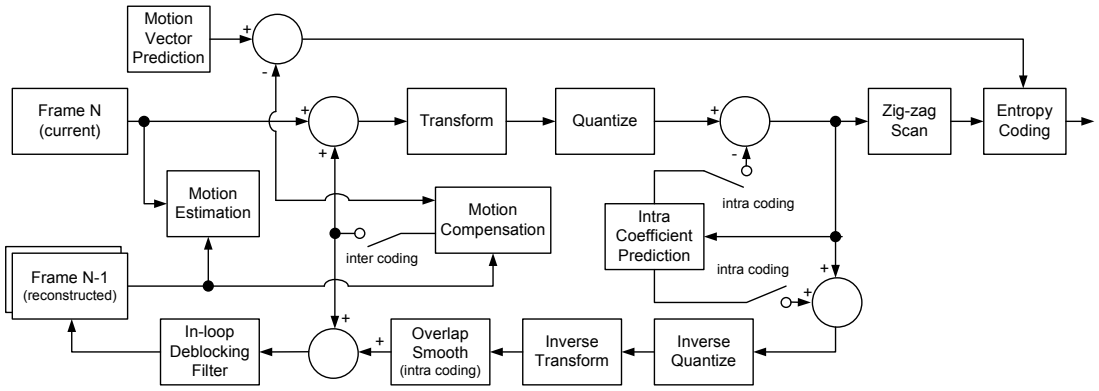



Figure A.1: Encoding loop of VC-1 codec.

and remove blocking artifacts [39]. Besides an in-loop filter, VC-1 also includes an overlap smoothing filter for smoothing real blocking artifacts [40]. However, the procedures of these two filters in VC-1 are both arranged in frame-based orders, which means that all horizontal edges (for in-loop filtering) or vertical edges (for overlap smoothing) should be performed first and followed by the vertical edges (for in-loop filtering) or horizontal edges (for overlap smoothing). Although there are many well-designed methods and architecture designs that specifically deal with loop filtering in a macroblock-based order [41][42][43][44][45], none of them is appropriate for VC-1 frame-based filtering. The reason is that these methods fail to filter all edges within the current macroblock due to the problem of data dependency in VC-1. According to VC-1 filtering procedure, in-loop filtering, for example, should be performed on horizontal edges prior to vertical edges. That is, it is difficult to filter all internal edges while relying on previous methods. In addition, the procedure of overlap smoothing, filtering vertical edges prior to horizontal edges, is also different from in-loop deblock filtering, in which the order is reversed. If these previous methods are applied directly to a VC-1 deblocking filter, they could cause many unnecessary processing cycles and inefficient memory access.

According to our analysis of VC-1 filters, including overlap smoothing and loop filtering, we propose a method and process them in a proposed block-based order different than the original frame-based order in the VC-1 standard for bet-

ter data access and processing efficiency. To efficiently perform VC-1 filtering, we define a 12×12 overlapped block and process both overlap smoothing and loop filtering within it. This method works with the proposed architecture to perform VC-1 deblocking filtering in a block-based order but not in a frame-based order so as to pipeline with the block reconstructing step. It also combines two VC-1 filters in order to improve the performance and reduce the cost by sharing circuits and resources. However, this technique requires additional memory access and increases on-chip memory size as compared to normal macroblock-based in-loop filters, such as H.264/AVC deblocking filter. For this reason, we propose two other processing methods, multiple macroblock processing order and modified chrominance processing order, to reduce the overhead of these system resources. These two proposed methods lower the requirement of system resources by changing the entire processing order for the block reconstruction and filtering steps.

## A.2 Deblocking Filters in VC-1

### A.2.1 Overlap Smoothing

Overlap smoothing is a technique used to reduce blocking artifacts in intra data by means of a lapped transform [40]. This operation should conditionally be performed across the edge of two neighboring intra blocks, for both the luma and chroma data. Unlike normal in-loop filters that may blur real edges or lose some details because these filters receive the data after inverse quantization, inverse transform, and reconstructing blocks; overlap transform performs a pre-processing step in the spatial domain during the encoding procedure and a post-processing step following inverse transform during the decoding procedure.

Overlap smoothing is performed on vertical edges first and then horizontal edges in a specific order within a frame or slice. This post-processing overlapped filter acts on four pixels straddling either the vertical or horizontal edge of a block. This operation will be carried out on the unclamped 10-bit reconstructed pixels. In other words, the input to the overlap smoothing process is the inverse transformed

spatial block of pixels whose dynamic range is 10 bits. This is necessary because the forward process associated with overlap smoothing can result in range expansion beyond the permissible 8 bit range for pixel values. For pixels performed in both vertical and horizontal filtering procedures, the intermediate result after vertical edge filtering is retained at the full precision of 11 bits. This data width is for 10 bits input data plus one bit worst case range expansion. Subsequent to filtering, the constant value of 128 will be added to each pixels of the block, which will then be clamped to the 8-bit range to produce the reconstructed output.

### A.2.2 In-loop Deblocking Filter

An in-loop deblocking filter is also used for removing block-boundary discontinuities introduced by quantization errors in interpolated frames. This filtering operation should conditionally be performed on each reconstructed frame or slice. It is performed prior to using the reconstructed frame as a reference for motion predictive coding. Because this filtering acts to smooth out the discontinuities at block boundaries, the process operates on the pixels that border two neighboring blocks. For I or B picture, filtering boundaries occur at every $8^{th}$, $16^{th}$, $24^{th}$, etc. pixel row and column. For P picture, the filtering boundaries may occur at every $4^{th}$, $8^{th}$, $12^{th}$, etc. pixel row or column depending on whether an 8×8, 8×4, 4×8, or 4×4 inverse DCT transform is used. Moreover, all blocks or subblocks that have a boundary along the $8^{th}$, $16^{th}$, $24^{th}$, etc. horizontal lines should be filtered first. All subblocks that have a horizontal boundary along the $4^{th}$, $12^{th}$, $20^{th}$, etc horizontal lines should be filtered later. I, B, and P interlace frame deblocking orders are different from the progressive deblocking orders and are defined in the VC-1 standard [33].

Fig. A.2 shows an example for all filtered edges relative to the luma data of a macroblock. For an I or B progressive frame, the loop filtering procedure would be $H0$, $H1$, $H2$, $V0$, $V1$, and $V2$. For a P progressive frame, the loop filtering procedure would be $H0$, $H1$, $H2$, $h0$, $h1$, $V0$, $V1$, $V2$, $v0$, and $v1$. Comparatively, the overlap smoothing procedure would be $V0$, $V1$, $V2$, $H0$, $H1$, and $H2$. These

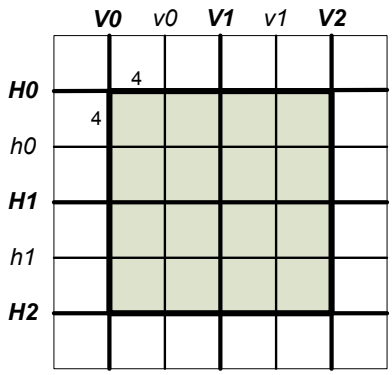



Figure A.2: All filtered edges relative to the luma data of a macroblock.

two filters have different processing order.

## A.3 Proposed Deblocking Processing Method in VC-1

The VC-1 filtering procedure is very time-consuming and requires significant memory access loading for the entire system. It is also difficult to pipeline filters with block reconstruction because the specified order of VC-1 cannot apply the filtering procedure one block/macroblock at a time. In this section, we propose several efficient methods to improve the filtering procedure, reduce the processing time, pipeline filtering operations, and greatly reduce memory access. In our methods, overlap smoothing operations are closely followed by loop filtering operations within an overlapped block for sharing common circuitry and reducing unnecessary memory access.

### A.3.1 Integrated Modified Processing Order

Our block-based procedure adopts a 12×12 overlapped block as our basic filtered block size. Fig. A.3 shows a 12×12 overlapped block, including a filtering area and an overlapped area. The filtering area is the region within the bold square, which is an 8×8 block within a macroblock, and the rest is the overlapped

area, which may be partially filtered for the current overlapped block. Other overlapped blocks will complete all filtering operations for this area afterward. This proposed overlapped block is employed as a common processing field for both overlap smoothing and loop filtering.

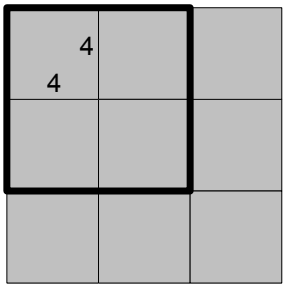



Figure A.3: A 12×12 overlapped block. The bold square defines an 8×8 block (luma or color-difference block) within a macroblock.

Two modified processing orders for overlap smoothing and loop filtering are proposed in Fig. A.4. Fig. A.4(a) shows an overlap smooth processing order applied to an overlapped block. Any edge with a predefined index in the figure could be a filtered edge during processing. A filtered edge with a smaller index should be performed prior to that with a larger index. That is, the filtered edge with index 1 should be filtered first, and the edge with index 2 will be filtered successively, and so on. However, not every predefined edge should be filtered. Edge 1 of an overlapped block could be edge 3 of the top neighboring block, and edge 4 also could be edge 6 of the left neighboring block. If these edges are filtered by a previous overlapped block, they cannot be filtered again. When pixels of the block are completely filtered by overlap smoothing operations, the constant value of 128 should be added to each pixel and then be clamped to the 8-bit range for producing the reconstructed output before loop filtering. VC-1 only supports 4:2:0 chromatic format, so each 16×16 macroblock is composed of six 8×8 blocks, four luma (Y) blocks and two chroma (Cb and Cr) blocks.

Fig. A.4(b) shows a loop filtering processing order applied to an overlapped block. The edge of loop filtering could be any boundary of the 4×4 block because of different transform types (8×8, 8×4, 4×8, or 4×4) within a macroblock. Similarly,

a filtered edge with a smaller index should be performed earlier than that with a larger index. Edge 1 and edge 4 are edge 3 and edge 6 of the left neighboring block, and the current loop filtering operation should not be processed redundantly.

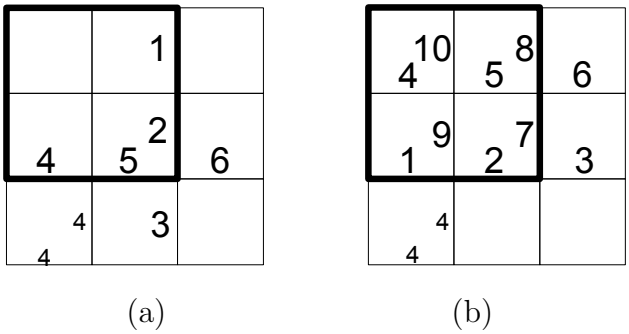



Figure A.4: Fundamental processing order. (a) Overlap smooth processing order applied to an overlapped block. (b) Deblocking filter processing order applied to an overlapped block.

Based on these two basic modified processing orders for an overlapped block, the proposed method employs the common pixels that technically accomplish the integration of two different filtering operations of VC-1. Every pixel after performing the first modified processing order should already be overlap-smoothed and clamped to an 8-bit value. The other processing order of loop filtering could be followed without any additional memory access. Fig. A.5 shows a processing flow of an overlapped block. First, unfiltered pixels are read from external memory or on-chip memory buffer, and all information and parameters are prepared. Second, a modified processing order for overlap smoothing is performed. Third, pixels of intra blocks are added with the constant value of 128 and clamped to 8 bits. Fourth, the other modified processing order for loop filtering is executed. Lastly, filtered pixels are output to external memory or the on-chip memory buffer.

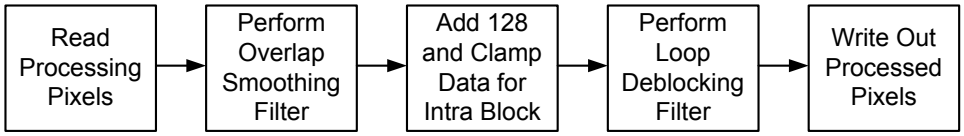



Figure A.5: Simple processing flow of an overlapped block.

### A.3.2 Pipeline Processing by Moving Macroblock Position

One important reason that the proposed method attempts to modify the original processing order to a block-based procedure is to pipeline the filtering step with the reconstruction step. Nevertheless, it fails to filter all edges within the current macroblock because some filtering edges should not be performed prior to particular edges on the boundary between the current macroblock and neighboring unreconstructed macroblock. Fig. A.6 demonstrates an example to show the data dependency problem within a reconstructed macroblock. Edges a, b, c, and d should be processed prior to edge f when performing overlap smoothing. However, edges c and d cannot be filtered due to the unavailable pixels right of the current macroblock while the current macroblock is being reconstructed. Similarly, edges e, f, g, and h should be processed prior to edge b when performing loop filtering. Edges g and h cannot be filtered right away due to the lack of pixel information below the current macroblock.

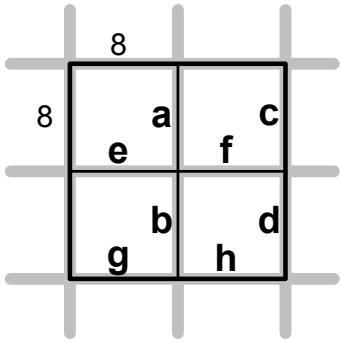



Figure A.6: 8×8 edges within a reconstructed macroblock. For overlap smoothing, edges a, b, c, and d should be filtered prior to edge f. For in-loop filtering, edges e, f, g, and h should be filtered prior to edge b.

In our method, the position of a filtered macroblock is shifted to a new position by 8 pixels in the x-axis (left) and 8 pixels in the y-axis (up) on a frame or slice from the position of a current macroblock for both luma and chroma data. Fig. A.7(a) demonstrates a current reconstructed macroblock within luma data, and the position of the first pixel in this macroblock is located at (m , n). Hence, the position of the first pixel in the filtered macroblock would be located at (m-8 ,

n-8) shown in Fig. A.7(b) when both the top macroblock and the left macroblock exist. When the left neighboring macroblock does not exist, the first and third blocks, i.e., two left blocks marked with 0 and 2, of a filtered macroblock do not exist either. When the top neighboring macroblock does not exist, the second and fourth blocks, i.e., two top blocks marked with 0 and 1, of a filtered macroblock do not exist either. If the current macroblock is the first in a frame or slice, only the fourth block, the bottom-right block marked with 3, of a filtered macroblock exists. Fig. A.7(c) shows all 4×4 input blocks when filtering a filtered macroblock. The proposed method separates a filtered macroblock into four basic overlapped blocks shown in Fig. A.7(d), which can be performed by our proposed modified processing orders for both overlap smoothing and loop filtering.

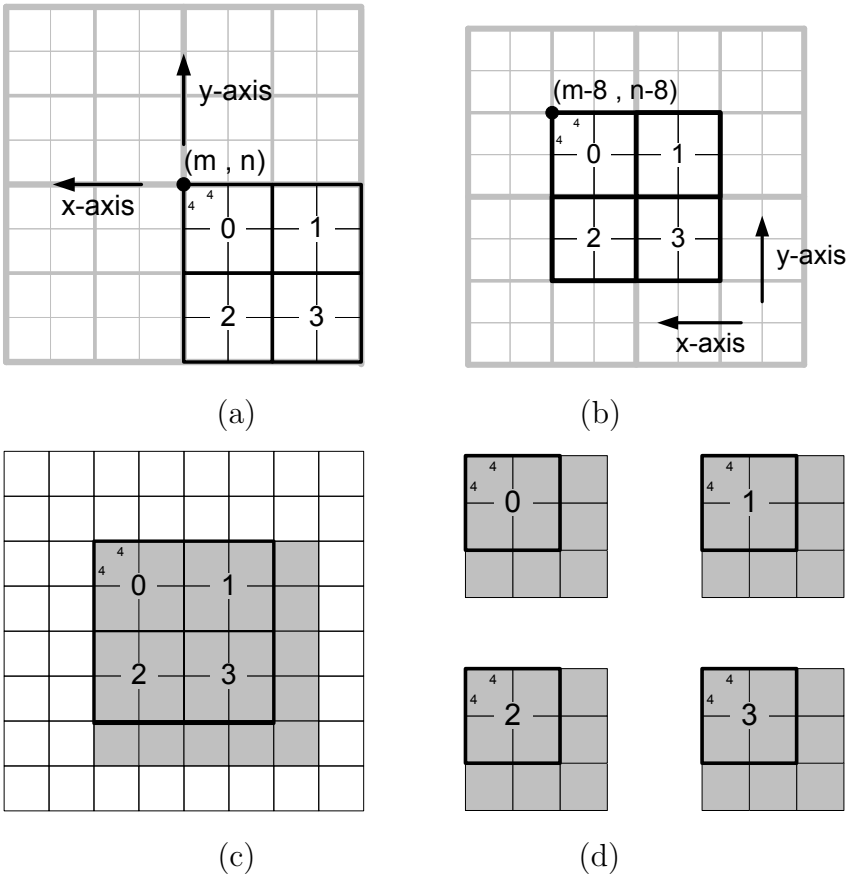
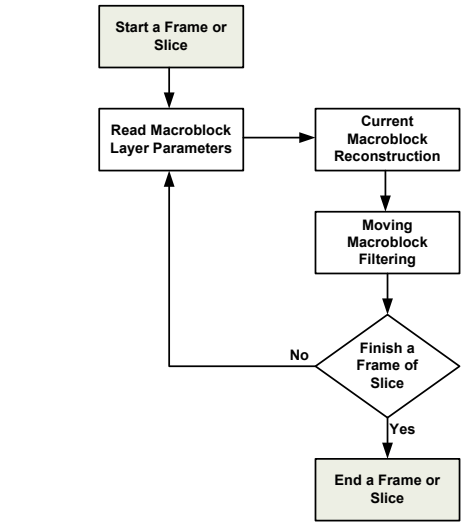



Figure A.7: (a) A current reconstructed macroblock. (b) A moving filtered macroblock with the current macroblock. (c) All 4×4 input blocks for a moving filtered macroblock. (d) Four separated overlapped blocks from the filtered macroblock.

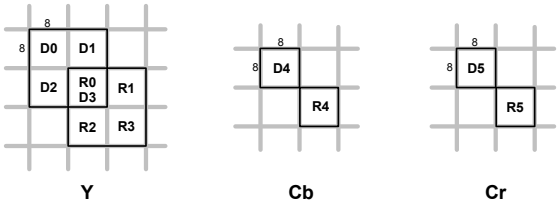
Fig. A.8(a) shows a processing flow for reconstruction and filtering within a frame or slice. First, macroblock-layer parameters are previously provided by software from the external memory buffer, which is usually implemented on DDR-SDRAM or DDR2-SDRAM. Next, a macroblock is reconstructed, and then the related moving macroblock is filtered. The precedure will repeat these steps until a frame or slice is finished. Fig. A.8(b) shows the pipeline schedule during the period of performing a reconstructed macroblock and a filtered macroblock. The first filtered 8×8 overlapped block has to wait until the first 8×8 reconstructed block completes. The basic schedule of a macroblock is combined with other macroblocks to accomplish the entire schedule of a frame or slice.

### A.3.3 Single and Multiple Macroblock Processing Order

After defining the processing order within a macroblock, we analyze and modify the processing order within a frame or slice in this section. Fig. A.9(a) shows a single processing order processing a macroblock following the most recent macroblock in a normal raster order. Two chroma 8×8 blocks are followed with four luma blocks when processing a 4:2:0 macroblock. In the proposed method, we need the temporal data buffer to store partially filtered or unfiltered pixels. There are two types of buffer: the temporal data buffer implemented by on-chip SRAM that stores short-term pixels for the current macroblock and the next macroblock; and the temporal data buffer allocated in external memory that stores long-term pixels for the next macroblock row. Data processing operations for processing a reconstructed macroblock and a filtering macroblock for luma data are shown in Fig. A.10. The data of the top three 8×8 blocks in Fig. A.10 are temporal neighboring read data, which are partially filtered by previously filtered macroblocks. The data of the bottom three 8×8 blocks in Fig. A.10 are temporal neighboring write data, which will partially be filtered by current macroblock and written back into the temporal data buffers. If there is no more filtering required in this area, the system will write this data into the decoded picture buffer of the external memory as the future reference data or for displaying. The data of the top-left




(a)




| Time | T0 | T1 | T2 | T3 | T4 | T5 | T6 |
|---|---|---|---|---|---|---|---|
| Reconstruction | R0 | R1 | R2 | R3 | R4 | R5 | - |
| Filtering | - | D0 | D1 | D2 | D3 | D4 | D5 |

(b)

Figure A.8: (a) Data processing flow of a frame or slice. (b) Pipeline time schedule of a reconstructed macroblock and a filtering macroblock.

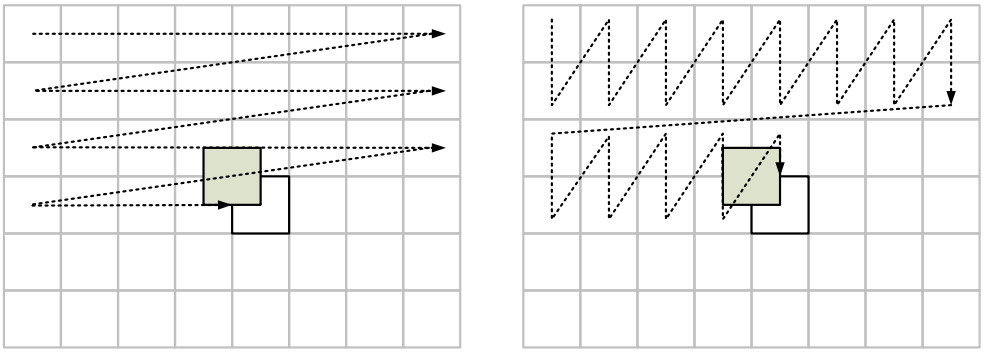
four 8×8 blocks in Fig. A.10 will be filtered pixels and will be written into the decoded picture buffer. The data of the bottom-right four 8×8 blocks in Fig. A.10 are the reconstructed pixels. The procedure for chroma data is similar to that for luma data. In our method, we adopt an external buffer to store long-term "Temporal Neighboring Data" instead of the local buffer although some papers propose H.264/AVC deblocking filter design [46][47] adopt the local memory to store huge temporal data in order to reduce memory access cycle. While it reduces memory access cycles, it also requires a large amount of SRAM when processing a

high resolution frame because the buffer size depends on the width of the frame. Consider 1080p (1920×1080) for example: the width of frame is 1920 pixels, and the buffer size would be 15.36 KB (1920 x 2 x 32bits) for 8-bit pixel data. For practical reasons, we remove this buffer and allocate a data buffer in the external memory. This external memory buffer size could be the width of the picture multiplied by the data size of 8 pixels and then multiplied by 2 (for both luma and chroma data): Data_Buffer_Size = Frame_Width × Pixel_Size × 8 × 2. Reference [48] also adopts external memory to store temporal data.

According to the single processing order in Fig. A.9(a), the data (usually the third (Y2), fourth (Y3), fifth (Cb), and sixth (Cr) blocks of a reconstructed macroblock) will be written into temporal data buffers after reconstruction and partial filtering. It means that this system has almost one-half of luma data and all chroma data in a frame that should be written when processing the current macroblocks and read back again when processing future macroblocks.

Generally speaking, several parts of the decoding process will be implemented in hardware. They are Parser, Variable Length Decoder (VLD), Inverse DCT (IDCT), Motion Compensation (MC), and Deblocking Filter (DF). For calculation issues, the system loading of the last three parts is much larger than the other two parts if we perform these tasks with a software implementation. Although the loading of the first two parts is not large, many hardware resources are required to implement them, especially for dealing with countless VLC tables and registers of parsing syntax in H.264 or VC-1. It will be more efficient to perform these two parts with a software solution. This is why many architecture designs choose a combination of MC and DF or a combination of IDCT, MC, and DF for hardware implementation if they plan to economically use hardware resource. Before modifying the processing order within a frame or slice, we assume that the software takes charge of syntax decoding and VLD, and we assume the relative parameters are decoded prior to reconstruction and filtering a current frame by the hardware. While the hardware processes a current frame, the software is decoding all parameters and information for the next frame. For example, when hardware is processing frame n, software processes processed frame n+1 or successive frames

so as to pipeline the computation. The latency will be slightly increased as hardware operation may be initialized only after software has already processed several frames. Based on this approach, we propose the multiple macroblock processing order as an alternative single macroblock processing order.



(a) (b)

Figure A.9: (a) Single macroblock processing order. (b) Dual macroblock processing order.

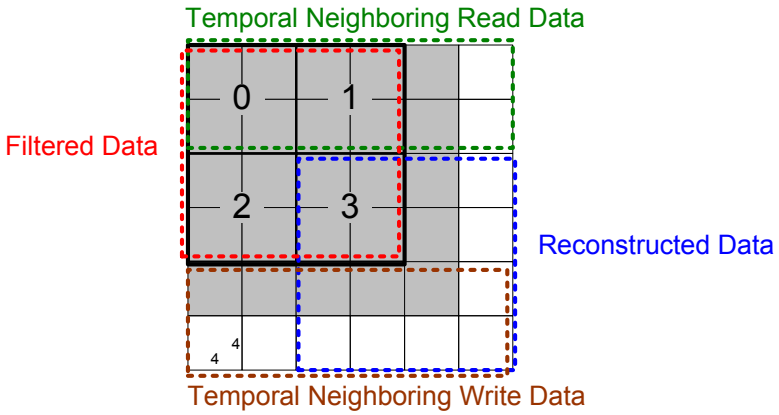



Figure A.10: Data processing for a reconstructed macroblock and a filtered macroblock.

Multiple processing order determines how many vertical macroblocks belong to an integrated unit and processes this unit in raster order. If the multiple processing order adopts two vertical macroblocks (dual blocks) per integrated unit, the dual macroblock processing order is shown in Fig. A.9(b). In this case, we reduce memory access from one-half to one-fourth of luma data and from one to

one-half of chroma data for a frame. In addition, no penalty is incurred for external memory size. Similarly, we can design our processing order in triple macroblock processing order and then reduce memory access from one-half to one-sixth of luma data and from one to one-third of chroma data. However, if we select more macroblocks into an integrated unit, we then require a larger on-chip memory size to implement. Regardless, it is still necessary to use more on-chip memory to mitigate memory loading issues when the bandwidth of system memory is limited for the overall system.

### A.3.4 Modified Chrominance Processing Order

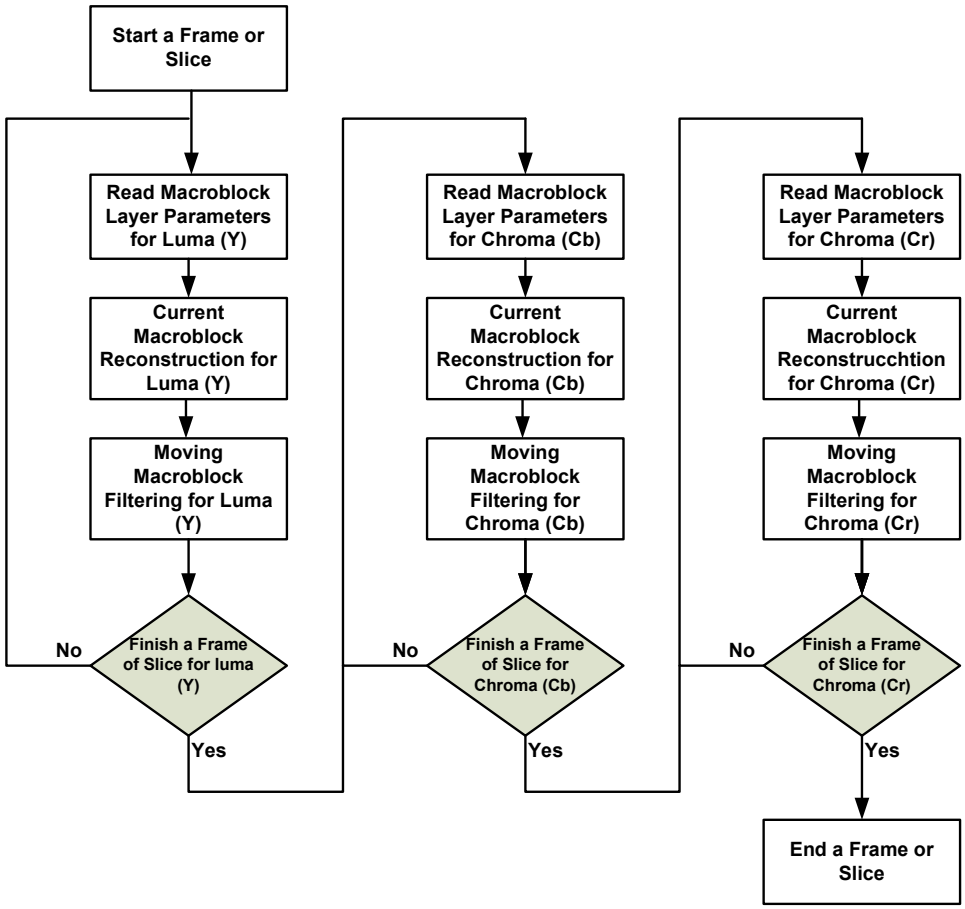



Figure A.11: Separated data processing flow for a picture or slice.

In order to go a step further in reducing external memory access cycles and on-chip memory size, we propose another modified processing order. Since a normal processing order alternately performs reconstructing steps and filtering steps among Y, Cb, and Cr data, it requires larger local memory size to store temporal data needed for the next macroblock in case it accesses external memory repetitively. Moreover, due to only one Cb and one Cr 8×8 block being processed within a macroblock, it also induces many additional memory cycles to perform processing. For these reasons, we propose reconstructing and filtering luma data of a frame or slice first, then processing chroma channel, Cb, and lastly processing the other chroma channel, Cr. This proposed procedure is shown in Fig. A.11. As we mentioned above, software should take charge of syntax decoding (Parsing) and Variable-length Decoding (VLD), and the parameters should be decoded before we reconstruct and filter a current frame or slice. Although software needs to access more external memory cycles for macroblock-layer parameters due to separating information of the three channels, it significantly reduces external memory cycles for temporal neighboring data. If a designer reorganizes the data format of macroblock-layer parameters for efficiently reading from memory, the external memory size may change very slightly for recording parameter information. Otherwise, there is no penalty for external memory size. This new method decreases local memory size because it only stores one kind of data for luma or chroma temporal data, and the size of the temporal data buffer in the external memory is reduced by half. Because of separating the processing procedure for Y, Cb, and Cr, we share the same architecture to process chroma macroblocks as luma macroblock and the same amount of local memory. The procedure for chroma data is shown in Fig. A.12 and called Y-analogous data procedure. Our method simplifies the decoding procedure and reduces the external memory requirement by one-half for temporal neighboring chroma data.

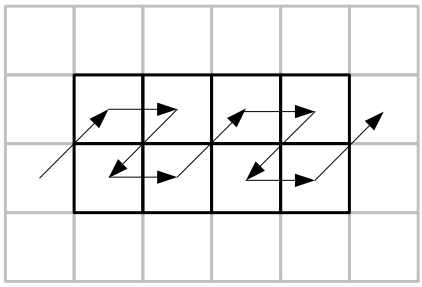


Figure A.12: Y-analogous data procedure for chroma data.

## A.4 Proposed Deblocking Filter Architecture and Implementation in VC-1

In this section, we prepare an architecture that implements all above proposed methods. Fig. A.13 demonstrates a block diagram of the proposed architecture performing both overlap smoothing and loop filtering based on an overlapped block. A Temporal Data Buffer stores unfiltered reconstructed data from the Motion Compensation Engine (MCE), unfiltered or partially filtered temporal neighboring data from external memory, and filtered or partially filtered data from our proposed integrated filter. MCE takes charge of reconstructing intra- or inter-coded blocks. The External Memory Controller manages read/write cycles from/to external memory, such as DDR2-SDRAM or DDR3-SDRAM. Both MCE and our proposed integrated filter connect with System Controller which has a HW/SW system interface, such as Memory-mapped I/O (MMIO), for communicating with the application or the driver. The Filter Control Unit is the core of the proposed integrated filter and includes two major finite state machines: one controls the flow among different overlapped blocks within a filtered macroblock; the other one is in charge of the schedule among different filtered edges within an overlapped block.

Consequently, there are three levels of memory access strategy for the overall system. The first level is system memory access. Memory controller takes unfiltered pixels from system memory (external memory) and saves this data into on-chip memory. The memory controller also write reconstructed and filtered pixels

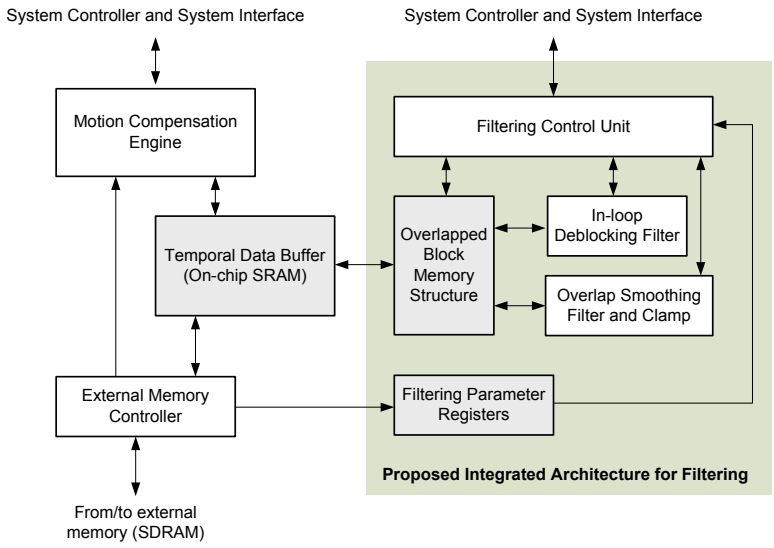



Figure A.13: Block diagram of the proposed integrated architecture.

back to system memory for display or storage. The second level is on-chip memory access. This dual-port on-chip memory (temporal data buffer) stores reconstructed unfiltered pixels and filtered pixels. The memory controller, Motion Compensation Engine, and proposed integrated architecture all access data through this on-chip memory. The third level is local memory access. Two phases, Load and Store, of processing an overlapped block will use synthesized dual-port local memory (Overlapped Block Memory Structure) to access the data of on-chip memory. With the exception of these two phases, local memory will not access on-chip memory. The data in local memory is used for performing filtering procedures for an overlapped block. These three levels are shown in Fig. A.14.

Before filtering an overlapped block, this architecture receives the related parameters previously stored by software in external memory and stores them in the Filtering Parameter Registers. These parameters are used to determine whether to perform overlap smoothing, clamping, or loop filtering an edge. The Overlapped Block Memory Structure is a local buffer storing all pixels within an overlapped block. The Filter Control Unit controls the Overlap Smoothing Filter and Clamp and In-loop Deblocking Filter to access input data from the Overlapped Block Memory Structure and proceed with filtering operations. In this implemen-

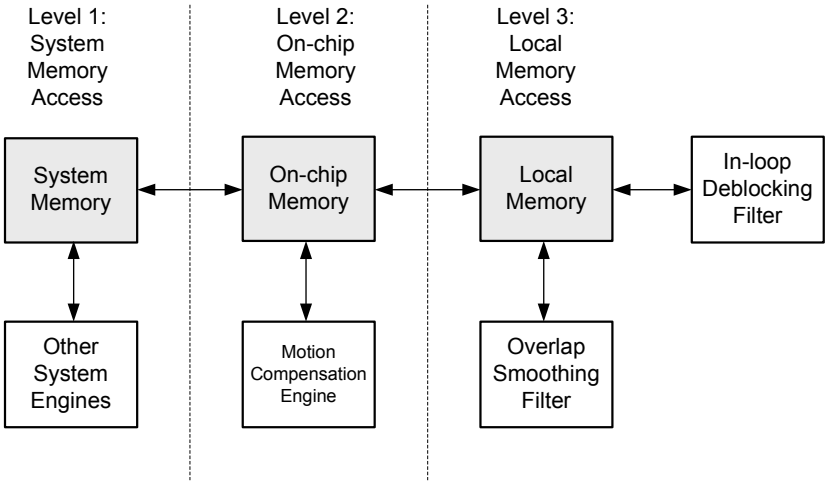



Figure A.14: Three hierarchical levels of memory access.

tation, each filtering operation (overlap smoothing or loop filtering) uses three cycles and outputs filtered data into the local memory at the fourth cycle. The filtering operation of the third pixel pair should be performed first on every edge of a 4-pixel segment because the result of this operation will determine whether the other three pixel pairs in the segment are filtered. We attempt to pipeline all filtering edges in order to reduce the processing time. However, switching different vertical and horizontal filtering edges may result in a reduction of data access speed. For solving this problem and getting input data at every cycle no matter which horizontal or vertical edge is filtered, our proposed architecture adopts a specific memory structure to implement the Overlapped Block Memory Structure.

Fig. A.15(a) shows nine 4×4 blocks within an overlapped block. We mark every other 4×4 block with gray, and filtered edges are the boundaries between any two different-color 4×4 blocks. Hence, we can separate all 4×4 blocks into two groups (memory structures L and R) shown in Fig. A.15(b). When Overlap Smoothing Filter or In-loop Deblocking Filter of Fig. A.13 filters an edge within an overlapped block, it accesses two pixels (overlap smoothing) or four pixels (loop filtering) from the L and R groups. Only one 4×4 block of the R or L group is involved in a filtering operation. A standard memory structure with a width of 32 bits will have several memory lines. Each data line may include for data pixels

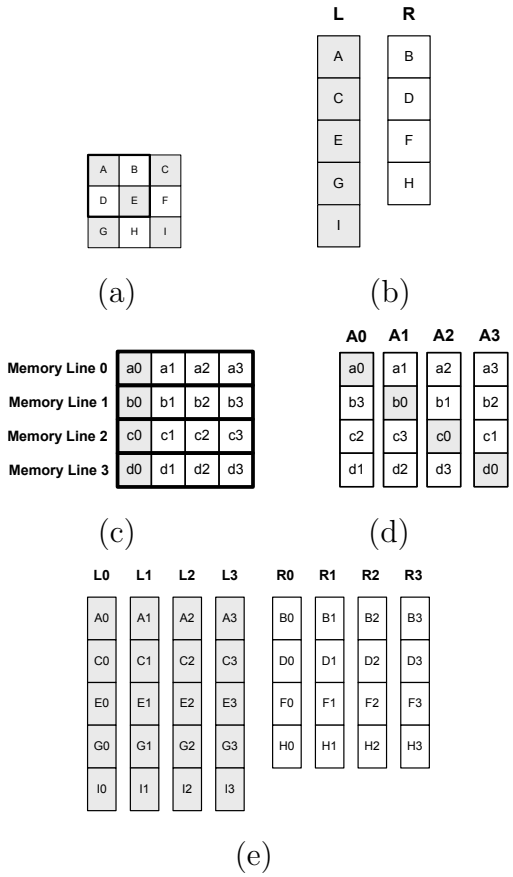



Figure A.15: (a) Nine 4×4 blocks within an overlapped block. Every filtered edge is the boundary between a gray 4×4 block and a white 4×4 block. (b) Two groups of memory structure. Filtering of an edge must obtain 4-pixels or 2-pixels data from each group. (c) Normal pixel mapping within a 4×4 block. (d) Rotated pixel mapping within a 4×4 block. (e) Proposed memory structure for an overlapped block including four 20×11bits and four 16×11bits memory structures.

at 8 bits each. In this case, our filters cannot obtain all input pixels in one cycle when a horizontal filtered edge is encountered. Fig. A.15(c) shows an example for normal pixel mapping within a 4×4 block. When filtering a horizontal edge, it accesses a0, b0, c0, and d0 for the input data, and it will spend more cycles accesses these pixels stored in different memory lines within the same memory structure. In our proposed architecture, we adopt an efficient method similar to [49] and rotate the order shown in Fig. A.15(d). We right-rotate by one pixel for the second row, two pixels for the third row, and three pixels for the last row. These pixels are separated into four memory structures. For the 4×4 block A of

Fig. A.15(a), the data of this block will be allocated into four memory structures (A0, A1, A2, and A3) shown in Fig. A.15(d). Because a0, b0, c0, and d0 are stored in different memory structures, the filters can access these pixels simultaneously to obtain all required input data regardless of whether a horizontal or vertical edge is selected for filtering. The structure in Fig. A.15(d) applies to all 4×4 blocks of an overlapped block, so we implement four 20×11bits (L0, L1, L2, and L3) and four 16×11bits (R0, R1, R2, and R3) memory structures in the Overlapped Block Memory Structure of Fig. A.13.

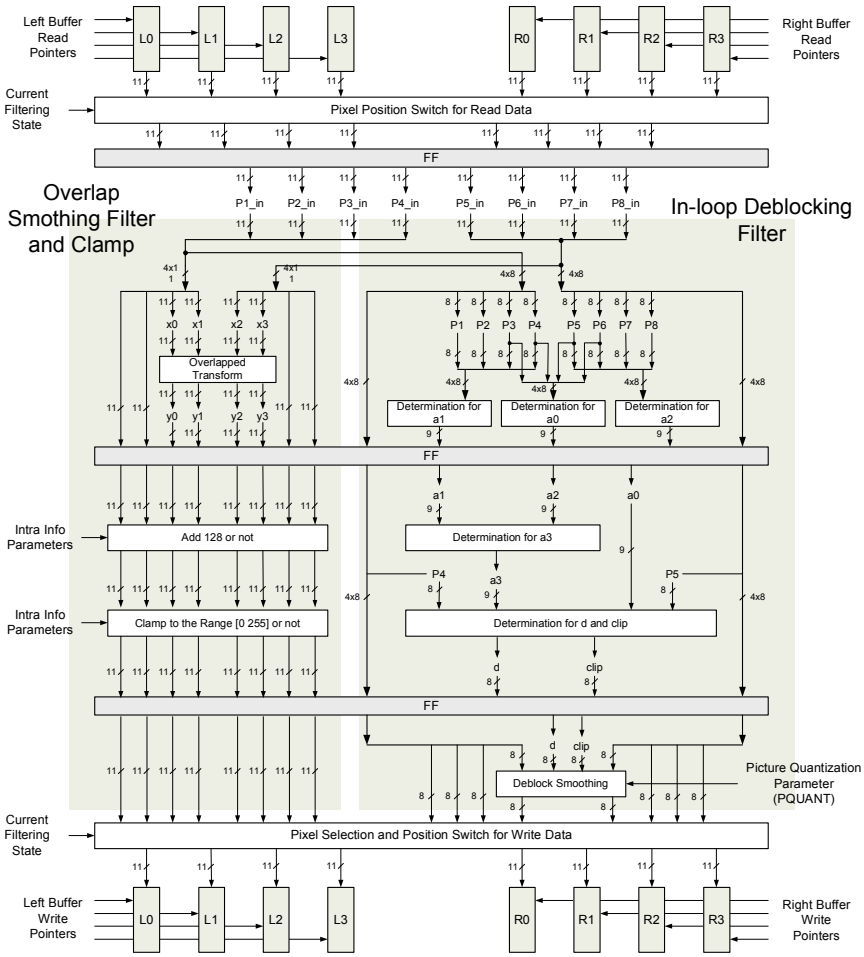



Figure A.16: Data flow of the Proposed Integrated Filter.

Fig. A.16 demonstrates the data flow of our proposed integrated architecture. This data flow begins with the L (L0, L1, L2, and L3) and R (R0, R1,

Table A.1: Front-end hardware cost of the proposed VC-1 filter architecture with TSMC 90-nm multi-Vt technology

| Functional Block | Gate Counts |
|---|---|
| Overlap Smooth and Clamp | 2956 (12.07%) |
| In-loop Deblocking Filter | 2718 (11.10%) |
| Filtering Control Unit and Others | 5568 (22.74%) |
| Overlap Block Memory Structure | 13248 (54.10%) |
| Total | 24490 |

R2, and R3) memory structures mentioned above and ends back to themselves after each filtering operation for overlap smoothing or loop filtering. The Filtering Control Unit has eight buffer read pointers to operate these memory structures, and the values of these pointers depend on the current state and which edge is being filtered. After reading data from the buffer, there are several multiplexers which rearrange the position of the data for the overlap smoothing filter or in-loop deblocking filter because the data has been rotated prior to storage in a memory structure. Four of these eight pixels (P3, P4, P5, and P6) are the input data (x0, x1, x2, and x3) of overlapped transform. When the pixels complete both horizontal and vertical filtering (or they belong to an intra-coded block without overlap smoothing), the constant value of 128 is added. The pixels are then clamped to 8 bits for the reconstructed output. All eight pixels from the memory structures are used for our in-loop deblocking filter. Finally, the output data from the overlap smoothing filter or in-loop deblocking filter are rearranged and stored into the memory buffer. This data flow is repeated until all filtered edge are complete for a frame or slice.

## A.5 Experimental Results and Analysis for the Proposed VC-1 Deblocking Filter

### A.5.1 Implementation and Performance

The specification of the proposed VC-1 filter is capable of HDTV1080p (1920×1080) 30fps video and HDTV 2048×1536 24fps video at 200MHz. To verify

the accuracy and efficiency of the proposed architecture, the proposed integrated architecture is designed in VHDL and implemented with TSMC 90-nm multi-threshold voltage technology. The implemented architecture operates with our VHDL- and C-model, and the result has been verified with the reference VC-1 decoder software. TABLE A.1 shows logic gate count including several functional blocks and a synthesized memory buffer in the proposed VC-1 filter synthesized with Cadence RTL complier at 200MHz. Total logic gate count including the synthesized memory buffer is about 24.49 K, and total logic gate count excluding the memory buffer is about 11.24 K. This synthesized memory buffer is an Overlapped Block Memory Structure composed of four 20×11bits and four 16×11bits memory structures with separated read/write ports, and it occupies 54.1% of the total area. After back-end routing and optimization using Cadence SOCE, the core area of proposed architecture without I/O Pads is 291$\mu$m × 264$\mu$m, and the die area is 311$\mu$m × 284$\mu$m. The core utilization rate is 0.894 based on 8-levels metal layout.

Table A.2: Processing time of the proposed VC-1 filter architecture

| Type (Worst Case) | Cycles | Time | Required Time |
|---|---|---|---|
| One Overlapped Block | 100 | 500 ns | - |
| One Filtered Macroblock | 607 | 3035 ns | - |
| 1080p, 30fps I, B-frame | 3829977 | 19.15 ms | $<$ 33.33 ms |
| 1080p, 30fps P-frame | 4771316 | 23.86 ms | $<$ 33.33 ms |
| 2048×1536, 24fps I, B-frame | 5767373 | 28.84 ms | $<$ 41.66 ms |
| 2048×1536, 24fps P-frame | 7184794 | 35.92 ms | $<$ 41.66 ms |

Filtering Control Unit controls the procedure of filtering operations. This block and the Overlap Block memory Structure share circuitry for two different filtering blocks. Without an architecture for sharing circuitry, the memory structure and control unit need to be duplicated, thereby increasing the cost by more than 70%.

For evaluating performance of the proposed architecture, we create several worst-case test patterns for I, B or P-frames in the two highest resolutions of VC-1, 1080p and 2048×1536. When performing these patterns, the filter will

Table A.3: Comparison with Different H.264/AVC Deblocking Architectures (The gate count excludes the local memory.)

| Function | [43] | [44] | [47] | [50] | Proposed |
|---|---|---|---|---|---|
| Codec | H.264 | H.264 | H.264 | H.264 | VC-1 OS + DF |
| Cycles per MB | 240 | 342 | 250 | 96 | 607 |
| Frequency (MHz) | 100 | 100 | 100 | 100 | 200 |
| Memory (bits) | 160×32 | 140×32 | 160×32 | 160×32 | 4×36×11 |
| Transpose Memory | 32×8×2 | 32×8 | 32×16 | No | No |
| Gate Count | 20.6K | 11.8K | 19.6K | 13.9K | 11.2K |
| Process ($\mu$m) | .25 | .18 | .18 | .18 | .09 |

process every boundary of all 8×8 blocks or 4×4 blocks for two kinds of filters. TABLE A.2 shows the processing cycles and processing time for different cases. HDTV1080p 30fps video requires less than 33.33ms of processing time, and HDTV 2048×1536 24fps video requires less than 41.66ms of processing time. Our proposed architecture meets the requirement even in the worst-case scenario.

Although there is no reference for the architecture of VC-1 filtering, we compare our proposed architecture with several different architecture designs for the H.264/AVC deblocking filter and show these comparisons in TABLE A.3. From this table, our architecture requires more cycles for one macroblock than previous architectures for H.264. This is because our architecture encounters more difficult problems and processes two different VC-1 filtering operations at the same time. In addition, our architecture operates at a higher clock frequency. However, our local memory size (Overlapped Block Memory Structure) is smaller than is required by other methods. The reason is that our method deals with the processing procedure one block at a time while all previous H.264 architectures deal with the entire macroblock at once.

### A.5.2 Resources Analysis

Besides efficiently processing overlap smoothing and loop filtering, one important design approach of our proposed method and architecture is to reduce

Table A.4: Analysis of external memory bandwidth for a worst-case 1080p 4:2:0 P-frame (without Overlap Smoothing)

| | Software | Method 1 | Method 2 | Method 3 | Method 4 |
|---|---|---|---|---|---|
| Required SRAM | - | 1.088 KB | 0.448 KB | 1.452 KB | 0.576 KB |
| Multiple MB | - | Single | Single | Dual | Dual |
| Modified CbCr | - | No | Yes | No | Yes |
| Write After MC | 3.13 MB | 2.06 MB | 1.53 MB | 1.01 MB | 0.75 MB |
| Read for DB | 12.49 MB | 2.06 MB | 1.53 MB | 1.01 MB | 0.75 MB |
| Write After DB | 3.12 MB | 3.13 MB | 3.13 MB | 3.13 MB | 3.13 MB |
| Total Access | 18.74 MB | 7.25 MB | 6.21 MB | 5.16 MB | 3.28 MB |

Table A.5: Analysis of external memory bandwidth for a worst-case 1080p 4:2:0 I-frame (Overlap Smoothing and In-loop Filtering)

| | Software | Method 1 | Method 2 | Method 3 | Method 4 |
|---|---|---|---|---|---|
| Required SRAM | - | 2.176 KB | 0.976 KB | 2.688 KB | 1.152 KB |
| Multiple MB | - | Single | Single | Dual | Dual |
| Modified CbCr | - | No | Yes | No | Yes |
| Write After MC | 6.27 MB | 4.12 MB | 3.07 MB | 2.03 MB | 1.51 MB |
| Read for OS | 6.22 MB | 4.12 MB | 3.07 MB | 2.03 MB | 1.51 MB |
| Write for OS | 6.22 MB | 0 MB | 0 MB | 0 MB | 0 MB |
| Read for Clamp | 6.27 MB | 0 MB | 0 MB | 0 MB | 0 MB |
| Write for Clamp | 3.13 MB | 0 MB | 0 MB | 0 MB | 0 MB |
| Read for DB | 6.22 MB | 0 MB | 0 MB | 0 MB | 0 MB |
| Write After DB | 1.55 MB | 3.13 MB | 3.13 MB | 3.13 MB | 3.13 MB |
| Total Access | 35.87 MB | 11.37 MB | 9.28 MB | 7.19 MB | 6.14 MB |

the memory loading of system resources, external memory bandwidth and on-chip SRAM size. As mentioned in Section A.3, we adopt multiple macroblock processing order, separate luma and chroma processing order, and use the Y-analogous data procedure for chroma data. In this section, we analyze the benefit of these proposed methods on system resources. The proposed are implemented and simulated as four distinct methods: method 1 adopts single macroblock processing order and does not introduce modified chrominance processing order; method 2 adopts single macroblock processing order and modified chroma processing order; method 3 adopts dual macroblock processing order and does not introduce modified chrominance processing order; method 4 adopts dual macroblock processing order and modified chrominance processing order. These proposed methods are compared with the bandwidth of a software implementation.

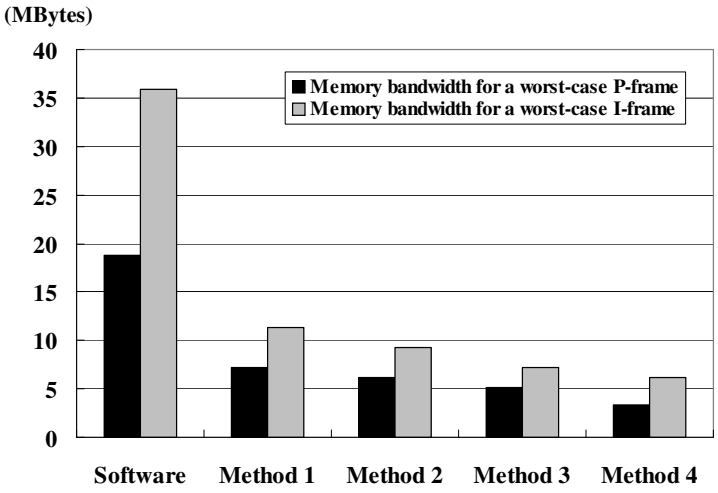



Figure A.17: Comparison about the requirement of memory bandwidth to the external memory.

TABLE A.4 shows the analysis of external memory bandwidth for a worst-case 1080p 4:2:0 P-frame without overlap smoothing. Compared with software, method 1 reduces external memory cycles by 61.31%, method 2 by 66.86%, method 3 by 72.47%, and method 4 by 82.5%. TABLE A.5 shows the analysis of external memory bandwidth for a worst-case 1080p 4:2:0 I-frame. Compared with software, method 1 reduces 68.3% of external memory cycles, method 2 reduces 74.13%, method 3 reduces 79.96%, and method 4 reduces 82.88%. Fig. A.17 shows the comparison of external memory bandwidth among these four proposed methods, and Fig. A.18 shows the comparison of required SRAM size (only for Temporal Data Buffer) among these four proposed methods. From these two comparisons, although using multiple macroblock processing order will increase the memory size, utilizing modified chrominance processing order would greatly reduce the memory size. Consequently, method 4 only requires 52.94% of method 1's SRAM size and reduces more than 82% of external memory cycles.

## A.6 Conclusions

We present several processing methods and an efficient integrated architecture for VC-1 filtering. These proposed methods are based on a 12×12 overlapped

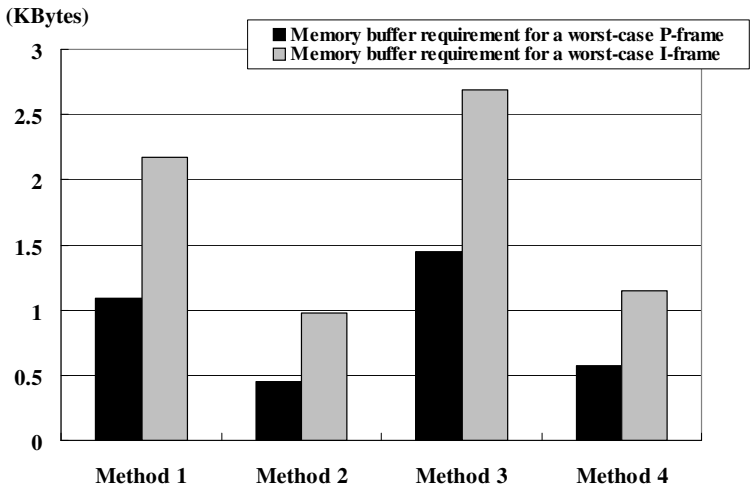



Figure A.18: Comparison about the SRAM requirement (Temporal Data Buffer).

block. In order to increase the performance and decrease the processing time, we integrate overlap smoothing with in-loop filtering and pipeline the procedure with block reconstructing even though the original filtering procedure is frame-based in the VC-1 standard. For efficiently utilizing system resources, we also propose two other methods: multiple processing order and modified chrominance processing order. Moreover, an integrated architecture is designed, synthesized, and implemented with TSMC 90-nm multi-Vt technology. The specification has the capability to process HDTV1080p 30fps video and HDTV 2048×1536 24fps video at 200MHz. In addition, we can apply these proposed methods to efficiently implement various deblocking filters in post-processing applications which have similar process order.

# Bibliography


[1] A. N. Netravali and J. D. Robbins, "Motion-adaptive interpolation of television frames," *Proc. Picture Coding Symp.*, pp.115. Jun. 1981.

[2] J.K. Su and R.M. Mersereau, "Motion-compensated interpolation of untransmitted frames in compressed video," *Conference Record of the Thirtieth Asilomar Conference on Signals, Systems and Computers*, Vol 1, pp. 100-104, Nov. 1996.

[3] Soo-Chul Han and J.W. Woods, "Frame-rate up-conversion using transmitted motion and segmentation fields for very low bit-rate video coding," *International Conference on Image Processing (ICIP)*, Vol 1, pp. 747-750, Oct. 1997.

[4] Yen-Kuang Chen, A. Vetro, Huifang Sun, and S.Y. Kung, "Frame-rate up-conversion using transmitted true motion vectors," *IEEE Second Workshop on Multimedia Signal Processing*, pp. 622-627, Dec. 1998.

[5] G. de Haan, P.W.A.C. Biezen, H. Huijgen, and O. A. Ojo, "True-motion estimation with 3-D recursive search block matching," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 3, Issue 5, pp. 368-379, 388, Oct. 1993.

[6] M. Soryani and R.J. Clarke, "Image segmentation and motion-adaptive frame interpolation for coding moving sequences," *1989 International Conference on Acoustics, Speech, and Signal Processing*, Vol. 3, pp. 1882-1885, May 1989.

[7] N. Grammalidis, D. Tzovarns, and M.G. Strintzis, "Temporal frame interpolation for stereoscopic sequences using object-based motion estimation and occlusion detection," *International Conference on Image Processing*, Vol. 2, pp. 382-379, 385, Oct. 1995.

[8] R. Castagno, P. Haavisto, and G. Ramponi, "A method for motion adaptive frame rate up-conversion," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 6, Issue 5, pp. 436-446, Oct. 1996.

[9] X. Q. Gao, C. J. Duanmu, and C. R. Zou, "A multilevel successive elimination algorithm for block matching motion estimation," *IEEE Transactions on Image Processing*, Vol. 9, Issue 3, pp. 501-504, March 2000.

[10] Y. Noguchi, J. Furukawa, and H. Kiya, "A fast full search block matching algorithm for MPEG-4 video," *Proceedings. 1999 International Conference on mage Processing. ICIP 99.*, Vol. 1, pp. 61-65, 1999.

[11] W. Li and E. Salari, "Successive elimination algorithm for motion estimation," *IEEE Transactions on Image Processing*, Vol. 4, Issue 1, pp. 105-107, Jan. 1995.

[12] L. De Vos and M. Stegherr, "Parameterizable VLSI architecture for the full-block matching algorithm," *IEEE Transactions on Circuits and Systems*, Vol. 36, pp. 1309-1316, Oct. 1989.

[13] Shay Har-Noy and T.Q. Nguyen, "LCD Motion Blur Reduction: A Signal Processing Approach," *IEEE Transactions on Image Processing*, Vol. 17, Issue 2, pp. 117-125, Feb. 2008.

[14] Tao Chen, "Adaptive temporal interpolation using bidirectional motion estimation and compensation," *Proceedings. 2002 International Conference on Image Processing. 2002.*, Vol. 2, pp. II313-II316, 2002.

[15] Bo-Won Jeon, Gun-Ill Lee, Sung-Hee Lee, and Rae-Hong Park, "Coarse-to-fine frame interpolation for frame rate up-conversion using pyramid structure," *IEEE Transactions on Consumer Electronics*, Vol. 49, Issue 3, pp. 499-508, 2003.

[16] Taehyeun Ha, Seongjoo Lee, and Jaeseok Kim, "Motion compensated frame interpolation by new block-based motion estimation algorithm," *IEEE Transactions on Consumer Electronics*, Vol. 50, Issue 2, pp. 752-759, May 2004.

[17] M. Biswas and T. Nguyen, "A novel motion estimation algorithm using phase plane correlation for frame rate conversion," *Conference Record of the Thirty-Sixth Asilomar Conference on Signals, Systems and Computers*, Vol. 1, , pp. 492-496, Nov. 2002.

[18] Jiefu Zhai, Keman Yu, Jiang Li, and Shipeng Li, "A low complexity motion compensated frame interpolation method," *IEEE International Symposium on Circuits and Systems, 2005. ISCAS 2005.*, Vol. 50, pp. 4927-4930, May 2005.

[19] Tien-Ying Kuo, JongWon Kim, C.-C.J. Kuo, "Motion-compensated frame interpolation scheme for H.263 codec," *Proceedings of the 1999 IEEE International Symposium on Circuits and Systems*, Vol. 4, pp. 491-494, May 1999.

[20] "Video Coding for Low Bit Rate Communication," ITU-T Rec. H.263.

[21] Fujiwara, S. and Taguchi, A., "Motion-compensated frame rate up-conversion based on block matching algorithm with multi-size blocks," *Proceedings of 2005 International Symposium on Intelligent Signal Processing and Communication Systems, 2005. ISPACS 2005.*, pp. 353-356, Dec. 2005.

[22] Byeong-Doo Choi, Jong-Woo Han, Chang-Su Kim, and Sung-Jea Ko, "Frame rate up-conversion using perspective transform," *IEEE Transactions on Consumer Electronics*, Vol. 52, Issue 3, pp. 975-982, Aug. 2006.

[23] Dane, G. and Nguyen, T.Q., "Optimal temporal interpolation filter for motion-compensated frame rate up conversion," *IEEE Transactions on Image Processing*, Vol. 15, Issue 4, pp. 978-991, April 2006.

[24] Choi, B.-D., Han, J.-W., Kim, C.-S., and Ko, S.-J., "Motion-Compensated Frame Interpolation Using Bilateral Motion Estimation and Adaptive Overlapped Block Motion Compensation," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 17, Issue 4, pp. 407-416, April 2007.

[25] Ya-Ting Yang, Yi-Shin Tung, and Ja-Ling Wu, "Quality Enhancement of Frame Rate Up-Converted Video by Adaptive Frame Skip and Reliable Motion Extraction," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 17, Issue 12, pp. 1700-1713, Dec. 2007.

[26] Z. Gan, L. Qi, and X. Zhu, "Motion compensated frame interpolation based on H.264 decoder," *Electronics Letters*, Vol. 43, Issue 2, pp. 96-98, Jan. 2007.

[27] Ai-Mei Huang and T.Q. Nguyen, "A Multistage Motion Vector Processing Method for Motion-Compensated Frame Interpolation," *IEEE Transactions on Image Processing*, Vol. 17, Issue 5, pp. 694-708, May 2008.

[28] Chunbo Yang, Pin Tao, and Shiqiang Yang, "An adaptive frame interpolation algorithm using statistic analysis of motions and residual energy," *2008 IEEE 10th Workshop on Multimedia Signal Processing*, pp. 235-240, Oct. 2008.

[29] V. Munoz-Jimenez, A. Mokraoui-Zergainoh, and J.-P. Astruc, "Bidirectional Motion Estimation Approach Using Warping Mesh Combined to Frame Interpolation," *IEEE International Symposium on Signal Processing and Information Technology*, pp. 249-253, Dec. 2008.

[30] H. Song, A. Men, and Guangjian Shi, "A method for halo artifact reduction in MEMC," *Digest of Technical Papers International Conference on Consumer Electronics*, pp. 1-2, Jan. 2009.

[31] Ai-Mei Huang and T.Q. Nguyen, "Correlation-Based Motion Vector Processing With Adaptive Interpolation Scheme for Motion-Compensated Frame Interpolation," *IEEE Transactions on Image Processing*, Vol. 18, Issue 4, pp. 740-752, April 2009.

[32] *Advanced Video Coding for Generic Audiovisual Services, ITU-T Rec. H.264/ISO/IEC 14496-10*, Mar. 2005.

[33] *VC-1 Compressed Video Bitstream Format and Decoding Process (SMPTE 421M-2006)*, SMPTE Standard, 2006.

[34] P. Lakamsani, "An architecture for enhanced three step search generalized for hierarchical motion estimation algorithms," *IEEE Transactions on Consumer Electronics*, Vol. 43, Issue 2, pp. 221-227, May 1997.

[35] H. So, J. Kim, W.-K. Cho, and Y.-S.Kim, "Fast motion estimation using modified diamond search patterns," *Electronics Letters*, Vol. 41, Issue 2, pp. 62-63, Jan. 2005.

[36] ITU-R Recommendation BT.500-11. "Methodology for the subjective assessment of the quality of television pictures", in Geneva, 2002.

[37] Sunkwang Hong, B. Berkeley, and Sang Soo Kim "Motion image enhancement of LCDs", *IEEE International Conference on Image Processing, 2005. ICIP 2005*, Vol. 2, , pp. 17-10, Sept. 2005.

[38] S. Srinivasan, S. L. Regunathan, "An overview of VC-1," Visual Communications and Image Processing, Proc. of SPIE, Vol. 5950, pp.720-728, 2005.

[39] P. List, A. Joch, J. Lainema, G. Bjontegaard, and M. Karczewicz, "Adaptive deblocking filter," *IEEE Trans. on Circuits and Sytems for Video Technology*, Vol.13, pp.614-619, 2003.

[40] T. D. Tran, J. Liang, and C. Tu, "Lapped transform via time-domain pre- and post-filtering", *IEEE Trans on Signal Processing*, Vol.51, pp.1557-1571, 2003.

[41] C. C. Cheng, T. S. Chang, and K. B. Lee, "An in-place architecture for the deblocking filter in H.264/AVC," *IEEE Trans. on Circuits and Systems-: Express Briefs*, Vol.53, pp.530-534, 2006.

[42] T. C. Chen, S. Y. Chien, Y. W. Huang, C. H. Tsai, C. Y. Chen, and L. G. Chen, "Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder," *IEEE Trans. on Circuits and Systems for Video Technology*, Vol.16, pp.673-688, 2006.

[43] Y.-W. Huang, T.-W. Chen, B.-Y. Hsieh, T.-C. Wang, T.-H. Chang, and L.-G. Chen, "Architecture design for deblocking filter in H.264/JVT/AVC," *Proceedings. 2003 International Conference on Multimedia and Expo*, 2003.

[44] S.-C. Chang, W.-H. Peng, S.-H. Wang, and T. Chiang, "A platform based bus-interleaved architecture for de-blocking filter in H.264/MPEG-4 AVC," *IEEE Trans. on Consumer Electronics*, Vol.51, pp.249-255, 2005.

[45] M. Sima, Y. Zhou, and W. Zhang, "An efficient architecture for adaptive deblocking filter of H.264/AVC video coding," *IEEE Trans. on Consumer Electronics*, Vol.50, pp.292-296, 2004.

[46] S.-Y. Shih, C.-R. Chang, and Y.-L. Lin, "A near optimal deblocking filter for H.264 advanced video coding," *Asia and South Pacific Design Automation Conference*, pp.170-175, 2006.

[47] T.-M. Liu, W.-P. Lee, T.-A. Lin, and C.-Y. Lee, "A memory-efficient deblocking filter for H.264/AVC video coding," *IEEE International Symposium on Circuits and Systems*, Vol. 3, pp.2140-2143, 2005.

[48] T.-M. Liu, T.-A. Lin, S.-Z. Wang, W.-P. Lee, J.-Y. Yang, K.-C. Hou, and C.-Y. Lee, "A 125 $\mu$W Fully Scalable MPEG-2 and H.264/AVC Video Decoder for Mobile Applications," *IEEE Journal of Solid-State Circuits*, Vol. 42, pp. 161-169, 2007.

[49] Lingfeng Li, S. Goto, and T. Ikenaga, "An efficient deblocking filter architecture with 2-dimensional parallel memory for H.264/AVC," *Asia and South Pacific Design Automation Conference*, Vol. 1, pp. 623-626, 2005.

[50] H.-Y. Lin, J.-J. Yang, B.-D. Liu, and J.-F. Yang, "Efficient Deblocking Filter Architecture for H.264 Video Coders," *Canadian Conference on Electrical and Computer Engineering*, pp. 2017-2020, 2006.

[51] T.-M. Liu, W.-P. Lee, and C.-Y. Lee, "An In/Post-Loop Deblocking Filter With Hybrid Filtering Schedule," *IEEE Trans. on Circuits and Systems for Video Technology*, vol.17, no.7, pp.937-943, July 2007.