AN ADAPTIVE TRUE MOTION ESTIMATION ALGORITHM FOR FRAME RATE UP-CONVERSION AND ITS HARDWARE DESIGN

by

MERT ÇETİN

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

Sabancı University

August 2009

AN ADAPTIVE TRUE MOTION ESTIMATION ALGORITHM FOR FRAME RATE UP-CONVERSION AND ITS HARDWARE DESIGN

APPROVED BY

Yard. Doç. Dr. İlker HAMZAOĞLU ……..
(Thesis Supervisor)

Yard. Doç. Dr. Hakan ERDOĞAN ……..

Yard. Doç Dr. Ahmet ONAT ……..

Doç. Dr. Meriç ÖZCAN ……..

Doç. Dr. Erkay SAVAŞ ……..

DATE OF APPROVAL: ……..

AN ADAPTIVE TRUE MOTION ESTIMATION ALGORITHM FOR FRAME RATE UP-CONVERSION AND ITS HARDWARE DESIGN

Mert ÇETİN

EE, MS Thesis, 2009

Thesis Supervisor: Assist. Prof. Dr. İlker HAMZAOĞLU

Keywords: Frame rate up conversion, true motion estimation, adaptive motion estimation, hardware architecture.

Abstract

With the advancement in video and display technologies, recently flat panel High Definition Television (HDTV) displays with 100 Hz, 120 Hz and most recently 240 Hz picture rates are introduced. However, video materials are captured and broadcast in different temporal resolutions ranging from 24 Hz to 60 Hz. In order to display these video formats correctly on high picture rate displays, new frames should be generated and inserted into the original video sequence to increase its frame rate. Therefore, Frame Rate Up-Conversion (FRUC) has become a necessity. Motion Compensated FRUC algorithms provide better quality results than non-motion compensated FRUC algorithms. Motion Estimation (ME) is the process of finding motion vectors which describe the motion of the objects between adjacent frames and is the most computationally intensive part of motion compensated FRUC algorithms. For FRUC applications, it is important to find the motion vectors that represent real motions of the objects which is called true ME. In this thesis, an Adaptive True Motion Estimation (ATME) algorithm is proposed. ATME algorithm produces similar quality results with less number of calculations or better quality results with similar number of calculations compared to 3-D Recursive Search true ME algorithm by adaptively using optimized sets of candidate search locations and several redundancy removal techniques. In addition, 3 different complexity hardware architectures for ATME are proposed. The proposed hardware use efficient data re-use schemes for the non-regular data flow of ATME algorithm. 2 of these hardware architectures are implemented on Xilinx Virtex-4 FPGA and are capable of processing ~158 and ~168 720p HD frames per second respectively.

GÖRÜNTÜ HIZI ARTIRIMI İÇİN UYARLANIR GERÇEK HAREKET TAHMİNİ ALGORİTMASI VE DONANIM TASARIMI

Mert ÇETİN

EE, Yüksek Lisans Tezi, 2009

Tez Danışmanı: Yard. Doç. Dr. İlker HAMZAOĞLU

Anahtar Kelimeler: Görüntü hızı artırımı, gerçek hareket tahmini, uyarlanır hareket tahmini, donanım tasarımı.

ÖZET

Video ve ekran teknolojilerindeki ilerlemeler sayesinde, yakın zamanlarda 100 Hz, 120 Hz, ve en yeni olarak da 240 Hz görüntü hızlarına sahip düz ekran Yüksek Çözünürlüklü Televizyon (YÇT) ekranları piyasaya çıkarıldı. Fakat video görüntüleri 24 Hz'den 60 Hz'e değişen farklı zamansal çözünürlüklerde kaydedilmekte ve yayınlanmaktadır. Bu farklı video biçimlerini yüksek görüntü hızlı ekranlarda doğru bir şekilde görüntülemek için, yeni kareler yaratılmalı ve görüntü hızını artırabilmek için video diziminin içine eklenmelidir. Bu yüzden Görüntü Hızı Artırımı (GHA) bir ihtiyaç olmuştur. Hareket Destekli GHA algoritmaları, hareket desteği olmayan GHA algoritmalarına oranla daha yüksek kaliteli sonuçlar vermektedir. Hareket Tahmini (HT), nesnelerin ardışık kareler boyunca hareketlerini tanımlayan hareket vektörlerini bulma işlemidir ve de Hareket Destekli GHA algoritmalarının işlemsel olarak en yoğun kısmını oluşturur. GHA uygulamaları için önemli olan nesnelerin gerçek hareketlerini ifade eden hareket vektörlerinin bulunabilmesidir. Buna Gerçek HT denir. Bu tezde Uyarlanır Gerçek Hareket Tahmini (UGHT) algoritması önerilmektedir. UGHT algoritması kullanıldığında, en uygun hale getirilmiş aday arama konumları kümelerinden ve de birtakım artıklık azaltıcı tekniklerden uyarlanır bir şekilde yararlanılıp, 3-D Recursive Search Gerçek HT algoritmasıyla karşılaştırıldığında daha az işlem yapılarak benzer kalitede sonuçlar veya da benzer sayıda işlem yapılarak daha yüksek kalitede sonuçlar elde edilmektedir. Ek olarak, UGHT için değişik karmaşıklığa sahip 3 farklı donanım mimarisi önerilmektedir. Önerilen donanımlarda UGHT algoritmasının düzenli olmayan veri akışı için verilerin verimli yeniden kullanımı için yöntemler uygulanmaktadır. Bu tasarımlardan 2'si Xilinx Virtex-4 FPGA üzerinde gerçeklenmiş ve de saniyede sırasıyla yaklaşık olarak 158 ve 168 720p YÇ çerçeve işleyebilmektedirler.

Acknowledgements

First and foremost I would like to thank my advisor Dr. İlker Hamzaoğlu for his invaluable guidance and support throughout my study. He made me realize that everything is possible with hard work and discipline. He has been a great mentor to me and I feel privileged to be his student.

I am sincerely grateful to my thesis committee members, Dr. Hakan Erdoğan, Dr. Dr. Ahmet Onat, Dr. Meriç Özcan, and Dr. Erkay Savaş, for their invaluable feedback.

I would like to thank to all members of System-on-Chip Design and Testing Lab, Yusuf Adıbelli, Çağlar Kalaycıoğlu, Murat Can Kıral, Kadir Akın, Aydın Aysu and Onur Can Ulusel who have been greatly supportive during my study. I also would like to thank Sibel Karadağ and Tolga Eren who were always there for me and provided me with endless motivation.

I would also like to express my deepest gratitude for my beloved family who always believed in me, and always tried their best to make things easier for me.

Finally I would like to acknowledge Sabancı University and TÜBİTAK for supporting me throughout my graduate education.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

2MI	:	Two-Mode Interpolation
3DRS	:	3-D Recursive Search
ATME	:	Adaptive True Motion Estimation
Bi-ME	:	Bilateral Motion Estimation
BM	:	Block Matching
CB	:	Current Block
CF	:	Current Frame
DMF	:	Dynamic Median Filter
FRUC	:	Frame Rate Up-Conversion
FS	:	Full Search
HD	:	High Definition
LFSR	:	Linear Feedback Shift Register
MC-FAVG	:	Motion Compensated Field Averaging
MC-FRUC	:	Motion Compensated Frame Rate Up-Conversion
MCI	:	Motion Compensated Interpolation
ME	:	Motion Estimation
MSE	:	Mean Squared Error
MV	:	Motion Vector
PB	:	Previous Block
PE	:	Processing Element
PF	:	Previous Frame
SAD	:	Sum of Absolute Differences
SD	:	Standard Definition
SMF	:	Static Median Filter
SW	:	Search Window
PSNR	:	Peak Signal-to-Noise Ratio
UW	:	Update Window
VMF	:	Vector Median Filter

Chapter 1

INTRODUCTION

The advancements in VLSI technology enabled the production of many multimedia products which introduced many video formats with different spatial and temporal resolutions. These formats include two main Standard Definition (SD) TV broadcast formats (50 Hz and 60 Hz with 625 and 525 lines respectively), and High Definition TV (HDTV) formats (720p and 1080i). The movie materials are recorded at 24, 25 or 30 frames per second. On the other hand, the advancement in display technologies enabled the production of large flat panel High Definition Television (HDTV) and PC displays with up to 100, 120 and most recently 240 Hz non-interlaced picture rates.

In order to display these formats correctly on high picture rate panels, new frames should be generated and inserted into the original sequence to increase its frame rate. Therefore, Frame Rate Up-Conversion (FRUC) has become a necessity [1]. An example FRUC scheme in which the frame rate of the input video sequence is multiplied by 4 is shown in Figure 1.1.

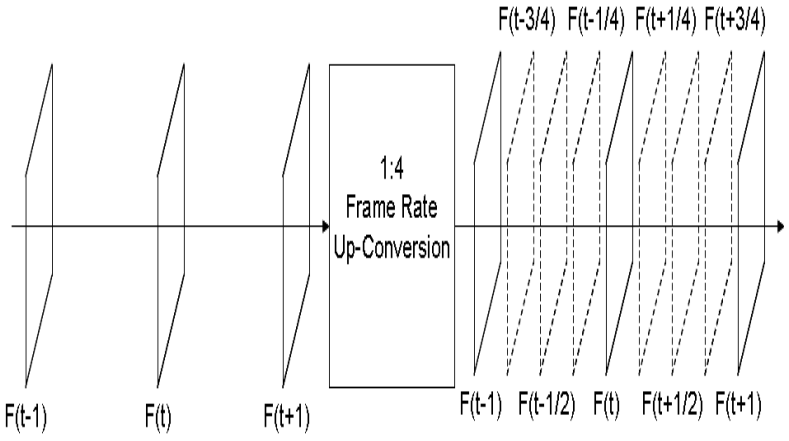
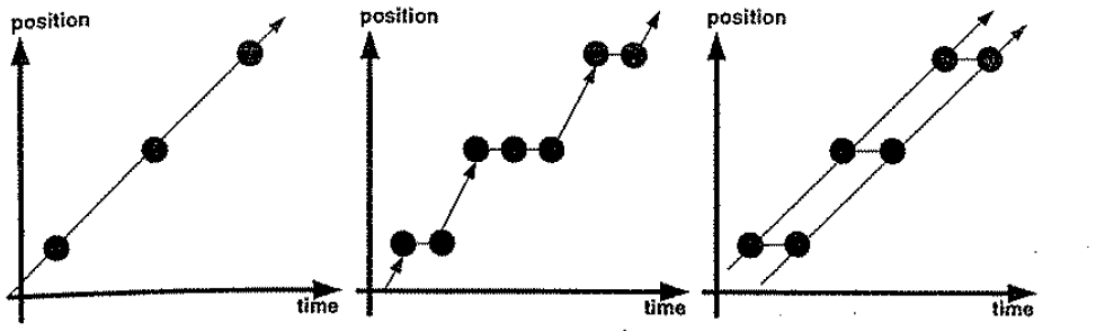


Figure 1.1: An Example FRUC System

The existing FRUC algorithms are mainly classified into two types [2]. First class of algorithms does not take motion of the objects into account, like frame repetition [3] or linear interpolation [4]. These algorithms are easy to implement without any significant computational cost, however at high spatial and temporal resolutions, these algorithms produce visual artifacts [5] like motion judder (if the difference between input and output frame rate is below 30 Hz) and motion blur (for higher differences). Figure 1.2 [1] shows the effect of these two situations.

In Figure 1.2(a) the original sequence is shown, where the linear motion of an object is illustrated as a straight line for 3 frames. In Figure 1.2(b), the case where the motion of the object is recorded by a 24 frames per second (fps) camera and displayed on a 60 Hz display is shown. When picture repetition is applied, some frames will be displayed two times and some will be displayed three times. This is called a 2-3 pull down [6]. In this case the viewer will experience an irregular or jerky motion which is called motion judder. On the other hand, in Figure 1.2(c), the case where a 50 Hz video is displayed on a 100 Hz display using picture repetition is shown. In this case, the viewer will experience a smooth motion, as the difference between input and output frame rates is higher than 30 Hz. However, the object will be perceived in both positions moving in parallel simultaneously, which will result in a double or blurred object. This is called motion blur.



(a) **(b)** **(c)**

Figure 1.2: Effect of Picture Repetition (a) Original sequence (b) Picture repetition from 24 Hz to 60 Hz (c) Picture repetition from 50 Hz to 100 Hz.

Second class of FRUC algorithms takes the motion of objects into account to reduce these artifacts and construct higher quality interpolated frames [2]. These Motion Compensated Frame Rate Up-Conversion (MC-FRUC) algorithms consist of two main stages, Motion Estimation (ME) and motion compensated interpolation (MCI). In ME, a

Motion Vector (MV) is calculated between successive frames, and in the MCI step this MV data from the previous step is used to generate a new frame to be inserted between the initial two successive frames, thus doubling the frame rate. This operation can be repeated to further increase the frame rate. In addition to the two main steps, there may be intermediate steps to improve the quality of the interpolated video output. These intermediate steps generally involve refinement of the MV field by various algorithms like Motion Vector Smoothing and Bilateral ME Refinement.

Among several ME algorithms, Block Matching (BM) is the most preferred method, which divides the frames of video sequences into *NxN* pixel blocks and tries to find the best matching block according to a cost function from previous frames inside a given search range. The most common cost function is Sum of Absolute Differences (SAD), because of its low computational cost.

There are various BM algorithms proposed in the literature. Full Search (FS) algorithm has the best performance as it exhaustively searches every location in the given search range [1]. However, its computational complexity is very high, especially for HD videos. On the other hand, many fast block matching algorithms are available [7-10], which have much less computational complexity while producing acceptable quality results. When motion vectors are generated for FRUC applications, it is important that the vectors represent real motions of the objects [1]. This is called the true motion. Although, these algorithms find the best SAD match which is sufficient for video compression, this does not guarantee that those vectors represent the true motion of the object. Therefore, generally, these algorithms perform poorly when used in frame rate up-conversion applications.

There are several ME algorithms [11-15] which aim to extract the true motion information between the frames of video sequences. These algorithms depend on two assumptions. The objects are larger than blocks so that surrounding neighbors of a block should have similar motions, and motions are continuous and spread through a duration of time so that blocks in successive frames of a video sequence should have similar motions. A recursive search algorithm takes advantage of these assumptions, and for the current block evaluates the motion vectors of spatial and temporal neighboring blocks instead of doing an exhaustive or static patterned search. 3-D Recursive Search (3DRS) [11] is one of the best implementations of these assumptions, and produces a smooth and accurate motion vector field suitable for MC-FRUC applications.

In this thesis, an adaptive true motion estimation algorithm (ATME) based on 3DRS is proposed. The candidate locations set of the 3DRS algorithm is optimized using a multi-objective genetic algorithm optimization [16], in order to produce high quality results with low computational costs. The optimized search location candidates are then integrated into an adaptive recursive search algorithm, which applies appropriate sets of search candidates, according to the smoothness and quality of the previous vector field. In addition, several computational complexity reduction and redundancy removal techniques are used for reducing the number of SAD calculations in single and multiple passes of the algorithm. One of these techniques also implicitly results in increasing smoothness of the motion vector field. Simulation results show that ATME algorithm generates similar quality results with lower computational costs or higher quality results with same computational costs compared to the 3DRS algorithm.

In addition, 3 different complexity hardware architectures for ATME are proposed. The first architecture is a basic implementation of ATME algorithm and is able to process ~158 720p HD frames per second. The second architecture uses an on-chip memory for efficient data re-use of pixel data for MVs that are close in value reducing the number of accesses to the off-chip SRAM which is costly both in terms of latency and power consumption. This architecture processes ~168 720p HD frames per second. Finally, a more complex architecture for use with large number of candidate search locations and large size video frames is proposed. This architecture uses a large on-chip search window memory for implementing a highly efficient data re-use scheme. The pixels are placed diagonally [17] in this search window memory to enable single cycle access to a row or column at any location inside the search window.

The rest of the thesis is organized as follows. In Chapter 2, ME algorithms, MCI algorithms, and several refinement steps used in MC-FRUC systems are explained in detail. In addition, video quality evaluation methods and metrics are presented. In Chapter 3, the ATME algorithm and its performance evaluation is presented. In addition, the software developed for implementation and testing of FRUC algorithms is explained. In Chapter 4, hardware implementations for ATME are presented in detail. Finally, Chapter 5 concludes this thesis.

Chapter 2

MOTION COMPENSATED FRAME RATE UP-CONVERSION

2.1 Motion Estimation

Motion estimation is the process of determining motion vectors that describe the transformation from one video frame to another, usually between adjacent frames in a video sequence. In Figure 2.1, a motion vector (MV) is shown as the motion trajectory which is the line that connects identical parts in adjacent frames. The estimation of these MVs is a difficult problem as the motion is in three dimensions but the images are a projection of the 3D scene onto a 2D plane. The MVs may relate to the whole image such as global motion, zooming or panning, or specific parts such as rectangular blocks, arbitrary shaped objects or even a pixel [1].

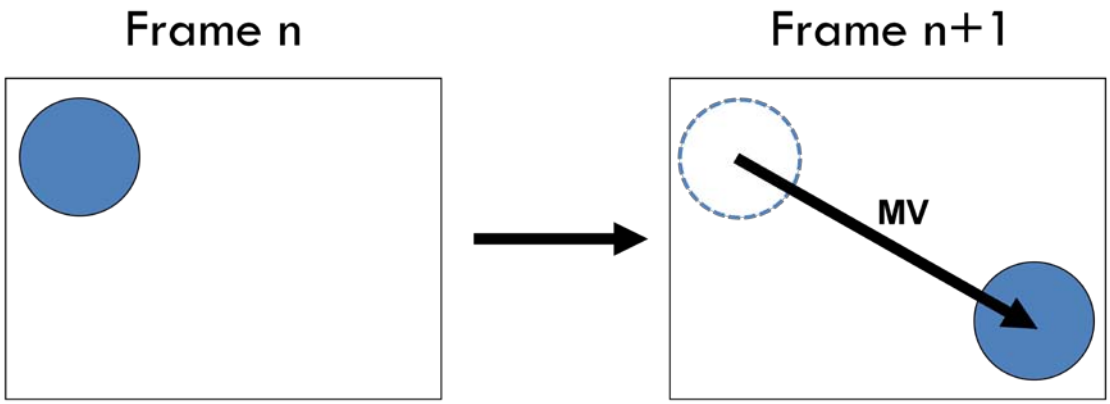


Figure 2.1: Motion Trajectory

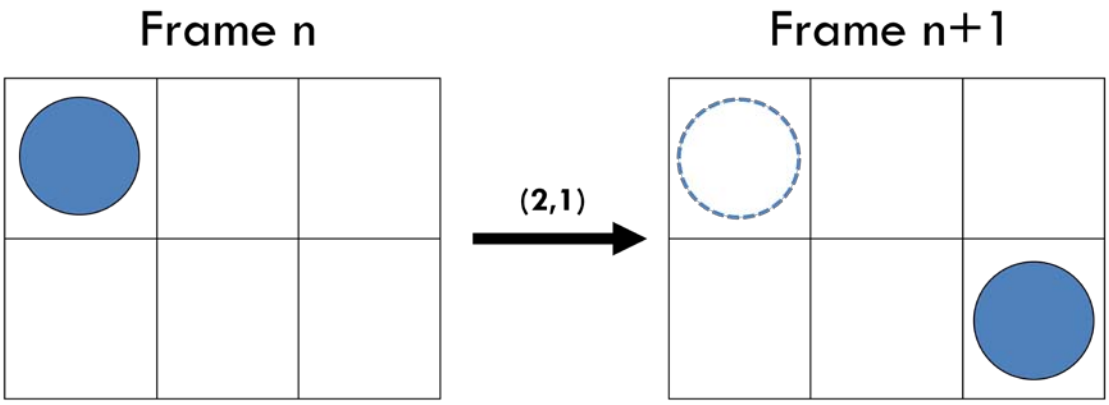


Figure 2.2: Motion Vector in BM Algorithms

Pixel based ME methods [18] involve significant calculations which makes them hard to implement both in software and hardware. Object based motion estimation [19] is an emerging method. But, the initial requirement of object based ME, the object segmentation, is a computationally demanding task. The block based motion estimation is the most preferred method in the literature and also in the industry due to its easy implementation and high quality results. The block based ME methods use Block Matching (BM) Algorithms, which divide the frames of video sequences into NxN pixel blocks and try to find the best matching block according to a cost function from previous frames inside a given search range. An example MV found by a BM algorithm is shown in Figure 2.2. The most common cost function is Sum of Absolute Differences (SAD) shown in Equation (2.1), because of its low computational complexity. The pixels inside a block $B(\vec{X})$ are assumed to have the same MV, which is assigned to $B(\vec{X})$ by BM algorithms.

$$SAD(\vec{v}, \vec{X}, n) = \sum_{\vec{x} \in B(\vec{X})} |F(\vec{x}, n) - F(\vec{x} - \vec{v}, n-1)| \quad (2.1)$$

Full Search (FS) algorithm is based on computing SADs at all possible locations in a given search window. It takes a block $B(\vec{X})$ in the current frame *n*, whose top left pixel is at position $\vec{X}$ and compares it to every block in the previous frame, *n-1*, inside a pre-defined search area $SA(\vec{X})$ which is also centered at $\vec{X}$. The motion trajectory connecting the best matching block (with the minimum SAD) in the previous frame with the current block $B(\vec{X})$ is assigned as the Motion Vector *V* of $B(\vec{X})$. This process is illustrated in Figure 2.3 [1]. The definition of full search is given in Equations (2.2) and

(2.3), where *C* denotes the candidate motion vectors pointing to possible search locations inside the search area *SA*, *N* and *M* denotes width and height of *SA* respectively, *V* denotes the selected MV.

$$\overrightarrow{SA} = \{\vec{C} | (X_x - N) \leq C_x \leq (X_x + N), (X_y - M) \leq C_y \leq (X_y + M)\} \quad (2.2)$$

$$\vec{V} = arg\ \min_{\vec{v} \in \overrightarrow{SA}} \{SAD(\vec{v}, \vec{X}, n)\} \quad (2.3)$$

FS guarantees finding the minimum SAD value inside a given search range. However, it is not designed to extract the true motion of the objects between frames and it is computationally expensive as it exhaustively evaluates every possible MV candidate.

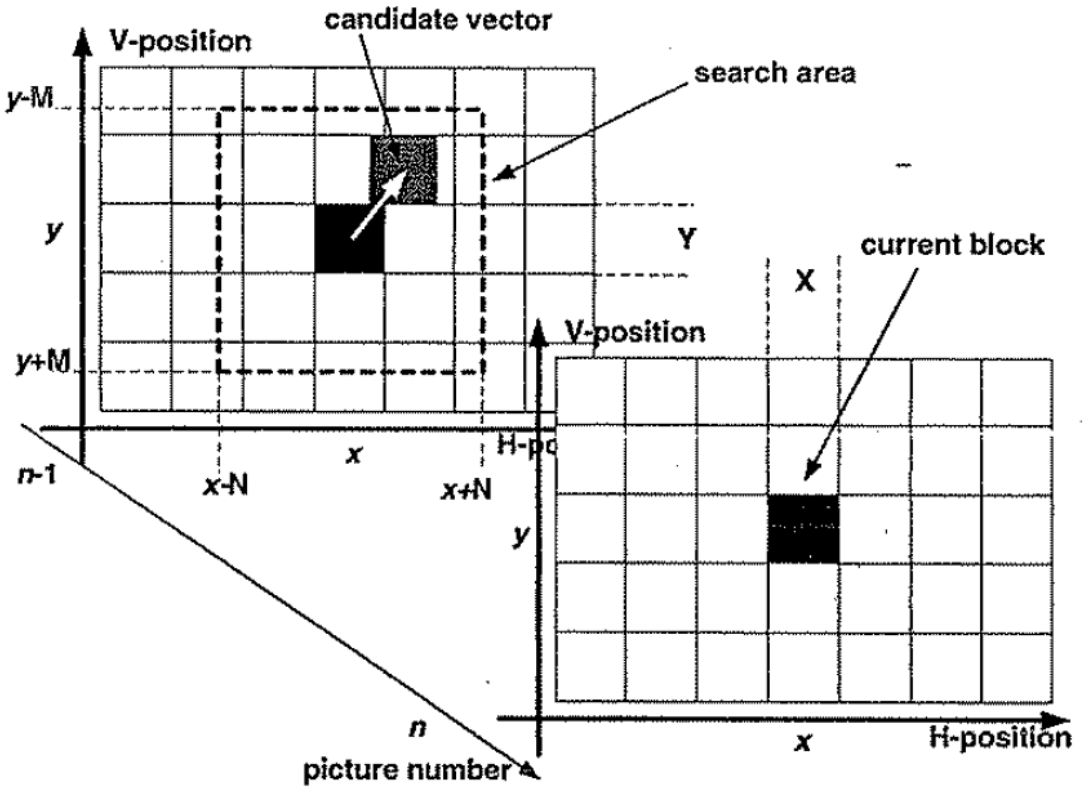


Figure 2.3: Full Search ME

The high computational complexity of the FS algorithm created the need for fast ME methods which try to achieve similar quality results with less computational complexity. There are many proposed fast ME methods [7-10] in the literature. For example, N-step search methods initially apply coarse search patterns, and continue

with finer patterns starting with the location found in the previous step. 3-step search pattern [7] is illustrated in Figure 2.4 [1].

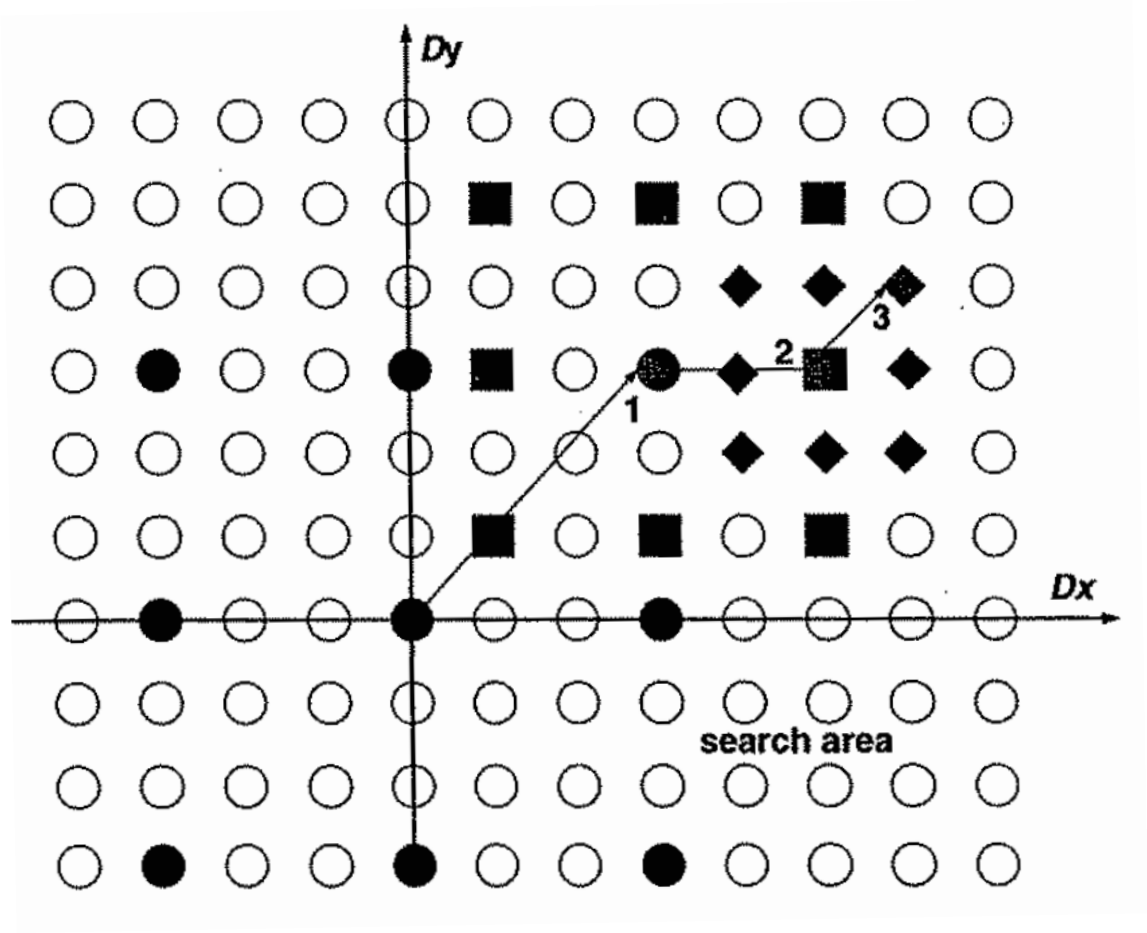


Figure 2.4: 3-Step Search Pattern

2.2 True Motion Estimation

The physical three-dimensional motion projected onto two-dimensional space is referred to as true motion. The ability to track true motion by observing changes in luminance intensity is critical to many video applications such as FRUC [20]. Different from the other motion estimation algorithms like FS, a true motion estimation algorithm should also take other measures into account like spatio-temporal consistency of the MV field around objects. This is based on two assumptions. Objects are larger than blocks so that MV field around a block should be smooth and objects have inertia, i.e. object motions are spread through time to several frames. Therefore, motions of the objects can also be tracked by analyzing previous frames.

There are several true motion estimation algorithms in the literature [11-15] that check the spatio-temporal consistency around blocks to obtain the true motion of the object containing that block. Three Dimensional Recursive Search (3DRS) [11] is one of the best implementations of these two assumptions. Instead of evaluating all possible candidate locations in a search window, 3-D recursive search algorithm uses spatial and temporal predictions to select only a few candidate vectors from the 3-D neighborhood (spatial and temporal neighbors) of the current block, thus reducing computational complexity of ME which is the most computationally expensive part of MC-FRUC and also resulting in a smooth and accurate true MV field.

There are two problems with the first assumption in 3DRS. First, because of the processing order of the blocks (starting from top-left block and ending with the bottom-right block), not all of the spatial neighboring blocks of the current block (CB) are available, e.g. the blocks to the right of the CB and the blocks that are below the CB. This problem is solved with the second assumption. Since the motion of the object continues over several frames, instead of the motion vectors of the spatial neighboring blocks that are not yet calculated the motion vectors of the corresponding temporal neighboring blocks are used.

Second, all vectors are zero or undefined at initialization. Therefore, the motion vector of the object cannot be found in any of the neighboring blocks in the first frame. This problem is solved by adding random update vectors from a pre-defined set of noise vectors, filling the MV field with not accurate but possible motion data. In [21], it is proposed to use the candidate set shown in Equation (2.4) and illustrated in Figure 2.5. Squares marked as *S* are vectors taken from spatial neighbors and square marked as *T* is the vector taken from the previous frame. CB denotes the current block.

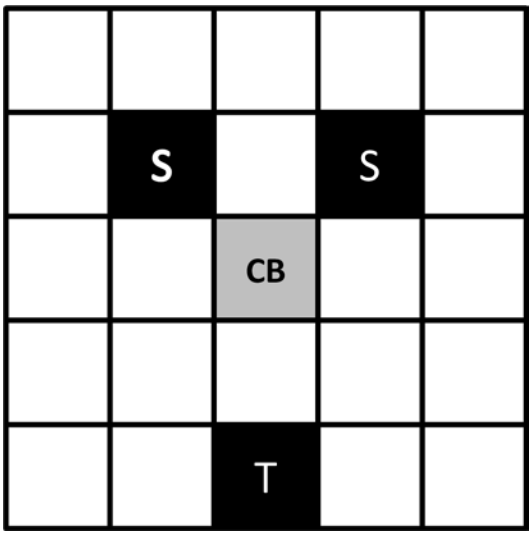


Figure 2.5: Candidate Search Locations Set for 3DRS

$$CS_{3DRS}(\vec{X},n)=\begin{Bmatrix}\vec{V}\left(\vec{X}+\begin{pmatrix}-1\\-1\end{pmatrix},n\right)+\overrightarrow{U_1}(\vec{X},n)\\ \vec{V}\left(\vec{X}+\begin{pmatrix}1\\-1\end{pmatrix},n\right)+\overrightarrow{U_2}(\vec{X},n)\\ \vec{V}\left(\vec{X}+\begin{pmatrix}0\\2\end{pmatrix},n-1\right)\end{Bmatrix} \tag{2.4}$$

where the update vectors $\overrightarrow{U_1}(\vec{X},n)$ and $\overrightarrow{U_2}(\vec{X},n)$ are randomly selected from the following update set:

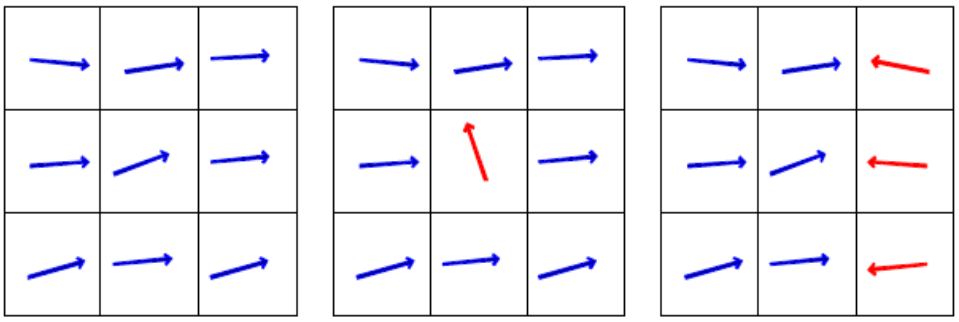
$$U_i\left(\vec{X},n\right)=\begin{Bmatrix}\vec{0}\\ \begin{pmatrix}0\\1\end{pmatrix},\begin{pmatrix}0\\-1\end{pmatrix},\begin{pmatrix}1\\0\end{pmatrix},\begin{pmatrix}-1\\0\end{pmatrix},\\ \begin{pmatrix}0\\2\end{pmatrix},\begin{pmatrix}0\\-2\end{pmatrix},\begin{pmatrix}3\\0\end{pmatrix},\begin{pmatrix}-3\\0\end{pmatrix}\end{Bmatrix} \tag{2.5}$$

2.3 Intermediate FRUC Steps

In addition to the two main FRUC steps, additional steps such as motion vector smoothing or bilateral motion estimation can be performed before MCI to improve the quality of the estimated motion vectors by refining them to obtain a smoother and more accurate MV field.

2.3.1 Motion Vector Smoothing

Motion fields are usually smooth functions except at object boundaries. However, there are cases where even true motion estimation may produce unreliable motion vectors. Therefore, outliers can occur as shown in Figure 2.6 (b). These outliers should be eliminated for FRUC applications.



(a) **(b)** **(c)**

Figure 2.6: Motion Vector Smoothing (a) Smooth region (b) Outlier MV (c) Object boundary

There are many approaches for motion vector smoothing. One of them is Vector Median Filtering (VMF) [22] which eliminates outliers while preserving boundaries between different objects.

Let, $MVF = \{mv_1, mv_2, \dots, mv_N\}$ be the set of MVs inside the smoothing window. Then the median vector mv_{median} is defined as the element in the set, which satisfies the inequality,

$$\begin{cases} mv_{median} \in MVF \\ \sum_{i=1}^{N} \|mv_{median} - mv_i\|_p \leq \sum_{i=1}^{N} \|mv_j - mv_i\|_p \end{cases} \quad , j = 1,2, \dots, N \tag{2.6}$$

where the norm $\|\cdot\|_p$ defines the metric used to convert a vector to a scalar value. For the norm operation generally the L1 norm (p = 1) is used since it has low computational complexity and it is an effective method for checking vector similarity [10]. L1 norm is defined as,

$$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |x_i| \tag{2.7}$$

where x_i is the i[th] component of the vector $\vec{x}$.

The size of the smoothing window is selected as 3x3 in practical applications. The block currently being processed is placed in the center of the window, and the 8 surrounding neighbors are used in the filtering process, making a total of 9 vectors in each window as shown in Figure 2.7.

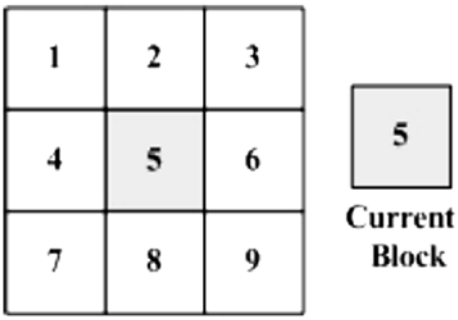


Figure 2.7: 3x3 Smoothing Window

An example application of motion vector smoothing is shown in Figure 2.8. The outliers in the boundary region cannot be processed because of the unavailability of some of the neighboring MVs.

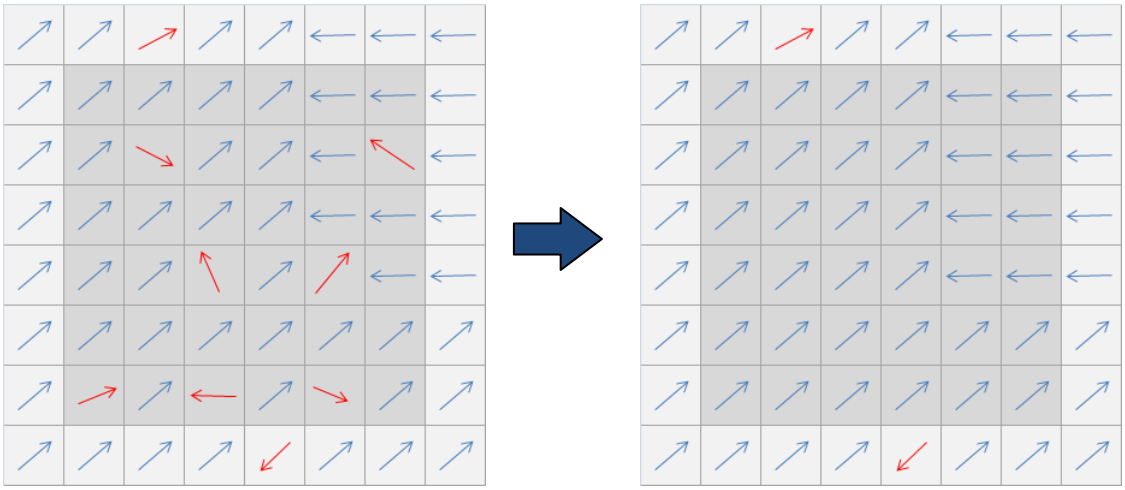


Figure 2.8: Example Application of Motion Vector Smoothing

2.3.2 Bilateral Motion Estimation

One of the potential problems with BM algorithms for FRUC is the possible hole and overlapped areas in the interpolated frames. Since a new frame is generated by interpolation between previous frame (PF) and current frame (CF) based on motion vectors (MV) and these vectors are obtained by ME which assumes that objects move along the motion trajectory, holes and overlapped areas may be produced in the interpolated frames due to no motion trajectory passing through and multiple motion trajectories passing through, respectively [23]. This degrades the quality of generated frames as shown in Figure 2.9. This problem can be solved by median filtering overlapped pixels [24], using spatial interpolation methods for holes [25], or prediction methods by analyzing MV fields for covered and uncovered regions [23][26]. However, these methods have high computational complexity and give unsatisfactory results, especially in cases of non-static backgrounds and camera motions. To overcome this problem more efficiently, Bilateral Motion Estimation (Bi-ME) methods are proposed [27]-[30], which construct a MV field from the viewpoint of the to-be-interpolated frame, and therefore do not produce any overlapped areas or holes during interpolation.

Figure 2.9: (a) Hole and Overlapping Regions (b) Frame Generated by Bilateral ME

In other ME algorithms, an NxN size block from CF, CB, is kept stationary and a match for this CB is searched inside a search window in PF. In Bi-ME, an imaginary frame is assumed to exist which will be the intermediate frame after it is interpolated, and ME is performed from the viewpoint of this frame. Therefore, the block inside the to-be-interpolated frame is kept stationary and a match for this block is tried to be found both in CF and PF at symmetric locations to each other. The trajectory connecting two symmetric blocks in CF and PF always passes through the stationary block inside the

to-be-interpolated frame. When the best match is found, the trajectory between two symmetric blocks is assigned as the MV to the block that will be interpolated. The Bi-ME process is shown in Figure 2.10.

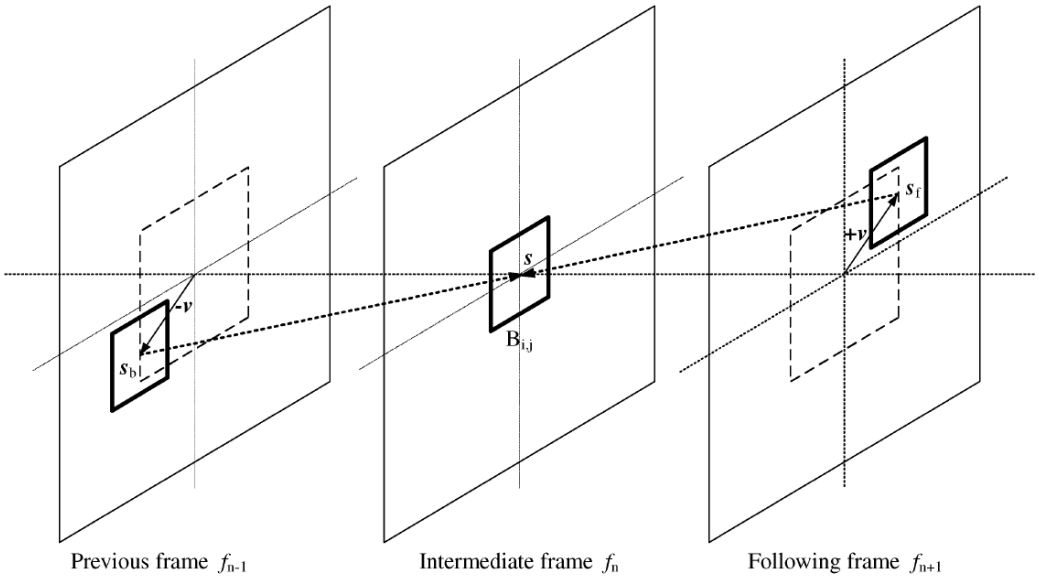


Figure 2.10: Bilateral Motion Estimation

Bi-ME, when used exclusively as the ME step, does not yield acceptable results for MC-FRUC applications due to its lack of true motion estimation capability. It is proposed in [27] that Bi-ME can be used as a refinement step to a ME algorithm as shown in Figure 2.11.

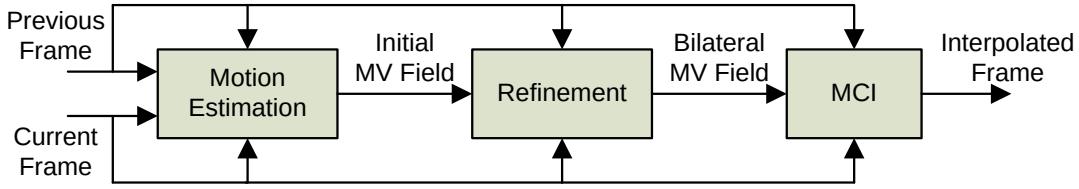


Figure 2.11: Bilateral ME as a Refinement Step

2.4 Motion Compensated Interpolation

The last step of a MC-FRUC system is the Motion Compensated Interpolation (MCI) step, which interpolates the pixel data of the intermediate frame using the motion vectors generated by the ME step between the previous and current frames. A robust MCI algorithm is as important as a robust ME algorithm. Even if the ME cannot

accurately estimate the true motion of the object like in the cases of covering and uncovering of different objects, MCI algorithm may detect these cases and be able to generate a high quality video output.

2.4.1 Motion Compensated Field Averaging

Motion Compensated Field Averaging (MC-FAVG) [1] is the most basic MCI method. MC-FAVG algorithm combines two adjacent frames linearly, with each block in the PF is shifted towards the CF according to the value of its MV, and similarly each block in the CF is shifted towards PF along its motion trajectory. The algorithm is shown in Equation (2.8)

$$F_{mca}(\vec{x}, n+\alpha) = \frac{1}{2}\left(F(\vec{x} - \alpha\vec{V}, n) + F(\vec{x} + (1-\alpha)\vec{V}, n+1)\right); \qquad 0 \leq \alpha \leq 1 \tag{2.8}$$

where $F(\vec{x}, n)$ denotes the intensity value of the pixel at location $\vec{x}$ in frame n, α denotes the up-conversion ratio (0.5 for doubling the frame rate), and $\vec{V}$ is the MV associated with that pixel.

2.4.2 Static Median Filtering

In some cases when a wrong MV is assigned to stationary objects like text areas, MC-FAVG produces blocking artifacts. This problem can be solved by Static Median Filter (SMF) algorithm [1]. In SMF, two inputs of a median filter is fed with two pixel values, one from the PF and one from the CF, both from the same location of the current pixel to be interpolated. The third input is connected to the output of the MC-FAVG algorithm. With this scheme, in cases of stationary fields, values of the two stationary pixels will be similar. This would result in the selection of one of those pixels. On the other hand, when there is a temporal discontinuity, values of the stationary pixels will be apart, therefore the MC-FAVG result will be used. The SMF algorithm is shown in Equation (2.9).

$$F_{smf}(\vec{x}, n+\alpha) = med\{F(\vec{x}, n), F(\vec{x}, n+1), F_{mca}(\vec{x}, n+\alpha)\} \tag{2.9}$$

2.4.3 Dynamic Median Filtering

Dynamic Median Filter (DMF) [1] also uses a 3-point median filter scheme. However, in DMF, two inputs of the filter is fed with motion compensated pixel values

from previous and current frames each taken from respective locations that the MV of the to-be-interpolated pixel points to. The third input is the non-motion compensated average of two pixels taken from the same location of the to-be-interpolated pixel both from CF and PF. The DMF is shown in Equation (2.10).

$$F_{dmf}(\vec{x}, n+\alpha) = med\left\{F\left(\vec{x}-\alpha\vec{V}, n\right), F\left(\vec{x}+(1-\alpha)\vec{V}, n+1\right), \frac{1}{2}(F(\vec{x}, n)+F(\vec{x}, n+1)\right\} \tag{2.10}$$

In cases where the motion vector is accurate, the compensated pixels will have about the same values, and therefore the median filter will select either of them. But if the motion vector is unreliable, then it is likely that values of the compensated pixels will be apart from each other, therefore the uncompensated input will be selected.

2.4.4 Two-Mode Interpolation

Two-Mode Interpolation (2MI) [1] algorithm aims at a relatively better interpolation at a reduced operation count. This algorithm is based on occlusion detection to have information about whether there is a covering or an uncovering situation in the frame or not. This detection is done by analyzing the MV field seeking significant discontinuities between neighboring vectors. When a discontinuity is found, it is assumed that borders of objects are reached, therefore MVs of those blocks are less reliable and MCI should be done with more caution. On the other hand, when the MV field is smooth, a simpler MCI algorithm like MC-FAVG is sufficient. For the occlusion detection, the difference between the MV values of the left and right blocks and the difference between the MV values of the top and bottom blocks are checked. If any of them is higher than a pre-defined threshold value, an occlusion is assumed to be found and the MCI is handled by DMF. Otherwise, MC-FAVG is used for that block. 2MI is shown in Equation (2.11).

$$F(\vec{x}, n+\alpha) = \begin{cases} med\left\{F\left(\vec{x}-\alpha\vec{V}, n\right), F\left(\vec{x}+(1-\alpha)\vec{V}, n+1\right), \frac{1}{2}(F(\vec{x}, n)+F(\vec{x}, n+1)\right\}, & occlusion \\ \frac{1}{2}\left(F\left(\vec{x}-\alpha\vec{V}, n\right)+F(1-\alpha)\vec{V}, n+1\right)\right) & , otherwise \end{cases} \tag{2.11}$$

This adaptation yields a generally improved output compared to each method individually. The operation count is reduced roughly 30% compared with that of the dynamic median filter, since dynamic median filtering is needed for a relative small portion of pixels in the image (on average less than %10). [1]

2.4.5 Overlapped Block Motion Compensation

The block based ME uses the assumption that all the pixels in a block have the same motion as there exists a single motion vector for each block. However, different parts of objects that move in different directions can be in the same block or MV field generated by the ME step may not represent the correct motion of the objects due to ME errors. In these cases, conventional block based interpolation may produce blocking artifacts or block boundary discontinuities that reduce the quality of the video both in subjective and objective metrics.

Overlapped Block Motion Compensation [31] is developed in order to avoid these blocking artifacts and increase the quality of the resulting frame in MC-FRUC. It is also used in video compression standards such as H.263 [32]. The main idea of OBMC is based on determining the motion of each pixel in a block by considering the motion vector of the block itself as well as the motion vectors of its neighboring blocks.

A simple OBMC technique is implemented in [27]. It employs OBMC during the interpolation stage by enlarging every *NxN* block in the to-be-interpolated frame to *(N+2w) x (N+2w)* block which form overlapped areas of width *w* in every block as shown in Figure 2.12. The purpose of this operation is having a smooth transition between adjacent blocks. The pixels at the corners of an *NxN* block are located in the overlapped area of the 4 neighboring blocks. The intensities of these pixels are calculated by averaging the intensity values generated by the motion vectors of each respective block. The intensities of the pixels that are located at the side boundaries of the interpolated block are calculated by averaging the intensity values generated by the motion vectors of the interpolated block and the adjacent block. The remaining interpolation is done by only using the motion vector of the to-be-interpolated block.

For example, in Figure 2.12, OBMC is not applied to the pixels in *R1* regions as these pixels belong to a single block. The pixels that are located in *R2* regions should be interpolated by taking motion vectors of both adjacent blocks into account, as these pixels belong to both blocks. The pixels in *R3* region are in the overlapped area of 4

neighboring blocks, therefore the interpolations of these pixels are performed by using 4 different motion vectors.

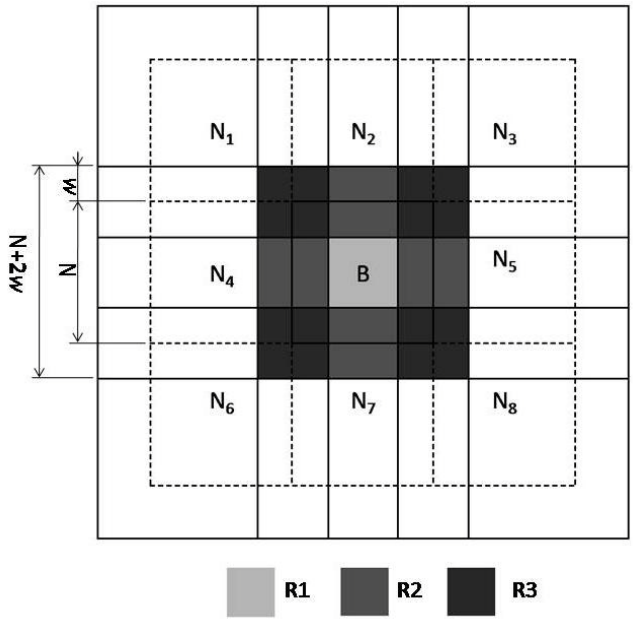


Figure 2.12: Overlapping Regions in OBMC

The interpolation of the block *B* is defined in Equations (2.12), (2.13) and (2.14) where the neighboring blocks are *Ni= 1, 2… 8*, $\vec{V}(\vec{x})$ refers to the motion vector of the block *B* at position $\vec{x}$ and $F_{mca}\,(\vec{x}, \vec{V}(B))$ denote the motion compensated field averaging for pixel at $\vec{x}$ using motion vector *V* of block *B*.

1. For *R1*:

$$F(\vec{x}) = F_{mca}\,(\vec{x} \in R1, \vec{V}(B)) \tag{2.12}$$

2. For *R2*:

$$F(\vec{x}) = \frac{1}{2}\left\{F_{mca}\,\left(\vec{x} \in R2, \vec{V}(B)\right) + F_{mca}\,(\vec{x} \in R2, \vec{V}(N_i))\right\}$$

where $N_i \in \{N_2, N_4, N_5, N_7\}$. (2.13)

3. For *R3*:

$$F(\vec{x}) = \frac{1}{4}\left\{F_{mca}\,\left(\vec{x} \in R3, \vec{V}(B)\right) + S_k\right\}, k = 1,2,3,4$$

(2.14)

where S_k is the sum of the MC-FAVG results for the neighboring blocks overlapped with *B* in *R3* and defined by:

$$S_1 = F_{mca}\left(\vec{x}, \vec{V}(N_1)\right) + F_{mca}\left(\vec{x}, \vec{V}(N_2)\right) + F_{mca}\left(\vec{x}, \vec{V}(N_4)\right)$$
$$S_2 = F_{mca}\left(\vec{x}, \vec{V}(N_2)\right) + F_{mca}\left(\vec{x}, \vec{V}(N_3)\right) + F_{mca}\left(\vec{x}, \vec{V}(N_5)\right)$$
$$S_3 = F_{mca}\left(\vec{x}, \vec{V}(N_4)\right) + F_{mca}\left(\vec{x}, \vec{V}(N_6)\right) + F_{mca}\left(\vec{x}, \vec{V}(N_7)\right)$$
$$S_4 = F_{mca}\left(\vec{x}, \vec{V}(N_5)\right) + F_{mca}\left(\vec{x}, \vec{V}(N_7)\right) + F_{mca}\left(\vec{x}, \vec{V}(N_8)\right)$$

(2.15)

The quality of the generated frame can further be improved by giving weights to pixels of neighboring blocks according to their spatial distance from the current block [28], favoring the CB's pixels inside that block, giving 50% weight to both blocks at the edge of two blocks, and decreasing the weight while moving away from the CB. The quality of the generated frame can also be improved by assigning weights to the neighboring blocks according to the reliability of their motion vectors, i.e. the smoothness of the MV field around the CB [29].

2.5 Evaluation Methods and Metrics

In this thesis, the performances of FRUC algorithms are evaluated as follows. Every even numbered frame is omitted from the sequence and ME is employed between odd frames. Then, MCI step is applied using these MVs to re-synthesize the even numbered frames as shown in Figure 2.13. After all even numbered frames are generated, the original even numbered frames and interpolated even numbered frames are compared as shown in Figure 2.14. The comparison is done using Mean Squared Error (MSE) metric by calculating the differences of each pixel at the same locations in the original and interpolated frames and summing the squares of these values as shown in Equation (2.16). After all MSEs for all even numbered frames are found, the corresponding Peak Signal-to-Noise (PSNR) ratios are found as shown in Equation (2.17).

$$MSE = \frac{1}{NM}\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}(I(i,j) - O(i,j))^2$$

(2.16)

where N and M denote the image height and width respectively, I is the interpolated frame and O is the original frame.

$$PSNR = 10.log_{10}\left(\frac{MAX^2}{MSE}\right) = 20.log_{10}\left(\frac{MAX}{\sqrt{MSE}}\right) \tag{2.17}$$

where MAX is the maximum possible error between two pixels. If pixel intensities are represented by 8 bits, then MAX is 255.

PSNR is a widely used evaluation metric for the quality of video sequences. PSNR is accepted as a good objective measure of quality. However, the perceived quality of the video is not always directly related to its objective quality. A viewer can identify a sequence as a low quality sequence because of its unpleasing artifacts around object edges even though every other pixel would have been interpolated perfectly thus having a very high PSNR value. On the other hand, a video can have a low PSNR value like in a case of blurring but that blurring could be unnoticeable by the viewer especially in scenes where objects move in high velocities. Therefore, when evaluating the performances of FRUC algorithms, subjective quality assessments should also be made along with objective quality assessments.

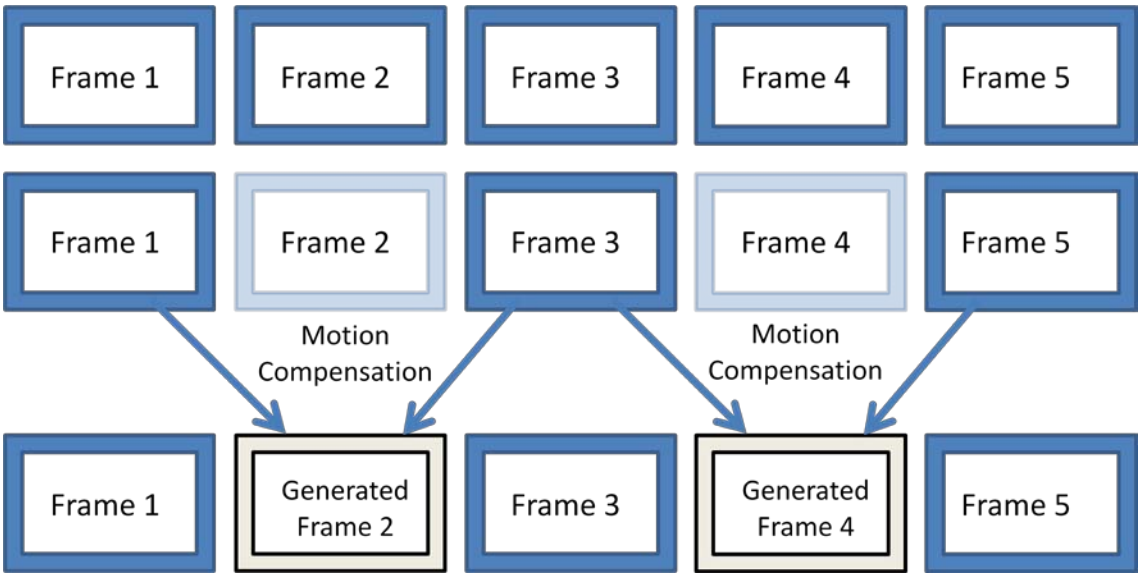


Figure 2.13: Generation of Even Numbered Frames

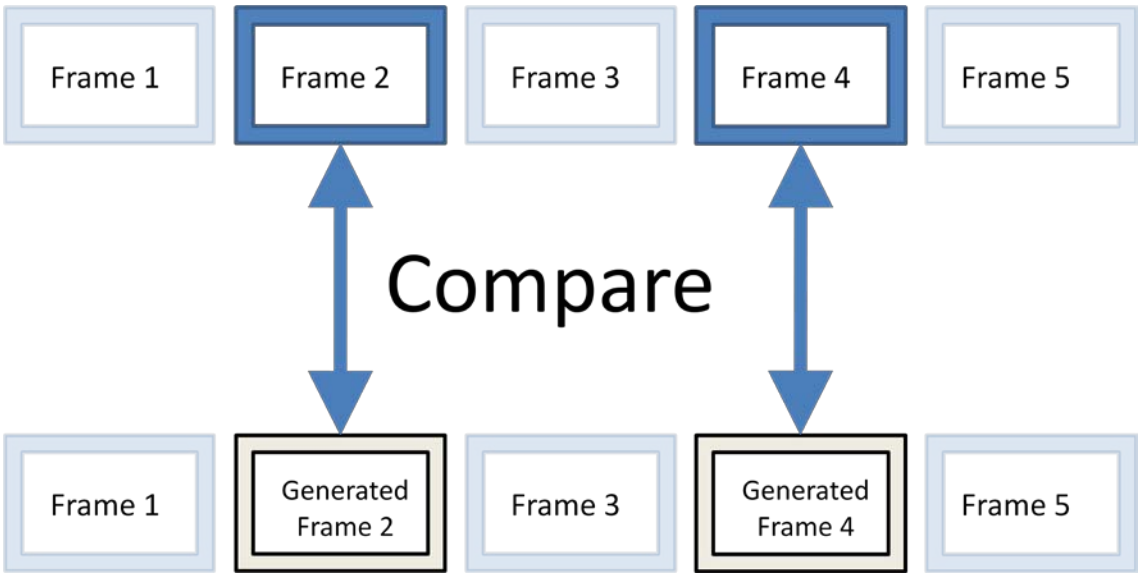


Figure 2.14: Comparison of Even Numbered Frames

Chapter 3

ADAPTIVE TRUE MOTION ESTIMATION ALGORITHM AND MOTION COMPENSATED FRAME RATE UP-CONVERSION SOFTWARE

3.1 Adaptive True Motion Estimation Algorithm

In this thesis, Adaptive True Motion Estimation Algorithm (ATME) is developed based on 3DRS. It is observed by analyzing the MV fields generated by 3DRS that the two main assumptions of recursive true motion algorithms are indeed correct, the objects are bigger than blocks and motions of the objects are continuous. Therefore, the candidate locations that will be evaluated by 3DRS for the current block will be close in value or even the same in many cases. In addition, multiple passes of 3DRS are observed to improve the smoothness of the MV field at each pass hence improving visual quality. The probability of being selected again as the best matching candidate for a block is quite high for a MV which was selected as the best matching candidate for that block in the first pass of the algorithm. Based on these facts, in order to reduce the computation cost of 3DRS, ATME algorithm avoids the evaluations of the same and similar MV candidates by applying computational complexity reduction and redundancy removal techniques. In addition, when the SAD value of the best match is decided not sufficient to be selected, ATME algorithm evaluates additional locations to improve the quality of the MV field. Using these techniques, it obtains similar quality results by less number of computations or better quality results by similar number of computations compared to 3DRS.

To obtain an optimal candidate set for the proposed ATME algorithm, a multi-objective genetic algorithm [16] is applied to all of the candidate locations, located (±5,±5) blocks around the current block. Populations in this genetic algorithm have 25 individuals, each representing a candidate set containing a minimum of one search

location to a maximum of 20 search locations. Objectives of this test are defined as maximizing the PSNR of the up-converted video sequences using the candidate sets of best-individuals in the population, and at the same time minimizing the total number of SAD calculations, which converges to the optimal set of candidates producing high quality results with small amount of work. This algorithm is run on a set of 10 video sequences[1] having various spatial resolutions from QCIF to HD for 100 generations, and the candidate sets which are on the pareto-front of the resulting population are noted down. It is observed that neighboring blocks which are closer to the current block are better candidates, whereas in cases where candidate sets contain small number of search locations, convergence is obtained faster by selecting candidates from opposite directions of the current block, as proposed in [33].

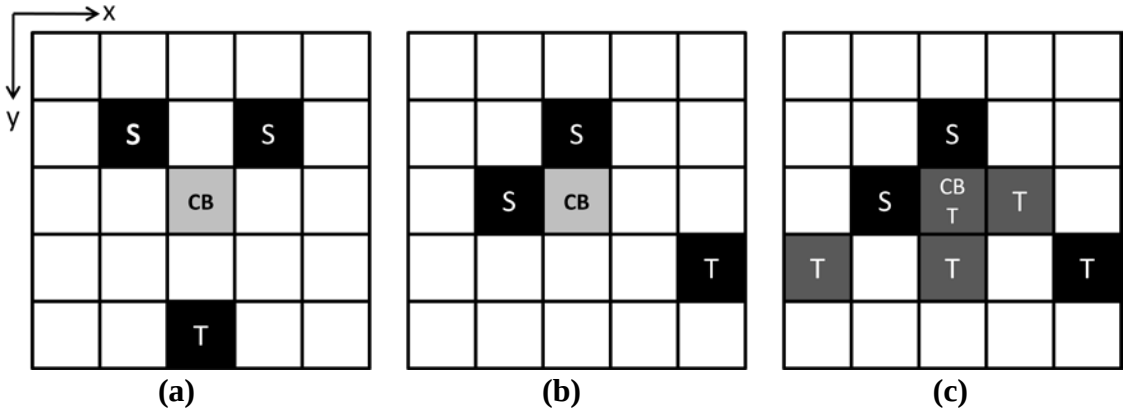


Figure 3.1: Candidate Vector Sets (a) 3DRS candidate set proposed in [21], (b) ATME minimal candidate set, (c) ATME extended candidate set shown in gray. The extended candidate set also contains no-motion vector, not shown in the figure.

The ATME algorithm uses two different sets of search locations which are applied adaptively based on several run-time checks. The minimal search location set consists of a small number of search locations to be used in the first two steps of the algorithm, and the extended search location set consists of more locations including the $\vec{0}$ vector which represents zero motion, to be used in the third step when the smaller set does produce sufficient results. The minimal and extended search location sets, proposed in this thesis based on the multi-objective genetic algorithm optimization, are shown in

[1] The video sequences used for this experiment are: Foreman(QCIF), Flower(SIF), Football(SIF), Mobile(CIF), CrowdRun(720p), NewMobCal(720p), ParkRun(720p), SthlmPan(720p), InToTree(720p), OldTownCross(720p).

Equations (3.1) and (3.2), and Figure 3.1(b) and Figure 3.1(c), respectively. The zero motion vector $\vec{0}$ is not shown in Figure 3.1(c).

$$CS_{min}(\vec{X}, n) = \begin{Bmatrix} \vec{V}\left(\vec{X} + \begin{pmatrix} -1 \\ 0 \end{pmatrix}, n\right), \\ \vec{V}\left(\vec{X} + \begin{pmatrix} 0 \\ -1 \end{pmatrix}, n\right), \\ \vec{V}\left(\vec{X} + \begin{pmatrix} 2 \\ 1 \end{pmatrix}, n-1\right) \end{Bmatrix} \tag{3.1}$$

$$CS_{ext}(\vec{X}, n) = \begin{Bmatrix} \vec{0} \\ \vec{V}(\vec{X}, n-1), \\ \vec{V}\left(\vec{X} + \begin{pmatrix} 1 \\ 0 \end{pmatrix}, n-1\right), \\ \vec{V}\left(\vec{X} + \begin{pmatrix} 0 \\ 1 \end{pmatrix}, n-1\right), \\ \vec{V}\left(\vec{X} + \begin{pmatrix} -2 \\ 1 \end{pmatrix}, n-1\right) \end{Bmatrix} \tag{3.2}$$

```
for each search location Lm⃗ in minimal set CSmin
          candidates_min[0 to Nm] = MV of the block at (B⃗ + Lm⃗)
     if all L1 Norms between candidates <= Vth
          vector0 = median of all candidates
          vector1 = vector0 + random update vector
          calculate SADs for vector0 and vector1
          assign MV producing bestSAD to block B
     else
          add random update vector to last candidate_min
          calculate SADs between all candidates_min and B
          if bestSAD > SADth
               for each search location Le⃗ in extended set CSext
                    candiates_ext[0 to Ne] = MV of the block at (B⃗ + Le⃗)
               add random update vector to last candidate_ext
               calculate SADs between all candidates_ext and B
          assign MV producing bestSAD to block B
```

Table 3.1: Pseudo-code for ATME

The pseudo-code for ATME algorithm is given in Table 3.1. The ATME algorithm first checks whether the vectors in the minimal search location set are consistent with the motion of the current block, i.e. belonging to the same object and representing similar motions. This is done by taking the L1 Norm of these 3 vectors. If the norm is below a predefined threshold value (V_{th}), this means that the motion associated with surrounding blocks is likely to be same as the motion of the current block. Therefore, the median of this minimal set is assigned to the current block without further SAD calculation. However, because of the recursive behavior of vector

selection, without an additional update vector, this scheme may converge to an invariable vector field. Therefore, the median vector and its random update vector added version are evaluated based on the SAD criterion, and the vector with the minimum SAD is selected and assigned to the current block. This step reduces the number of SAD calculations in a spatio-temporally smooth video sequence without a significant PSNR loss and at the same time smoothes the vector field because of the median operation, which is used as a separate step in many FRUC algorithms. As a result of this motion vector field smoothing at a reduced cost, increased PSNR values are observed in some cases, while none of the cases resulted in significant PSNR losses.

If the L1 Norm of the minimal search location set is not below the threshold V_{th}, this means that there are inconsistent MVs around the current block, and therefore all 3 MVs in the minimal candidate set are searched individually. If the minimum SAD resulting from this step is below a predetermined SAD threshold, SAD_{th}, then the motion represented by the minimum SAD producing MV is assigned to the current block. However, if the minimum SAD obtained by evaluating the minimal search locations set is not below SAD_{th}, then the motion vector representing the motion of the current block is probably not available in that candidate set, and therefore additional search locations should be evaluated. In this case, extended search locations set consisting of 5 new search locations is introduced and SAD calculation is done for the MVs of the neighboring blocks at these new search locations. If the minimum of these SAD values are smaller than the result of the minimal search location set, then that motion vector is assigned to the current block, otherwise the result of the minimal set is used.

Since the recursive true ME algorithms depend on the evaluation of some MVs at spatial and temporal neighboring locations, convergence of the MV field can be obtained by applying the true ME algorithm to the same frame more than one time. This multiple pass technique increases the quality of the FRUC by generating a smoother MV field, i.e. representing the true motion of the objects more correctly [34]. After each pass of ME, some of the incorrect vectors will converge to better vectors, whereas most of the time, they will keep their values from the previous pass. Therefore, if the SAD values of the vectors are kept between each pass of the algorithm, instead of redundantly calculating the same SAD value, the SAD value from the previous iteration can be used. This redundancy removal technique is used in ATME algorithm. It resulted in significant reduction in computation amount while producing exactly same results.

3.2 Motion Compensated Frame Rate Up-Conversion Software

There was a need for a robust, fast, flexible and easily modifiable software for the implementation and testing of FRUC algorithms. Therefore, in this thesis, a FRUC software environment is implemented using C. The backbone of the software consists of a loop which reads image data from YUV files stored locally on the hard disk. For memory efficiency, instead of reading all frames of the video sequence into memory, one frame at a time is read and stored in two static arrays, one for the previous frame (PF) and one for the current frame (CF). In addition, instead of reading two frames in one iteration, the pointer to the CF at the previous iteration is set to be the PF at the current iteration and the new frame is read to the location of the PF at the previous iteration and its pointer is set to be the CF at the current iteration. This double buffering technique significantly increases the performance of the software.

Inside the main loop, before any calculation, the PF is resized by mirroring pixel data at all of the four edges to provide valid data for MVs pointing out of frame bounds of the image. The resize amount is set by a user defined parameter. Figure 3.2(a) shows an example resize scheme where an 8x6 pixel image is resized with resize amount set to 3. The numbers inside cells denote the pixel positions in the original frame. Figure 3.2(b) shows the first frame in the ForemanCIF sequence with resize amount set to 32.

19	18	17	17	18	19	20	21	22	23	24	24	23	22
11	10	9	9	10	11	12	13	14	15	16	16	15	14
3	2	1	1	2	3	4	5	6	7	8	8	7	6
3	2	1	1	2	3	4	5	6	7	8	8	7	6
11	10	9	9	10	11	12	13	14	15	16	16	15	14
19	18	17	17	18	19	20	21	22	23	24	24	23	22
27	26	25	25	26	27	28	29	30	31	32	32	31	30
35	34	33	33	34	35	36	37	38	39	40	40	39	38
43	42	41	41	42	43	44	45	46	47	48	48	47	46
43	42	41	41	42	43	44	45	46	47	48	48	47	46
35	34	33	33	34	35	36	37	38	39	40	40	39	38
27	26	25	25	26	27	28	29	30	31	32	32	31	30

(a)

(b)

Figure 3.2: Resizing of Frames (a) 8x6 frame with resize amount = 3 (b) First frame of ForemanCIF sequence with resize amount = 32

Another important parameter in the software is, *replace* switch. It is defined to control the main behavior of the software whether to replace all even numbered frames for testing purposes or to perform a full FRUC to double the frame rate. After all pointers are set, ME, MCI and other steps if scheduled are applied to the image data. All of the ME and MCI operations are defined by individual C functions passing relevant data from one another. The functions that will be used are selected by user-defined parameters. This is a very efficient and flexible implementation as the user could easily change the order of operations or define additional operations without actually having to worry about the underlying data transfer as long as same set of data structures are used.

Instead of hard-coding all user defined parameters before each run of the software, a dynamic text parser is implemented so that the software can be run with many different configurations without rebuilding the whole project again. This parser reads all of the parameters from a configuration file which can be manually edited by a regular text editor. New parameters can easily be added by adding a few lines to the parser code. The parameters inside the configuration file includes, video frame size (QCIF, CIF, SIF, 4CIF, 720p HD, and 1080p HD), frame count, block size, frame resize amount, search window size, refinement window size, all of the input and output file names, operational switches like replace or early termination, the ME and MCI algorithms to be used, number of ME passes for recursive algorithms. In addition, parameters for individual algorithms like search candidate locations for 3DRS or ATME are defined in this configuration file. The screen shot of an example configuration file is shown in Figure 3.3.

During ME, MVs for each block are kept in a dynamic array for recursive usage at next ME iteration and they are also written into a text file for external use like MV visualization. During MCI, each pixel is interpolated using the MV of the block it belongs to and the resulting intensity value is written as a pixel value of the intermediate frame.

After the completion of the main loop, i.e. all frames are processed, and the output video is generated, the comparison begins. If the *replace* switch is set to true, the software compares the original even numbered frames with the interpolated even numbered frames by calculating MSE and then PSNR values. The PSNR value and the total number of calls to SADCalculate function, *SAD Count*, are written to a log file. If the *replace* switch is set to false, only *SAD Count* is written to log file.

This software is a robust and flexible environment for implementing and testing FRUC algorithms. It is used by two senior graduation projects [35-36] which developed and implemented their own ME and MCI algorithms using this software.

```
//INPUT_VIDEO     ParkRun400.yuv    //500 frames
INPUT_VIDEO      NewMobCal400.yuv   //500 frames
//INPUT_VIDEO     football_sif.y4m.yuv //124 frames
//INPUT_VIDEO     mobile_cif.yuv    //300 frames
//INPUT_VIDEO      ParkJoy_720p_50_400.yuv
//INPUT_VIDEO        ParkJoy_1080p_50_400.yuv
//INPUT_VIDEO           irobot.yuv
//INPUT_VIDEO           spiderman.yuv
//INPUT_VIDEO           gladiator.yuv

//input parameters
IMG_SIZE        720p
FRAMECOUNT      4
BLOCKSIZE       16

FSWIN           36
BSWIN           5

//flags
REPLACE         0       //use to replace even numbered frames with interpolated ones
FIRST_FS        0       //apply full search to first frame pair
ALL_FS          0       //apply full search to entire sequence
BI_FS           0       //apply bilateral search
UPDATE          1       //enable updating mechanism for 3DRS
OCCLUSION_TH    2       //occlusion detection threshold
PASS_COUNT      1       //number of 3DRS passes
EARLY_TERMINATE 0       //enable early termination for no-motion vectors
NEW3DRS         1       //apply new 3DRS

UC_ALGO         FAVG    //select interpolation algorithm: FAVG,DYNM,TWOM,BI_FAVG,OBMC,MC


//filenames
OUTPUT_VIDEO    output.yuv
MVs_FILE_BI     bi_MVs.txt
MVs_FILE        MVs.txt
MV_X_FILE       MV_X.txt
MV_Y_FILE       MV_Y.txt
RESULTS         results.txt

SAD_THRESHOLD 65536
VECTOR_THRESHOLD 2

//3DRS parameters
CANDIDATE_COUNT 3
EXT_CANDIDATE_COUNT 4
//SEARCH_LOCS       -1 0   2 2   0 -1   1 -2   2 -1   -2 -2   0 0   0 -2   3 -1
//SEARCH_LOCS       -1 0   0 -1   2 2   -1 0 //UPDATED
SEARCH_LOCS         -1 0    0 -1  2 1
//SEARCH_LOCS           0 2    -1 -1  1 -1  //haan min
//SEARCH_LOCS           -1 0    0 -1  -2 1  0 1   0 0  2 1  1 0
//EXT_SEARCH_LOCS    0 0   2 -1  -2 -2   1 -2 -3 3  0 -2
EXT_SEARCH_LOCS      -2 1  0 1   0 0  1 0

//3DRS weight definitions
SPATIAL_COEF            1.0
TEMPORAL_COEF           1.045
```

Figure 3.3: Configuration File

In the current version of the software the following algorithms have been implemented.

ME Algorithms:

Full Search

Search window size is parameterized.

3DRS

The number of search candidates, their locations and update location are parameterized. The user can also select whether to fill the initial MV field by random update vectors or apply a Full Search between the first two frames.

Bi-ME

In a senior graduation project [35], we collaboratively proposed a new adaptive bilateral motion estimation algorithm to be used as a refinement step to improve the quality of the MVs found by true motion estimation algorithms. By employing a spiral search pattern [37] and by adaptively assigning weight coefficients to candidate search locations, the proposed algorithm refines the motion vector field between successive frames which results in a better interpolation of the intermediate frame. As a result of this search scheme, by favoring the candidate search locations near the center where the initial MVs point to, true motion property of the motion vector field is conserved. In this software, Bi-ME can be both used as a standalone ME step or as a refinement step after a true ME algorithm. Both regular FS and spiral search patterns are implemented. The Bilateral Search Window size and the threshold values used for adaptivity are parameterized.

ATME

The proposed Adaptive True Motion Algorithm is implemented. The vector threshold and SAD threshold values are parameterized. In addition, minimal set and extended set search location counts and their locations are configurable.

MCI Algorithms:

Motion Compensated Field Averaging

MC-FAVG is implemented as in Equation (2.8). When 3DRS is selected as the ME algorithm and the *update* switch is set to false, all of the MVs for the first frame will be set to zero and they will not be updated in the following frames. Therefore, in this case, MC-FAVG will function as non-motion compensated field averaging, i.e. linear interpolation.

Static Median Filter

SMF is implemented as in Equation (2.9).

Dynamic Median Filter

DMF is implemented as in Equation (2.10).

Two-Mode Interpolation

2MI is implemented as in Equation (2.11). An occlusion detection function checks whether the difference between MVs of surrounding blocks are greater than a parameterized occlusion threshold value. If occlusion is detected then DMF is called, else MC-FAVG is called.

OBMC

Basic OBMC and sinusoidal OBMC algorithms are implemented with parameterized window overlap amounts. In addition, weighted coefficient OBMC algorithm (WC-OBMC) which is developed in collaboration with a senior graduation project [36] is implemented. This algorithm assigns weights to motion vectors of neighboring blocks. This results in higher quality video output than the other two OBMC algorithms.

Utilities:

The video sequences used for evaluating all of these algorithms are taken from video quality expert ftp sites such as university archives and video quality experts group [38]. However, especially the HD video sequences are distributed in several different color spaces and formats (AVI, YUV2, ABEKAS), some of them have leading and trailing empty frames, and some of them are divided into image files which contain only one frame. Therefore, using MATLAB and C, these video sequences are all processed and converted to 4:0:0 and 4:2:2 YUV formats.

In addition, several utilities are developed using MATLAB. One of them, *playyuv*, using Image Processing Toolbox, can read many different YUV formats, convert them back to RGB, which the computer screens can display, and open them inside a media player interface as a playable video. Another utility is *plotMV*, which can parse the MV file generated by the FRUC software, generate a block grid, and plot each MV according to their direction and magnitude on this grid as shown in Figure 3.4. It then generates images for every frame pair showing the flow of MVs, and combines them to a playable video. This motion vector visualization tool is useful for testing ME algorithms, as erroneous MVs can be easily seen when they are visualized. The performances of different ME algorithms can also be compared by analyzing the flow of MVs from one frame pair to another.

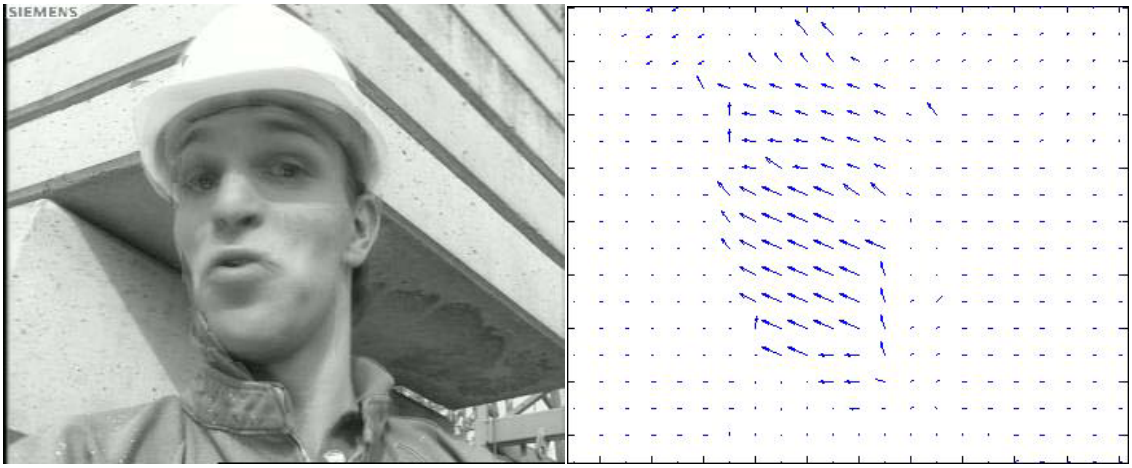


Figure 3.4: Motion Vector Visualization

3.3 Performance Results

Several video sequences with different resolutions are used for evaluating the performance of the ATME algorithm. One 176x144 pixel resolution (QCIF) video sequence, one 352x288 pixel resolution (CIF) video sequence, one 352x240 pixel resolution (SIF) video sequence, five 1280x720 pixel resolution (720p) video sequences and three 1920x1080 pixel resolution (1080p) video sequences are used. All video sequences are composed of 8-bit luminance (Y) data.

First 100 frames of each video sequence are used, therefore, 49 even numbered frames are synthesized by applying ME and MCI algorithms to the odd numbered frames, and the 100^{th} frame is taken from the original video sequence. For ME, 16x16 pixel block size is used. For the last 8 pixels of 1080p video sequences, which do not fit into the 16x16 pixel block grid, non-motion compensated frame interpolation, i.e. linear interpolation, is used. For all other cases, Motion Compensated Field Averaging is used as it is the most basic MCI method using motion estimation. The random update vector selections are done by using a 2^{31}-1 pseudo-random number sequence.

SAD calculation is the most computationally demanding part of ME algorithms. In order to calculate the SAD value for one search location, three arithmetic operations (one subtraction, one absolute value calculation and one addition) have to be performed for each pixel in a block. Therefore, the number of SAD calculations is a good metric for determining the computational complexity of a ME algorithm.

The number of SAD calculations done and the resulting PSNR value for different video sequences processed by the original 3DRS algorithm (3 candidates with 2 update vectors added) [21], 3DRS algorithm using minimal search location set (3 candidates with one update vector added), 3DRS algorithm using all search locations in both minimal and extended set including $\vec{0}$ (8 candidates with 2 update vectors added), and Full Search (FS) algorithm are shown in Tables 3.2 and 3.3. Search window size used for FS is (±64,±64) pixels for 720p and 1080p sequences, and (±32,±32) pixels for the other sequences. Non-motion-compensated pixel averaging results are given as reference. Since only the re-synthesized frames are compared with the original frames, the PSNR and SAD count values are calculated for 49 frames.

As it can be seen from Tables 3.2 and 3.3, minimal candidate set performs better than the original candidate set with the same number of SAD calculations and full set

gives higher PSNR results compared to other two sets with the cost of doing more SAD calculations in a single pass. In addition, multiple passes of each set clearly improves the FRUC results. However, generally two or three passes produce highest improvements, while the benefit of multi passes diminishes after more than three passes.

	3 Candidate Sets (3DRS Original and Minimal Sets)					**8 Candidate Set (3DRS Full Set)**			**FS**
No. of Passes	**1 Pass**	**2 Pass**	**3 Pass**	**4 Pass**	**5 Pass**	**1 Pass**	**2 Pass**	**3 Pass**	**N/A**
QCIF	0.01	0.03	0.04	0.06	0.07	0.04	0.08	0.12	19.87
CIF	0.06	0.12	0.17	0.23	0.29	0.15	0.31	0.46	79.48
SIF	0.05	0.10	0.14	0.19	0.24	0.13	0.26	0.39	66.23
720p	0.52	1.05	1.58	2.11	2.64	1.38	2.79	4.20	2890
1080p	1.16	2.34	3.52	4.70	5.89	3.09	6.24	9.39	6455

Table 3.2: Number of 10^6 SAD Calculations Done by ME Algorithms

In the first stage of the ATME algorithm, an adaptive decision is made based on whether L1 Norms of candidate MVs are above or below a predetermined threshold value, V_{th}. Since MVs have 1 pixel resolution, the V_{th} metric is defined in pixels. In order to determine the threshold value, 5 different values for V_{th} (0, 1, 2, 3, 4 pixels) are tested using only the first stage of the ATME algorithm on 4 different video sequences.[2] SAD Count value is normalized by $10*\log_{10}$ to be comparable to PSNR. Figure 3.5 shows PSNR/SAD Count efficiency versus V_{th}. The average PSNR/SAD Count efficiency versus V_{th}, based on the results from Figure 3.5, is shown in Figure 3.6. As it can be seen from these Figures, the maximum efficiency is obtained when V_{th} is 2 pixels.

[2] The sequences used in this experiment are: ParkJoy(720p), NewMobCal(720p), Foreman(CIF), SthlmPan(720p).

	3DRS Original Set			3DRS Minimal Set					3DRS Full Set			FS	Ref
No. of Passes	**1 Pass**	**2 Pass**	**3 Pass**	**1 Pass**	**2 Pass**	**3 Pass**	**4 Pass**	**5 Pass**	**1 Pass**	**2 Pass**	**3 Pass**	**N/A**	**N/A**
ForemanQCIF	32.29	32.79	33.17	33.09	33.50	33.82	33.62	33.76	33.75	34.27	34.51	32.70	32.36
ForemanCIF	30.50	31.28	31.61	31.92	32.44	32.61	32.56	32.60	32.02	32.88	33.08	31.62	29.86
FootballSIF	20.35	20.73	20.81	20.63	20.89	21.02	21.14	21.10	21.16	21.48	21.65	21.32	19.89
ParkJoy720p	22.58	24.31	24.80	24.23	25.81	25.86	26.08	26.09	25.11	25.93	26.21	25.63	20.11
NewMobCal720p	31.84	32.62	33.01	33.70	34.08	34.06	34.09	34.07	33.69	34.11	34.11	32.58	29.76
SthlmPan720p	33.11	33.96	34.22	33.98	34.83	34.90	34.89	34.89	34.10	35.03	35.06	30.40	23.96
InToTree720p	34.71	34.97	35.11	35.60	35.78	35.79	35.82	35.81	35.82	36.02	36.03	31.16	31.87
CrowdRun720p	25.75	26.26	26.43	26.94	27.26	27.30	27.30	27.31	27.41	28.01	28.18	26.43	24.51
ParkJoy1080p	23.32	24.53	25.08	24.13	25.26	26.01	26.16	26.22	24.70	25.63	26.02	25.39	20.15
InToTree1080p	33.92	34.11	34.17	34.40	34.51	34.51	34.51	34.51	34.50	34.61	34.62	31.52	30.97
CrowdRun1080p	26.32	26.98	27.21	27.19	27.75	27.87	27.89	27.91	27.64	28.31	28.50	26.33	24.24

Table 3.3: Comparison of Modified 3DRS Algorithms Using Optimized Sets of Candidate Locations along with Full Search and Non-Motion Compensated Interpolation Results
Table cells show PSNR values in dB.

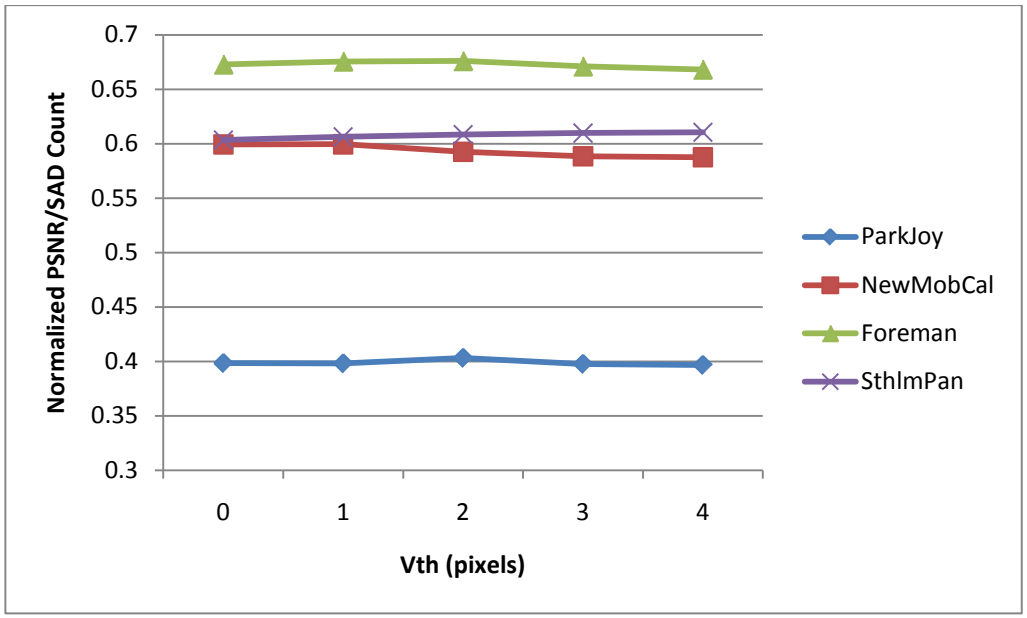


Figure 3.5: PSNR/SAD Count for Vector Threshold Selection

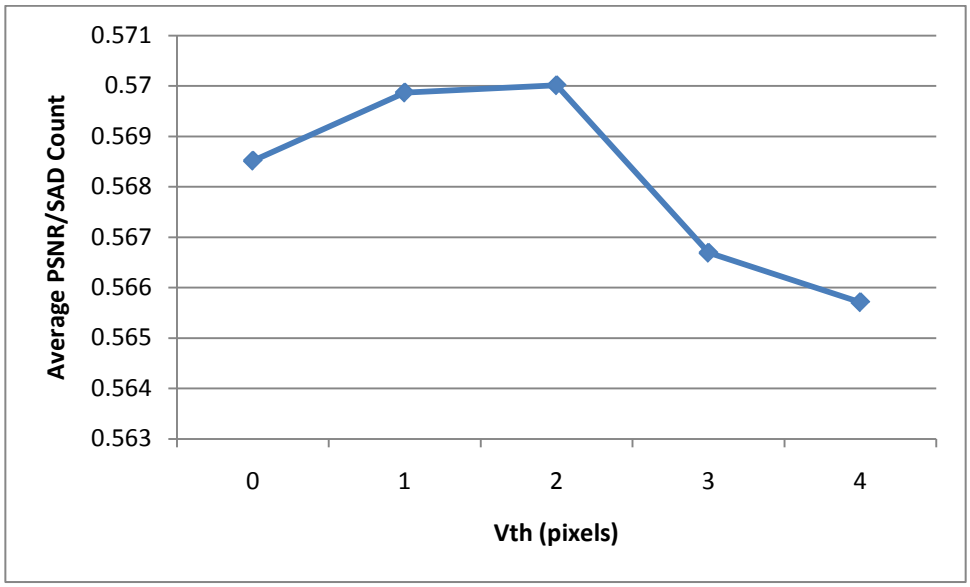


Figure 3.6: Average PSNR/SAD Count for Vector Threshold Selection

Tables 3.4 and 3.5 present the impact of the redundancy removal and computational complexity reduction techniques used in ATME algorithm. Table 3.4 shows the impact of the redundancy removal and computational complexity reduction in only the first stage of ATME algorithm. In this test, extended set of candidates and the redundancy removal technique for multiple passes of the algorithm are not used. Two different candidate sets are used, the minimal set which contains 3 candidates and the full set which contains the minimal set and the extended set including zero-motion vector $\vec{0}$. As it can be seen from Table 3.4, when V_{th} is 0, ATME algorithm produces

exactly the same PSNR results compared to reference algorithms in which redundancy removal technique is not used, and the number of SAD calculations is reduced up to 20% and 38% for minimal and full sets respectively for a single pass. When 3 passes are done, the number of SAD calculations is reduced up to 25% and 43% for the minimal and full sets respectively.

When V_{th} is 2 pixels, the number of SAD calculations is further reduced (up to 31%) with 0.7 dB PSNR loss in one case and with 0.2 dB PSNR loss on average for the minimum candidate set. For the full candidate set case, median filtering larger number of candidates to a single candidate results in up to 61% reduction of SAD calculations with 0.9 dB PSNR loss in one case and with 0.3 dB PSNR loss on average for a single pass of the algorithm. When three passes are done, the number of SAD calculations is reduced up to 64% with up to 0.5 dB PSNR loss and 0.2 dB PSNR loss on average. In addition, when V_{th} is set to a non-zero value, in some cases such as SthlmPan video sequence, the implicit motion vector smoothing behavior of the median filter in ATME improves the quality of the output video.

In Table 3.5, the impact of the redundancy removal technique for multiple passes of the ATME algorithm is presented. For this test, SAD_{th} parameter is set to 2500 which produces high quality results with low amount of computation. In order to determine the impact of only the multi-pass redundancy removal technique, the V_{th} parameter is set to a negative value so that all candidate vectors are evaluated. Two different cases with 3 and 5 passes of ATME algorithm are compared. Columns labeled “Red.” show the number of SAD calculations when redundancy removal technique is not used. Columns labeled “Rem.” show the number of SAD calculations when redundancy removal technique is used. Columns labeled “%” show the reduction percentage. As it can be seen from Table 3.5, the multi-pass redundancy removal technique reduces the number of SAD calculations by 25% on average in 3 passes case and 30% on average in 5 passes case.

No. of Passes	1			3			1			3		
Algorithm	**3DRS Min**	**ATME 2 Stage / Min**		**3DRS Min**	**ATME 2 Stage / Min**		**3DRS Full**	**ATME 2 Stage / Full**		**3DRS Full**	**ATME 2 Stage / Full**	
Vth	**N/A**	**0**	**2**	**N/A**	**0**	**2**	**N/A**	**0**	**2**	**N/A**	**0**	**2**
ForemanQCIF	33.09 0.01	33.09 0.01	33.08 0.01	33.82 0.04	33.82 0.04	33.85 0.03	33.75 0.04	33.75 0.03	33.63 0.03	34.51 0.12	34.41 0.11	34.39 0.08
ForemanCIF	31.92 0.06	31.92 0.06	31.77 0.05	32.61 0.17	32.61 0.16	32.50 0.15	32.02 0.15	32.02 0.15	31.97 0.13	33.08 0.46	33.02 0.42	32.94 0.35
FootballSIF	20.63 0.05	20.63 0.04	20.57 0.04	21.02 0.14	21.02 0.13	20.92 0.12	21.16 0.13	21.16 0.11	21.02 0.10	21.65 0.39	21.65 0.33	21.64 0.29
ParkJoy720p	24.23 0.52	24.23 0.50	24.08 0.44	25.86 1.58	25.93 1.44	25.83 1.25	25.11 1.38	25.11 1.32	24.74 1.09	26.21 4.20	25.94 3.63	25.68 2.74
NewMobCal720p	33.70 0.52	33.70 0.42	32.93 0.36	34.06 1.58	34.06 1.19	33.52 1.08	33.69 1.38	33.69 0.87	32.72 0.54	34.11 4.20	34.11 2.42	33.63 1.52
SthlmPan720p	33.98 0.52	33.98 0.43	34.02 0.39	34.90 1.58	34.90 1.29	34.88 1.18	34.10 1.38	34.10 0.97	34.21 0.70	35.06 4.20	35.09 3.00	35.10 2.12
InToTree720p	35.60 0.52	35.60 0.48	35.41 0.40	35.79 1.58	35.79 1.43	35.62 1.21	35.82 1.38	35.82 1.18	35.49 0.71	36.03 4.20	36.03 3.43	35.86 2.11
CrowdRun720p	26.94 0.52	26.94 0.50	26.61 0.42	27.30 1.58	27.30 1.43	26.91 1.26	27.41 1.38	27.41 1.30	27.07 1.02	28.18 4.20	28.21 3.63	27.96 2.94
ParkJoy1080p	24.13 1.16	24.13 1.14	24.25 1.04	26.01 3.52	26.01 3.27	25.96 2.90	24.70 3.09	24.70 3.01	24.46 2.66	26.02 9.39	25.92 8.36	25.80 6.69
InToTree1080p	34.40 1.16	34.40 1.10	34.29 0.92	34.51 3.52	34.51 3.23	34.44 2.76	34.50 3.09	34.50 2.79	34.34 1.82	34.62 9.39	34.62 7.99	34.54 5.19
CrowdRun1080p	27.19 1.16	27.19 1.13	27.10 1.01	27.87 3.52	27.87 3.26	27.77 2.98	27.64 3.09	27.64 2.96	27.50 2.60	28.50 9.39	28.51 8.26	28.43 7.31

Table 3.4: Performance of the First Stage of ATME Algorithm

In each table cell, upper value is the PSNR value in dB and the lower value is the number of SAD calculations scaled by 10^6.

No. of Passes	3 Pass			5 Pass		
	Red.	**Rem.**	**%**	**Red.**	**Rem.**	**%**
ForemanQCIF	54	40	26%	90	61	32%
ForemanCIF	208	161	23%	345	249	28%
FootballSIF	274	211	23%	455	333	27%
ParkJoy720p	2656	1951	27%	4385	2969	32%
NewMobCal720p	1810	1087	40%	3021	1573	48%
SthlmPan720p	1620	1103	32%	2707	1669	38%
InToTree720p	1607	1222	24%	2684	1907	29%
CrowdRun720p	2759	2076	25%	4588	3239	29%
ParkJoy1080p	5755	4381	24%	9454	6701	29%
InToTree1080p	3561	2792	22%	5947	4384	26%
CrowdRun1080p	5768	4489	22%	9558	7022	27%

Table 3.5: Multi-pass Redundancy Removal Performance
The values inside the cells of Red. and Rem. columns are the number of SAD calculations scaled by 10^3.

Table 3.6 shows the PSNR obtained and the number of SAD calculations done by the ATME algorithm with vector threshold values $V_{th}=0$ and $V_{th}=2$. For all the experiments, SAD_{th} value is set to 2500. In each table cell, upper value shows the PSNR obtained and lower value shows the number of SAD calculations done for that video sequence. For V_{th}=0, ATME algorithm generates higher quality results with same computational costs or similar quality results with lower computational costs compared to 3DRS minimal set. For V_{th} = 2 pixels, the median filtering in first stage of the ATME algorithm results in fewer SAD calculations while producing similar quality results. Moreover, in some cases such as SthlmPan video sequence, the implicit motion vector smoothing resulting from the median filtering produces higher PSNR results. The number of SAD calculations can further be decreased by using higher V_{th} values.

	Vth = 0					Vth = 2				
No. of Passes	**1 Pass**	**2 Pass**	**3 Pass**	**4 Pass**	**5 Pass**	**1 Pass**	**2 Pass**	**3 Pass**	**4 Pass**	**5 Pass**
ForemanQCIF	33.73 0.02	34.28 0.03	34.39 0.04	34.39 0.05	34.24 0.06	33.40 0.01	34.00 0.02	34.18 0.03	34.17 0.04	34.09 0.05
ForemanCIF	32.11 0.07	32.73 0.11	33.00 0.16	33.03 0.20	33.03 0.24	31.95 0.06	32.46 0.10	32.76 0.15	32.99 0.18	32.97 0.22
FootballSIF	21.20 0.09	21.50 0.15	21.67 0.20	21.77 0.26	21.83 0.32	20.90 0.07	21.39 0.13	21.63 0.19	21.64 0.24	21.72 0.30
ParkJoy720p	25.10 0.86	25.96 1.32	26.21 1.76	26.32 2.20	26.31 2.64	24.90 0.65	25.60 1.01	26.06 1.35	26.21 1.68	26.25 2.01
NewMobCal720p	33.66 0.45	34.09 0.67	34.10 0.89	34.10 1.10	34.08 1.32	32.97 0.37	33.50 0.57	33.59 0.77	33.66 0.95	33.72 1.15
SthlmPan720p	34.10 0.44	34.98 0.71	35.05 0.99	35.04 1.26	35.05 1.54	34.14 0.40	35.03 0.65	35.05 0.89	35.00 1.14	35.05 1.38
InToTree720p	35.71 0.49	35.89 0.83	35.90 1.17	35.90 1.51	35.90 1.84	35.41 0.40	35.66 0.70	35.72 0.98	35.68 1.27	35.73 1.56
CrowdRun720p	27.40 0.87	27.98 1.42	28.13 1.97	28.21 2.51	28.26 3.06	26.90 0.66	27.49 1.11	27.60 1.56	27.70 1.99	27.78 2.43
ParkJoy1080p	24.69 1.95	25.57 3.07	26.02 4.11	26.08 5.16	26.35 6.19	24.64 1.64	25.21 2.54	25.87 3.34	26.18 4.15	26.26 4.95
InToTree1080p	34.44 1.11	34.56 1.91	34.56 2.70	34.57 3.49	34.57 4.27	34.32 0.93	34.46 1.62	34.48 2.30	34.49 2.97	34.50 3.64
CrowdRun1080p	27.67 1.88	28.28 3.12	28.46 4.34	28.56 5.55	28.60 6.77	27.44 1.59	28.12 2.70	28.29 3.77	28.38 4.83	28.45 5.88

Table 3.6: Performance of the ATME Algorithm

In each table cell, upper value is the PSNR value in dB and the lower value is the number of SAD calculations scaled by 10^6.

PSNR and computational complexity comparison of two typical configurations of ATME algorithm, where V_{th} is set to 0 and 2, SAD_{th} is set to 2500 and 3 passes are done, with reference algorithms is shown in Table 3.7. The positive values in PSNR columns show the PSNR improvement by ATME algorithm. The positive values in "Red%" columns show the percentage reduction of SAD calculations by ATME algorithm. It can be seen from this table that ATME algorithm reduces the number of SAD calculations up to 82% with up to 0.59 dB PSNR loss. There are several cases where the number of SAD calculations is reduced more than 70% with less than 0.02 dB PSNR loss. In several cases, ATME algorithm produces higher PSNR results than reference algorithms while at the same time reducing the number of SAD calculations up to 58%. Therefore, ATME algorithm produces high quality video sequences with significantly lower computational cost.

	Vth = 0 / Min		**Vth = 0 / Full**		**Vth = 2 / Min**		**Vth = 2 / Full**	
	PSNR	**Red.%**	**PSNR**	**Red.%**	**PSNR**	**Red.%**	**PSNR**	**Red.%**
ForemanQCIF	0.57	9%	-0.11	66%	0.35	22%	-0.33	71%
ForemanCIF	0.39	8%	-0.08	65%	0.15	16%	-0.31	69%
FootballSIF	0.65	-42%	0.02	47%	0.61	-29%	-0.02	52%
ParkJoy720p	0.35	-12%	0.00	58%	0.26	14%	-0.09	68%
NewMobCal720p	0.04	44%	-0.01	79%	-0.49	51%	-0.54	82%
SthlmPan720p	0.15	37%	-0.01	77%	0.14	43%	-0.02	79%
InToTree720p	0.11	26%	-0.13	72%	-0.08	38%	-0.32	77%
CrowdRun720p	0.82	-25%	-0.05	53%	0.28	2%	-0.59	63%
ParkJoy1080p	0.00	-17%	0.00	56%	-0.15	5%	-0.15	64%
InToTree1080p	0.05	23%	-0.06	71%	-0.03	35%	-0.15	76%
CrowdRun1080p	0.59	-23%	-0.04	54%	0.42	-7%	-0.21	60%

Table 3.7: PSNR and Computational Complexity Comparison of ATME with Reference Algorithms

Although PSNR is a good metric for objective quality, the perceived quality of a video is not always same with its objective quality. Therefore, for evaluating the performance of FRUC algorithms, subjective quality assessments should also be made along with objective quality assessments. The same frame taken from the Foreman CIF sequences generated by Full Search, 3DRS as proposed in [21] and ATME with V_{th}=2 and SAD_{th}=2500 is shown in Figures 3.7, 3.8 and 3.9, respectively. MC-FAVG is used as the MCI algorithm in these three cases.

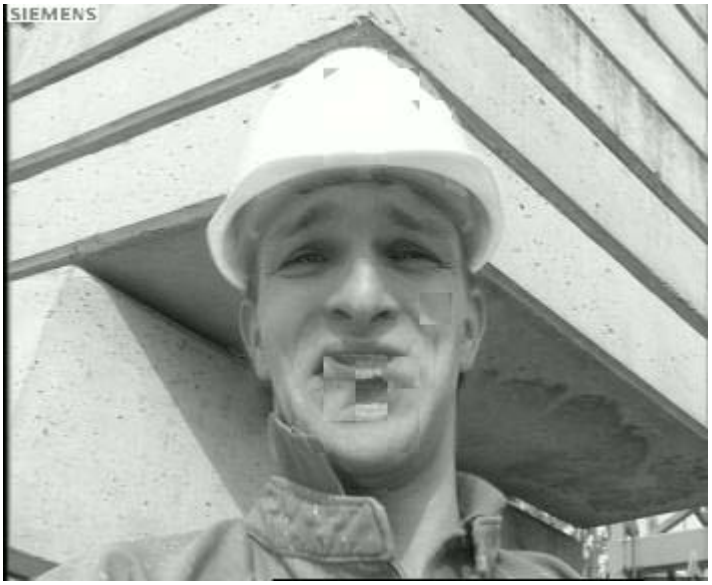


Figure 3.7: Full Search Subjective Quality Assessment

In Figure 3.7, it is clearly seen that even though FS finds the best matching SAD for each block, these MVs may not represent the true motions of the objects these blocks belong to. MV fields generated by FS are not smooth, therefore the possibility of blocking artifacts is high. On the other hand, when a true ME algorithm such as 3DRS is used, the resulting MV field is smoother, and therefore the blocking artifacts are not very likely. However, there may still be blocking artifacts when ME fails to find the true motion associated with each block. In Figure 3.8, these errors can be seen on the mouth, on the right side of the neck, on the top side of the helmet and on the text “Siemens”. These errors decrease both objective and subjective qualities of the generated video. Figure 3.9 shows that ATME algorithm performs better than 3DRS. As it can be seen from this figure, the blocking artifacts are eliminated by correct estimation of true motion vectors.

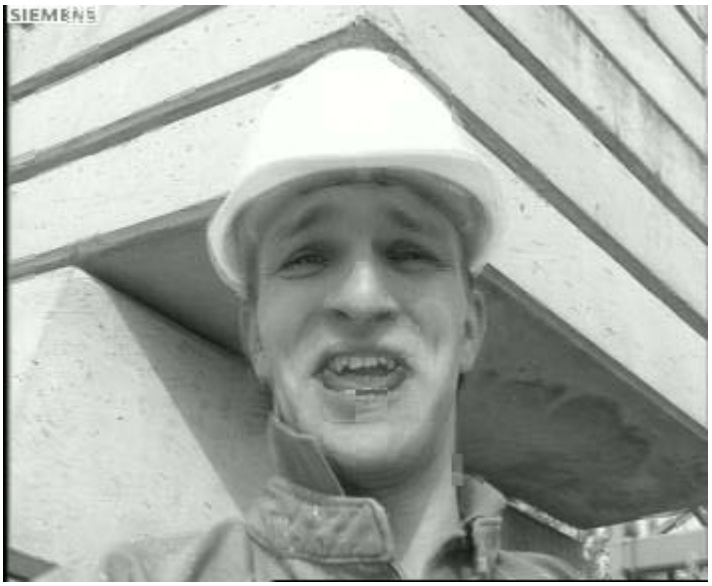


Figure 3.8: 3DRS Subjective Quality Assessment

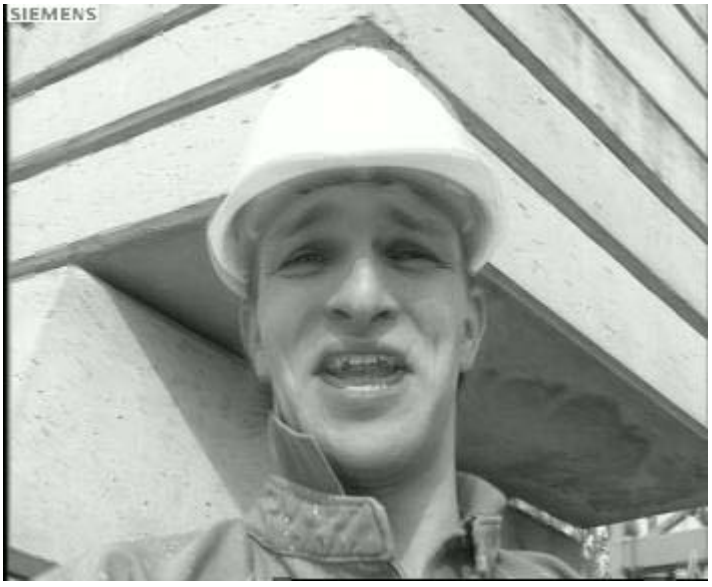


Figure 3.9: ATME Subjective Quality Assessment

More complex MCI algorithms can eliminate unpleasing artifacts and therefore improve the visual quality of the video sequence generated by FRUC. In Tables 3.8-3.12, the performances of 3DRS algorithm as proposed in [21], ATME algorithm and Full Search algorithm with 3 more complex MCI algorithms for 5 video sequences are presented.[3] MC-FAVG and non-motion compensated pixel averaging results are also given as references. As it can be seen from these tables, more complex MCI algorithms

[3] The sequences used in this experiment are: Foreman(CIF), NewMobCal(720p), SthlmPan(720p), ParkJoy(1080p), InToTree(1080p).

increases the objective quality of FRUC results when a non-true ME algorithm such as FS is used. On the other hand, these MCI algorithms do not always increase the objective quality of FRUC results when a true ME algorithm is used.

	3DRS		ATME		FS
MCI Algo.	**1 pass**	**3 pass**	**1 pass**	**3 pass**	**N/A**
MC-FAVG	30.50	31.61	31.95	32.76	31.62
Sta. Med.	31.45	32.07	31.87	32.39	32.39
Dyn. Med.	30.96	31.69	31.56	32.22	32.16
Two-Mode	30.79	31.68	31.86	32.62	32.44
Non-MC	29.86				

Table 3.8: PSNR (dB) Results of MCI Algorithms for "Foreman CIF" Sequence

	3DRS		ATME		FS
MCI Algo.	**1 pass**	**3 pass**	**1 pass**	**3 pass**	**N/A**
MC-FAVG	31.84	33.01	32.97	33.59	32.58
Sta. Med.	31.72	32.02	31.91	32.03	32.24
Dyn. Med.	31.78	32.34	32.24	32.49	32.36
Two-Mode	31.96	32.99	33.05	33.55	33.32
Non-MC	29.76				

Table 3.9: PSNR (dB) Results of MCI Algorithms for "NewMobCal 720p" Sequence

	3DRS		ATME		FS
MCI Algo.	**1 pass**	**3 pass**	**1 pass**	**3 pass**	**N/A**
MC-FAVG	33.11	34.22	34.14	35.05	30.40
Sta. Med.	27.35	27.49	27.45	27.56	27.18
Dyn. Med.	31.39	32.09	32.96	33.38	31.00
Two-Mode	32.92	34.02	34.08	34.97	31.61
Non-MC	23.96				

Table 3.10: PSNR (dB) Results of MCI Algorithms for "SthlmPan 720p" Sequence

	3DRS		ATME		FS
MCI Algo.	**1 pass**	**3 pass**	**1 pass**	**3 pass**	**N/A**
MC-FAVG	23.32	25.08	24.64	25.87	25.39
Sta. Med.	22.03	22.53	22.25	22.64	22.61
Dyn. Med.	22.92	24.15	23.79	24.79	24.75
Two-Mode	23.25	24.96	24.59	25.83	25.62
Non-MC	20.15				

Table 3.11: PSNR (dB) Results of MCI Algorithms for "ParkJoy 1080p" Sequence

	3DRS		ATME		FS
MCI Algo.	**1 pass**	**3 pass**	**1 pass**	**3 pass**	**N/A**
MC-FAVG	33.92	34.17	35.43	35.71	31.52
Sta. Med.	33.00	33.11	33.17	33.24	33.04
Dyn. Med.	33.55	33.75	33.85	33.99	32.36
Two-Mode	33.95	34.20	34.41	34.57	32.52
Non-MC	30.97				

Table 3.12: PSNR (dB) Results of MCI Algorithms for "InToTree 1080p" Sequence

The same frame taken from Foreman CIF sequences which are processed by ATME algorithm (V_{th}=2, SAD_{th}=2500) and interpolated by 4 different MCI algorithms are shown in Figures 3.10-3.13. The MCI algorithms used are MC-FAVG, Static Median Filtering, Dynamic Median Filtering, and Two Mode Interpolation (occlusion threshold = 2 pixels) respectively. In Figure 3.14, the same frame interpolated by non-motion compensated pixel averaging method is given as reference. It can be seen from these figures that blocking artifacts resulting from ME errors are removed by complex MCI algorithms. For example, errors in the stationary parts on the left side of the neck in Figure 3.10 are removed by Static Median Filter. Similarly, errors in the moving parts above the mouth are removed by Dynamic Median Filter. Two Mode Interpolation algorithm, by adaptively switching between Dynamic Median Filter and MC-FAVG, obtains a smoother image.

Figure 3.10: Subjective Assessment of MCI Algorithms - MC-FAVG

Figure 3.11: Subjective Assessment of MCI Algorithms – Static Med. Filter

Figure 3.12: Subjective Assessment of MCI Algorithms – Dynamic Med. Filter

Figure 3.13: Subjective Assessment of MCI Algorithms – Two Mode Interpolation

Figure 3.14: Subjective Assessment of MCI Algorithms – Non-Motion Compensated Interpolation

Chapter 4

ADAPTIVE TRUE MOTION ESTIMATION HARDWARE DESIGN

Three different complexity hardware architectures for implementing the ATME algorithm are proposed. In all three hardware, memory elements, MV and position values are designed to process 1080p HD frames and control signals are parameterized for processing smaller size frames.

4.1 Basic ATME Hardware

The block diagram of the first hardware, the Basic ATME hardware, is shown in Figure 4.1. The architecture consists of 6 modules and 3 on-chip memories. The Current Block contains 256x8 bits and holds the CB of size 16x16 pixels. It feeds this data to Processing Elements (PE) inside the PE Array module and is loaded when the processing for the next current block starts. The Search Block also contains 256x8 bits and holds the PB of size 16x16 pixels. It also feeds this data to PE Array and is loaded for each search location. MV Array holds the MVs for each block in a single frame. MV Array sends the candidate MVs for the CB to the Address Generator module. MV Array has two additional ports for address and data which enables external access to MV data.

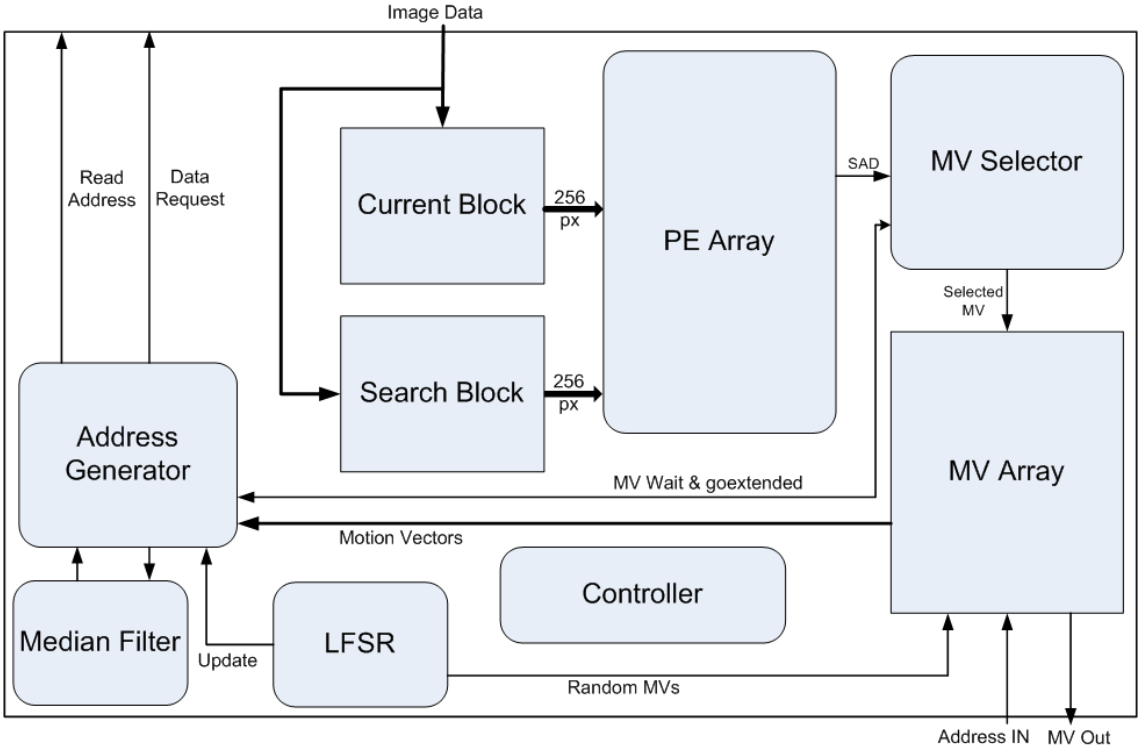


Figure 4.1: Block Diagram of Basic ATME Hardware

PE Array is the largest module and it contains 256 Processing Elements which are responsible for SAD calculation between two 16x16 pixel blocks. Each PE is composed of a comparator, two 2x1 multiplexers and an 8-bit subtractor. Each PE is responsible for calculating the SAD value between two pixels, one from the CB and one from the PB. The comparator determines which of the two pixels are greater in value. Based on the result of this comparison, two multiplexers connected to the inputs of the subtractor selects the proper pixels, the larger one to the first input and the smaller one to the second input. This ensures that the resulting value will always be positive so that the absolute difference between those two pixels is taken. This operation is done in one clock cycle, therefore 256 PEs calculate the absolute differences between the 256 pixels in CB and PB in one cycle. The outputs of PEs are connected to an adder tree to find the sum of absolute difference between two blocks. The adder tree has three pipeline stages for faster operation. Even though the SAD calculation for a single block takes 3 clock cycles, after the first SAD calculation the throughput is 1 SAD calculation per clock cycle. Therefore, the SAD calculation for 3 MVs takes 5 cycles.

MV Selector module compares the SAD values of the MV candidates for CB and selects the MV which gives the lowest SAD value. If the SAD of the selected MV is larger than a certain parameterized threshold value (SAD_{th}), it asserts a signal for using

the extended search candidates set. After the final MV is selected, it is written to the location of the CB in the MV Array. LFSR module contains a 15 bit linear feedback shift register which generates a pseudo-random number sequence. For each block evaluation, it outputs random update vectors selected from a lookup table by taking the modulus of the LFSR value. Median Filter module finds the median of three MVs and checks whether the pair-wise L1 norms of these three vectors are under V_{th} or not.

Address Generator module generates read and write addresses for Search Block. In addition, depending on the results of vector threshold and SAD threshold techniques, it selects which MV will be evaluated next and adds random update vectors when necessary. Controller module keeps track of state, the position of current block and other control signals necessary for correct operation of the other modules.

4.1.1 Operation of Basic ATME Hardware

The operation of ATME hardware begins with the *start* signal. Controller keeps the count and location of the CB. It provides the *current position* signal to MV Array which is used as the write address of the MV selected for the CB. The processing of the first frame is a special case, where MV Array is initially empty. Therefore, for each block in the first frame a random update vector taken from LFSR is written to the corresponding address in the MV Array. LFSR is a 15 bit linear feedback shift register with a 2 tap primitive polynomial where 14^{th} and 15^{th} bits of the shift register are XNORed. This LFSR produces a pseudo-random number sequence from 0 to 32766. In the software implementation, the random update vector set contained 25 elements. However, for modulus values other than the powers of 2, modulus operation requires a division. Therefore, in order to simplify the modulus hardware, 7 more update vectors are added to the random update vector set making a total of 32 elements.

When MV Array is filled with random vectors after the processing of the first frame, *frameend* signal is asserted by Controller module. Then, processing of the next frame starts. First, Current Block is filled with current block pixels in 32 cycles (8 pixels per cycle). After CB is filled, the control is handed to the Address Generator module. First, it gets 3 MVs that will be evaluated for the CB from MV Array and sends them to Median Filter module. Median Filter calculates the median of these 3 MVs and sends it to Address Generator. Median Filter also calculates the pair-wise L1 Norms of these MVs and sends a signal, *underth*, if all of them are under the vector threshold, V_{th}. Address Generator then calculates the starting address of the Search Block pixels in off-

chip SRAM, reads the search block pixels and stores them to Search Block. After SB is filled with search block pixels in 32 cycles, *SBFilled* flag is set. Then *currentMV* is set to the median MV and the SAD for the median MV is calculated. The next state depends on the value of *underth* signal. If *underth* is 1, an update vector is added to the median MV and assigned as *currentMV*. On the other hand, if *underth* is 0, an update vector is added to the second MV and assigned as *currentMV*. Address Generator reads the corresponding search block pixels from off-chip SRAM and stores them to Search Block. After SB is filled, *SBFilled* flag is set and the SAD for the current MV is calculated. If *underth* is 0, the SAD for the third MV is also calculated. After the SAD values for the MVs are calculated, Address Generator informs MV Selector and waits for the *blockend* signal.

MV Selector stores all the MVs processed for the CB and their SAD values. For each CB, either two or three MVs are processed. After all the MVs are processed for the CB, depending on the value of the counter, MV Selector compares the SAD values of these two or three MVs and outputs the MV with the minimum SAD. The minimum SAD value is compared with the SAD_{th} parameter. If the minimum SAD is higher than the predetermined SAD threshold, then *goextended* signal is asserted. If this signal is asserted, the SAD values for the MVs in the extended candidate set are calculated. After the SAD values for all MVs are calculated, the MV with the minimum SAD is written to *current position* address of MV Array.

MV Array is composed of 8 dual-port Block RAMs. The first port is used for writing and reading MVs inside the ATME hardware. The second port is configured as read only and provides MV data outside the ATME hardware. After the MV for the CB is written, *blockend* signal is asserted and Controller starts processing the next block. After all blocks in a frame are processed, *frameend* signal is asserted and Controller starts processing the next frame.

4.1.2 Implementation Results of Basic ATME Hardware

The basic ATME hardware architecture is implemented in Verilog HDL. Since the total number of cycles needed for processing a frame is not deterministic, in order to find an average value, 10 frames from NewMobCal720p video sequence are processed to double the frame rate. This operation took ~3807000 cycles, therefore on average a frame is processed in ~380700 cycles.

The Verilog RTL code of the basic ATME hardware is synthesized to a 4vlx200ff1513 Xilinx Virtex-4 FPGA with speed grade -11 using Mentor Graphics Precision RTL tool. The resulting netlist is placed and routed to the same FPGA using Xilinx ISE tool. The hardware implementation is verified with post place and route simulation using Mentor Graphics Modelsim tool. The hardware uses 13425 4-input LUTs, 5327 Flip-Flops, 8 dual-port Block RAMs, and consumes 8% of the Slices. It works at 59.86 MHz and is capable of processing ~158 720p HD frames per second doubling the frame rate to ~316 fps which satisfies the real-time requirements.

4.2 ATME Hardware with Update Window

The block diagram of the ATME hardware with Update Window (UW) is shown in Figure 4.2. The 22x22 pixel UW is constructed by enlarging the 16x16 pixel Search Block by 3 pixels in each direction in order to implement an efficient data re-use scheme. UW is implemented as a 22 22x1 pixel distributed memory block. There are two reasons for using an UW of size 22x22. First, since true motion estimation creates smooth MV fields around objects, the MVs that will be evaluated for a block are expected to be similar. Second, a random update vector is always added to one of the MVs in the ATME algorithm. Since the random update vector set consists of vectors in [-3,+3] pixels range, the updated MV will always be inside the UW of the MV that is updated. Therefore, before processing a CB, the UW is filled with pixels centered on the location pointed by the median of the three vectors in the minimal set. If pair-wise L1 Norms of these three vectors are less than V_{th}, their median MV will be evaluated along with its updated version, in which case all required pixels will be inside the UW. On the other hand, if any L1 Norm is larger than V_{th}, the pixels required for second and third MVs in the set will probably be inside the UW. Address Generator fills the UW with proper pixels based on the current MV, and checks whether the pixels required for the next MV is inside the UW or not. If all the pixels are inside the UW, the SAD calculation for that MV is done. However, if any pixel required for the SAD calculation of that MV is not inside the UW, the SAD calculation is done after the entire UW is refilled with the required pixels for that MV.

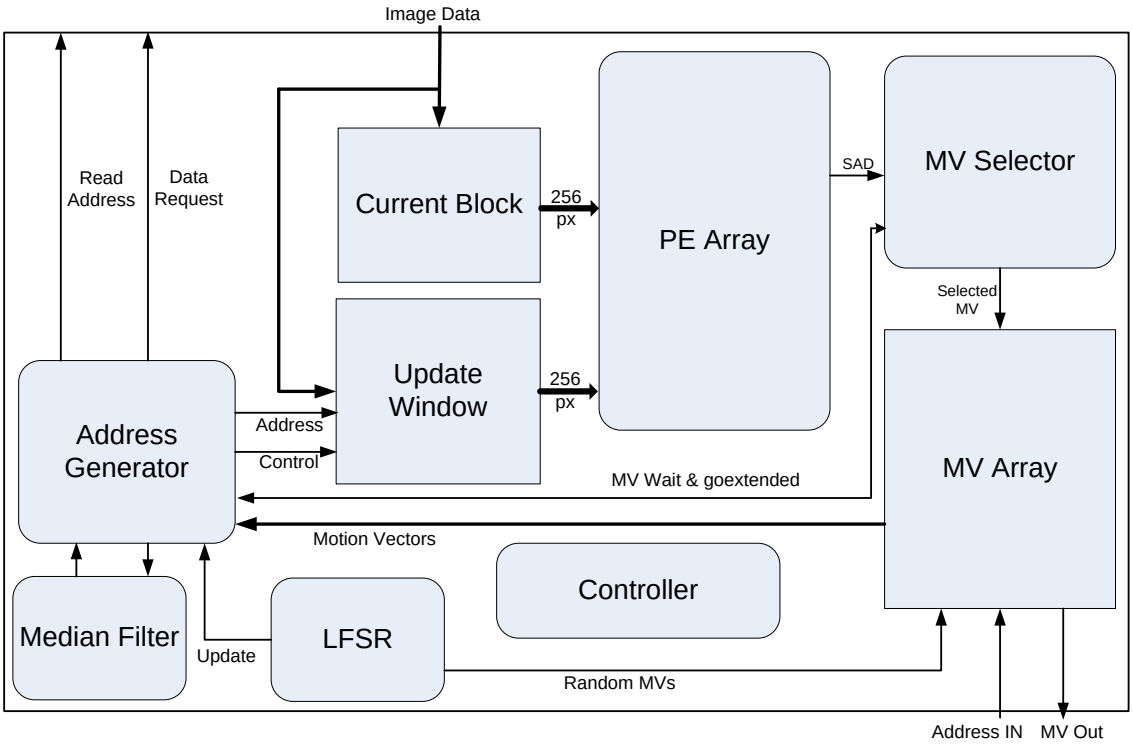


Figure 4.2: Block Diagram of ATME Hardware with UW

In order to select the 16x16 pixel PB from the 22x22 pixel UW, horizontal and vertical multiplexers are used. These multiplexers can select any 16x16 pixel block inside the UW and sent it to the PE Array as the PB for SAD calculation. This selection is implemented in two steps. First, a 7x16 multiplexer, the horizontal multiplexer, selects the 16 columns of the UW which contain the columns of PB. Then, a 7x16 multiplexer, the vertical multiplexer, selects the 16 rows of the output of the horizontal multiplexer. The select signals for horizontal and vertical multiplexers are sent by the Address Generator. The resulting 16x16 pixel PB is sent to the PE Array.

The operation of UW is illustrated in Figure 4.3. This figure shows the case where the UW is centered by the MV (1,1). Therefore, the center of UW, i.e. the 16x16 block starting from the 4^{th} row and 4^{th} column of UW, contains the PB that is located (1,1) away from the CB. The (-2,+1) random update vector is added to that MV, therefore the next MV that will be evaluated is (-1,+2) and the PB pointed by this MV is inside the UW. Since the columns of the 16x16 PB are located in 2^{nd} column to 17^{th} column of UW, the select signal for the horizontal multiplexer sent by the Address Generator is 1. Since, the rows of the 16x16 PB are located in 5^{th} row to 20^{th} row of UW, the select signal for the vertical multiplexer sent by the Address Generator is 4.

To show the advantage of using the UW, the number of pixels read from off-chip SRAM by Basic ATME Hardware and ATME Hardware with Update Window are shown in Table 4.1. The column labeled "Re-Use" show the data re-use percentage. The ATME configuration is V_{th} = 2 pixels, SAD_{th} = 2500, and a single pass is done. As it can be seen from this table, other than 2 very static video sequences (SthlmPan and InToTree), the number of pixels read from off-chip SRAM is reduced. This increases the performance and reduces the power consumption of the ATME hardware.

Sequence	Basic (10^3)	w/ UW (10^3)	Re-Use
ForemanCIF	24998	24262	3%
ParkJoy720p	234890	227041	3%
NewMobCal720p	185530	173865	6%
SthlmPan720p	198571	200823	-1%
InToTree720p	204040	220666	-8%
CrowdRun720p	246847	205649	17%
ParkJoy1080p	561073	541651	3%
InToTree1080p	464933	524882	-13%
CrowdRun1080p	603929	543362	10%

Table 4.1: Number of Pixels Read from Off-Chip SRAM

4.2.1 Implementation Results of ATME Hardware with Update Window

The ATME hardware with Update Window is implemented in Verilog HDL. NewMobCal720p video sequence is processed for 10 frames to double the frame rate. This operation took ~3740000 cycles, therefore on average a frame is processed in ~374000 cycles. The Verilog RTL code of the ATME hardware with Update Window is also synthesized to a 4vlx200ff1513 Xilinx Virtex-4 FPGA with speed grade -11 using Mentor Graphics Precision RTL tool. The resulting netlist is placed and routed to the same FPGA using Xilinx ISE tool. The hardware implementation is verified with post place and route simulation using Mentor Graphics Modelsim tool. The hardware uses 33773 4-input LUTs, 7442 Flip-Flops, 8 dual-port Block RAMs, and consumes 21% of the Slices. It works at 62.63 MHz and is capable of processing ~168 720p HD frames per second doubling the frame rate to ~336 fps which satisfies the real-time requirements.

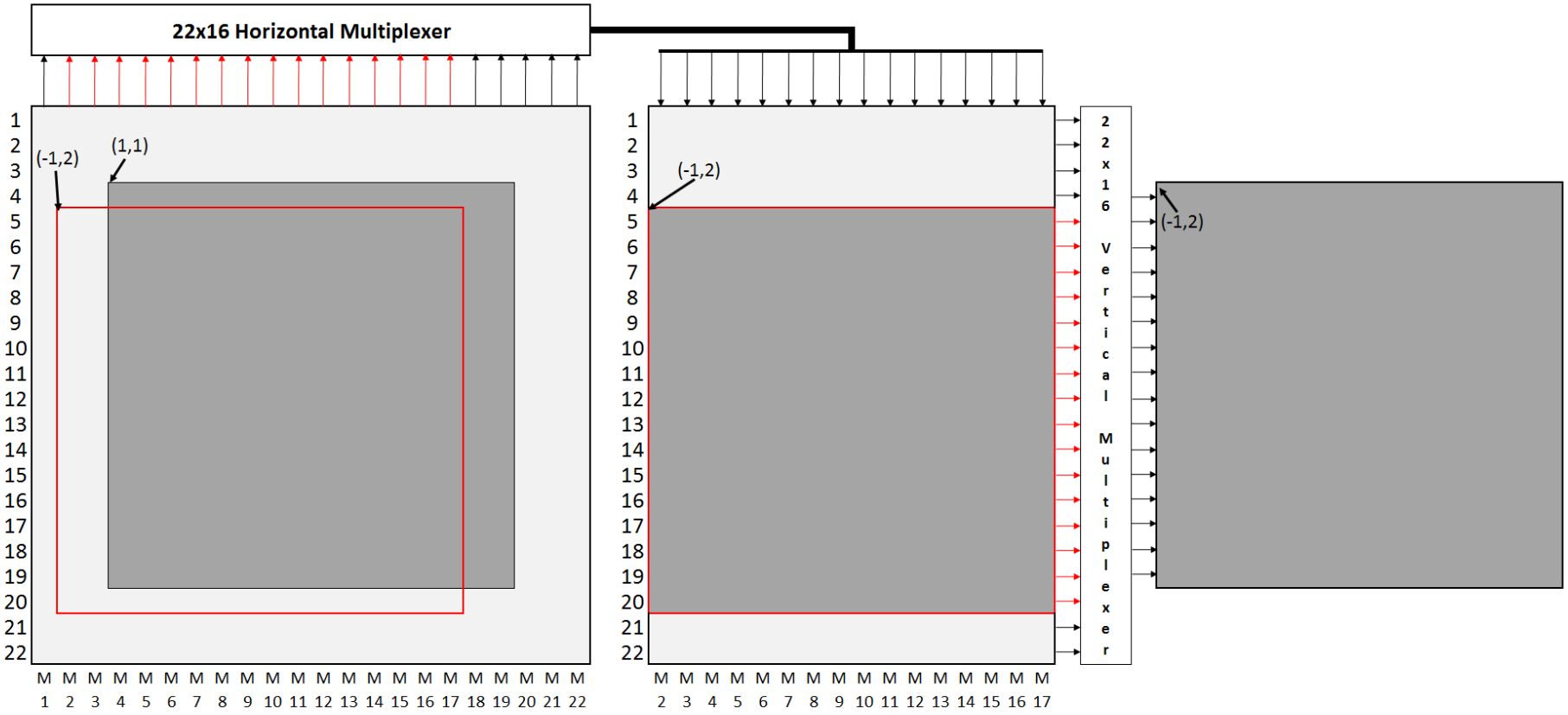


Figure 4.3: Operation of Horizontal and Vertical Multiplexers in UW

4.3 ATME Hardware with Search Window

In *ATME Hardware with Update Window*, the pixels in the 22x22 pixel UW are re-used only when UW contains all the pixels in the block pointed by the current MV. If any pixel in this block is not in the UW, then the entire 22x22 pixel UW is re-filled. However, if most of the pixels in this block are inside the UW, instead of refilling the entire UW, only the missing pixels can be loaded into UW. If the number of rows and columns that should be loaded into UW is more than 22, it is inefficient to load them into UW one at a time. However, if the number of rows and columns that should be loaded into UW is less than 22, then by loading them one at a time and re-using the rest of the pixels already in the UW, the number of accesses to off-chip SRAM can be reduced, performance can be increased and power consumption can be reduced.

Therefore, ATME Hardware with Search Window implements this data re-use technique. When this data re-use technique is used, the existing pixels in the rows or columns of the UW are replaced with the new pixels. In this case, the addressing scheme for the UW is rotated so that this replacement is not visible to the rest of the hardware in terms of read and write addresses.

The process of replacement in UW for the case where UW is centered on location (4,3) and the next MV that will be evaluated is (8,8) is shown in Figure 4.4. In this case, in order for the UW to include the PB, two rows and one column should be replaced in the UW. After the replacement in the UW, proper select signals are sent to the horizontal and vertical multiplexers. In this hardware, 22x16 horizontal and vertical multiplexers are used in order to be able select any 16x16 pixel PB. For the case shown in Figure 4.4, the select signal for the horizontal multiplexer is 7, and the select signal for the vertical multiplexer is 8. This selection process is shown in Figure 4.5.

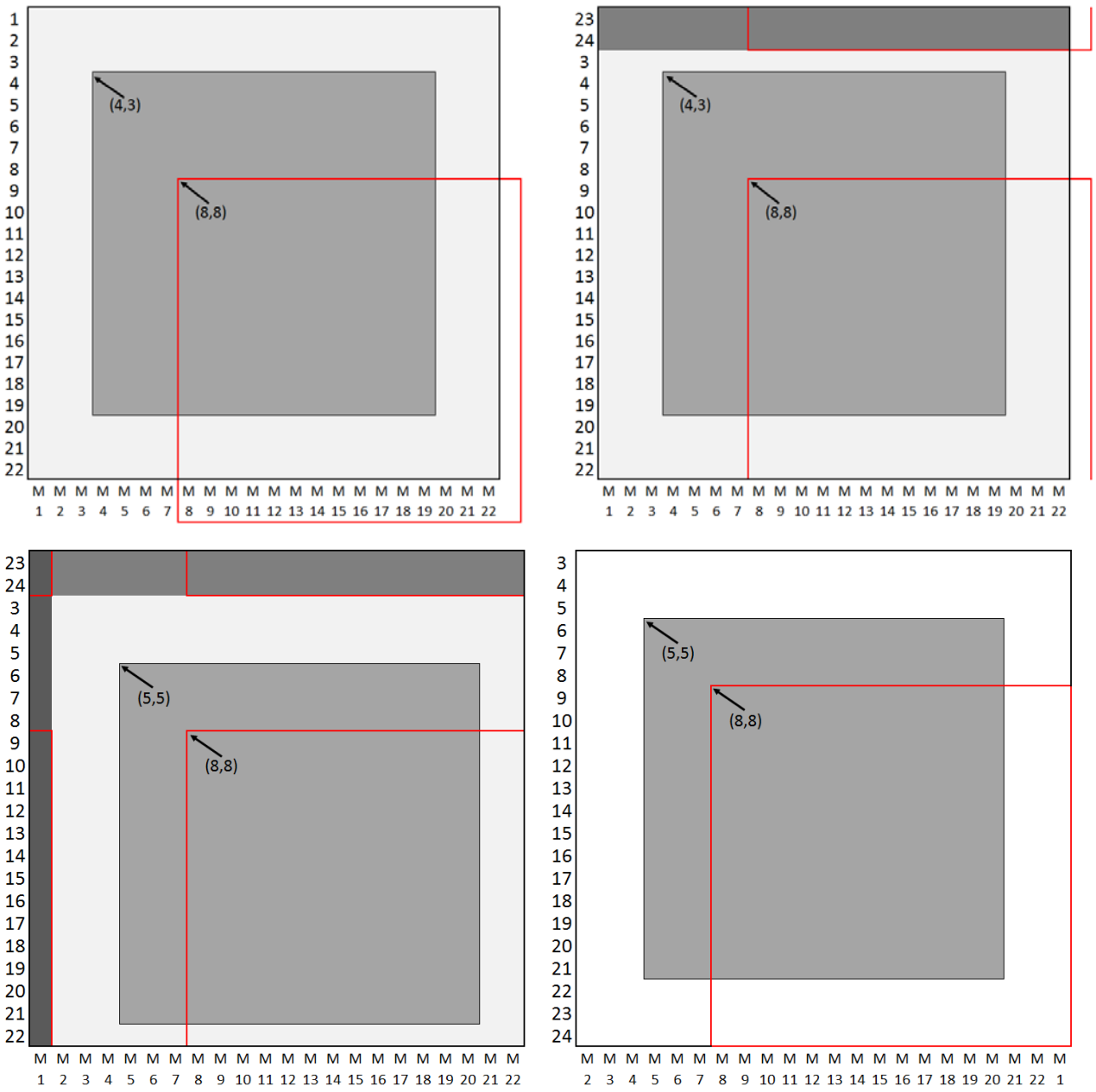


Figure 4.4: Replacement in UW

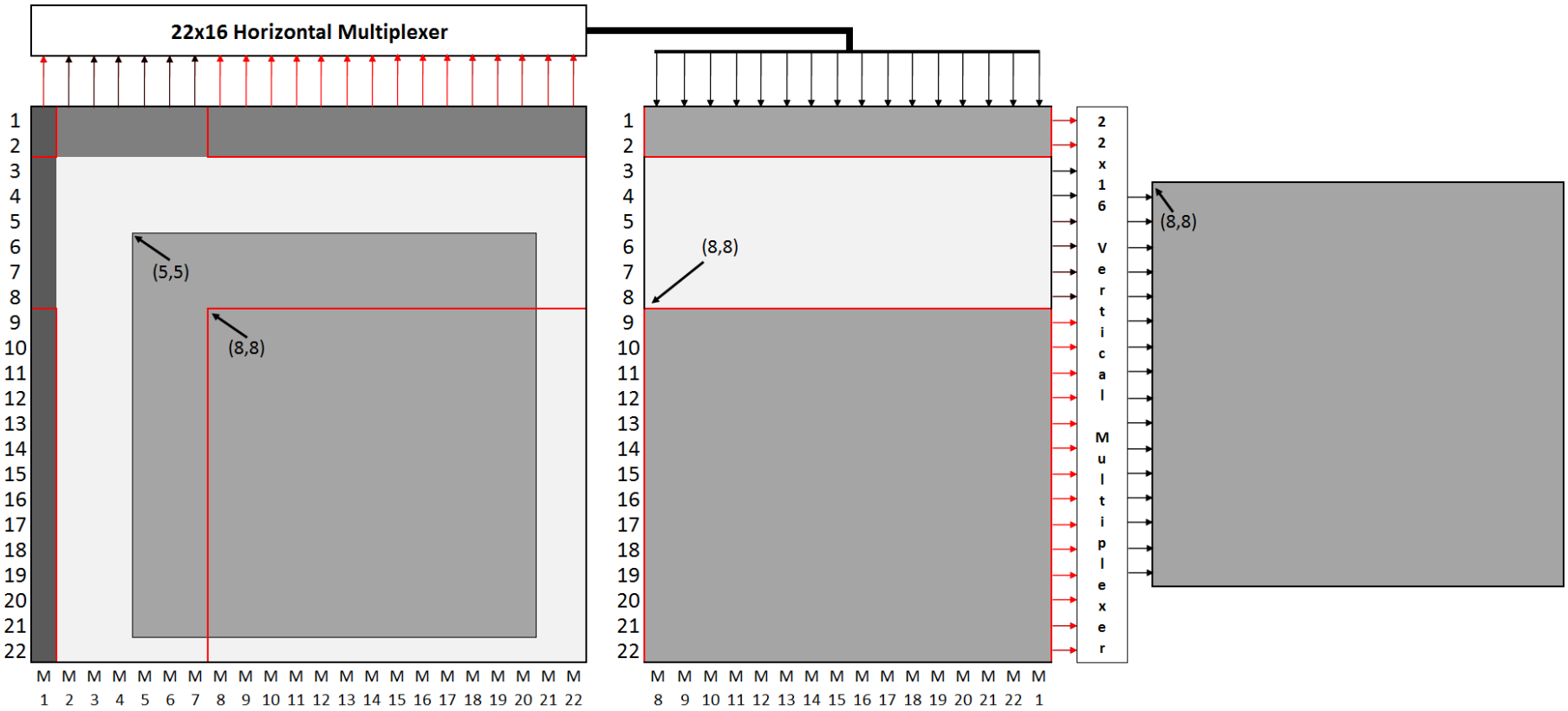


Figure 4.5: Operation of Horizontal and Vertical Multiplexers in ATME Hardware with SW

The number of pixels read from off-chip SRAM by three ATME hardware for different video sequences are shown in Table 4.2. As it can be seen from this table, this technique significantly reduces the number of off-chip SRAM accesses.

Sequence	Basic (10^3)	w/ UW (10^3)	Re-Use	w/ SW (10^3)	Re-Use
ForemanCIF	24998	24262	3%	20733	17%
ParkJoy720p	234890	227041	3%	202666	14%
NewMobCal720p	185530	173865	6%	171547	8%
SthlmPan720p	198571	200823	-1%	181274	9%
InToTree720p	204040	220666	-8%	195317	4%
CrowdRun720p	246847	205649	17%	180435	27%
ParkJoy1080p	561073	541651	3%	470774	16%
InToTree1080p	464933	524882	-13%	465308	0%
CrowdRun1080p	603929	543362	10%	436811	28%

Table 4.2: Number of Pixels Read from Off-Chip SRAM by ATME Hardware

The block diagram of ATME Hardware with Search Window is shown in Figure 4.6. Video frames are stored in the off-chip SRAM in row-major or column-major order. Therefore, in order to be able to access proper pixels consecutively from rows and columns of a frame in each cycle, an on-chip Search Window memory implemented with dual-port Block RAMs is used in this hardware. SW size can be multiples of 22. In this hardware, SW contains 88x88 8-bit pixels, centered on the position of CB. These pixels are distributed into 22 dual-port Block RAMs. This requires limiting MV values to a range of [-36,+36] pixels. The necessary checks for this MV limitation are implemented in the Address Generator module.

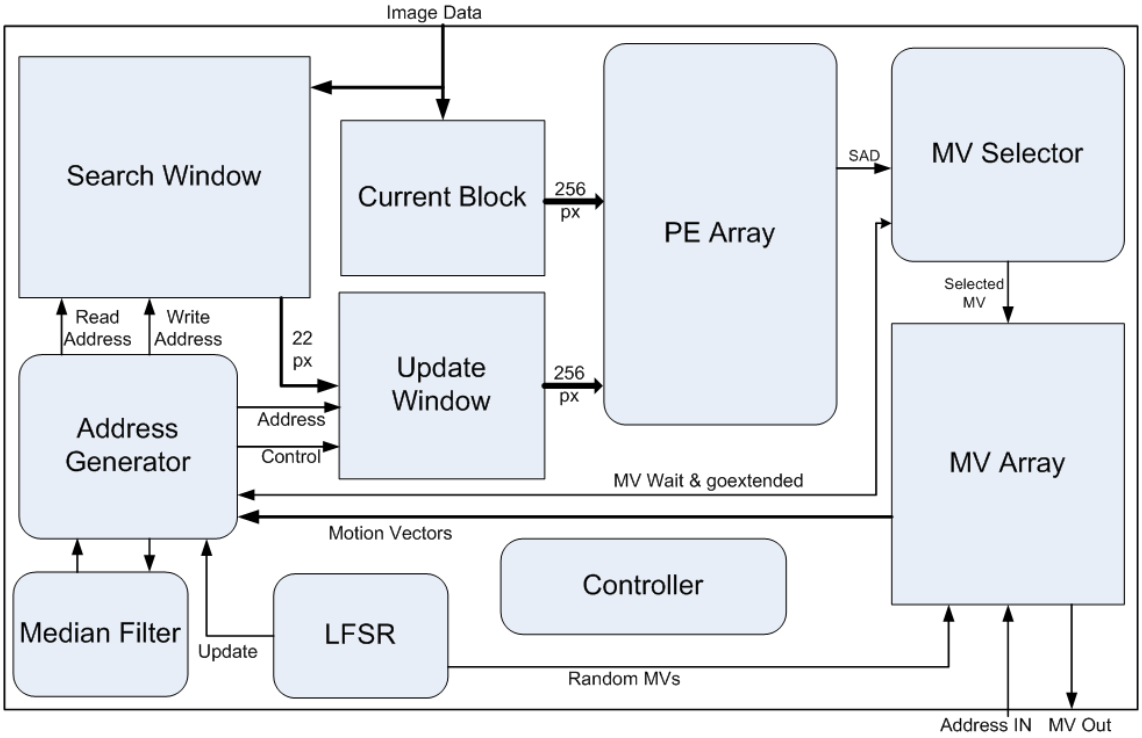


Figure 4.6: Block Diagram of ATME Hardware with SW

The pixels are placed diagonally in the SW as shown in Figure 4.7 [17]. The numbers in each cell indicate the Block RAM containing the corresponding pixel in SW. Each Block RAM is configured as dual-port, one port is used for writing and the other port is used for reading. Block RAMs in Xilinx FPGAs can be configured with different port widths. In this hardware implementation, the write port is configured as 32 bits and the read port is configured as 8 bits. Therefore, 4 pixels can be written into and 1 pixel can be read from each Block RAM in each cycle. A limited number of, generally 64, bits can be read from off-chip SRAM in each cycle. Therefore, only 8 pixels can be written into SW in one cycle and each column of SW is filled in 11 cycles.

The placement of pixels in Block RAMs is shown in Table 4.3. The two numbers inside each cell indicate the row and column of the SW the corresponding pixel belongs to. The first 22 pixels in a column of SW are stored in a different Block RAM. After 22 pixels starting from the top left pixel of SW is written into 22 Block RAMs, the 23rd pixel is written into the next location of first Block RAM. Therefore, consecutive four locations in a Block RAM contain four pixels from the same column of SW.

Address	BRAM1	BRAM2	…	BRAM11	BRAM12	…	BRAM21	BRAM22
0	(1, 1)	(2,1)	…	(11,1)	(12,1)	…	(21,1)	(22,1)
1	(23,1)	(24,1)	…	(33,1)	(34,1)	…	(43,1)	(44,1)
2	(45,1)	(46,1)	…	(55,1)	(56,1)	…	(65,1)	(66, 1)
3	(67,1)	(68,1)	…	(77,1)	(78,1)	…	(87,1)	(88,1)
4	(22,2)	(1,2)	…	(10,2)	(11,2)	…	(20,2)	(21,2)
5	(44,2)	(23,2)	…	(32,2)	(33,2)	…	(42,2)	(43,2)
⋮	⋮	⋮		⋮	⋮		⋮	⋮
170	(47,43)	(48,43)	…	(57,43)	(58,43)	…	(45,43)	(46,43)
171	(69,43)	(70,43)	…	(79,43)	(80,43)	…	(67,43)	(68,43)
172	(2,44)	(3,44)	…	(12,44)	(13,44)	…	(22,44)	(1,44)
173	(24,44)	(25,44)	…	(34,44)	(35,44)	…	(44,44)	(23,44)
⋮	⋮	⋮		⋮	⋮		⋮	⋮
346	(47,87)	(48,87)	…	(57,87)	(58,87)	…	(45,87)	(46,87)
347	(69,87)	(70,87)	…	(79,87)	(80,87)	…	(67,87)	(68,87)
348	(2,88)	(3,88)	…	(12,88)	(13,88)	…	(22,88)	(1,88)
349	(24,88)	(25,88)	…	(34,88)	(35,88)	…	(44,88)	(23,88)
350	(46,88)	(47,88)	…	(56,88)	(57,88)	…	(66,88)	(45,88)
351	(68,88)	(69,88)	…	(78,88)	(79,88)	…	(88,88)	(67,88)

Table 4.3: Locations of the SW Pixels in Block RAMs

After the MV for CB is found, SW for the next CB should be loaded. The proposed hardware refills the entire SW only for the first CB in each block row of the input frame. Since the SW for CB and SW for the next CB have a 72x88 pixels overlap, instead of reading entire 88x88 pixels of the SW from off-chip SRAM for the next CB, the proposed hardware reads 16 non-overlapping columns from the off-chip SRAM and writes them to the leftmost 16 columns in SW. This data re-use scheme requires rotating read addresses for the SW for each new CB. This address rotation is handled by the Address Generator.

The address rotation between the first CB and the next CB in a frame is illustrated in Figure 4.8. In this figure, the numbers over the columns and the numbers to the left of the rows show the actual positions of the columns and rows inside the video frame respectively. The symbols inside the cells show the Block RAMs containing the corresponding pixels. As it can be seen from Figure 4.8(a), for the next CB, 16 new SW columns (89 to 104) are needed and SW columns 1 to 16 are not needed. Therefore, the new 16 columns are written to first 16 columns of the SW as shown in Figure 4.8(b).

Because of address rotation, other modules in the hardware perceive the SW as shown in Figure 4.8(c).

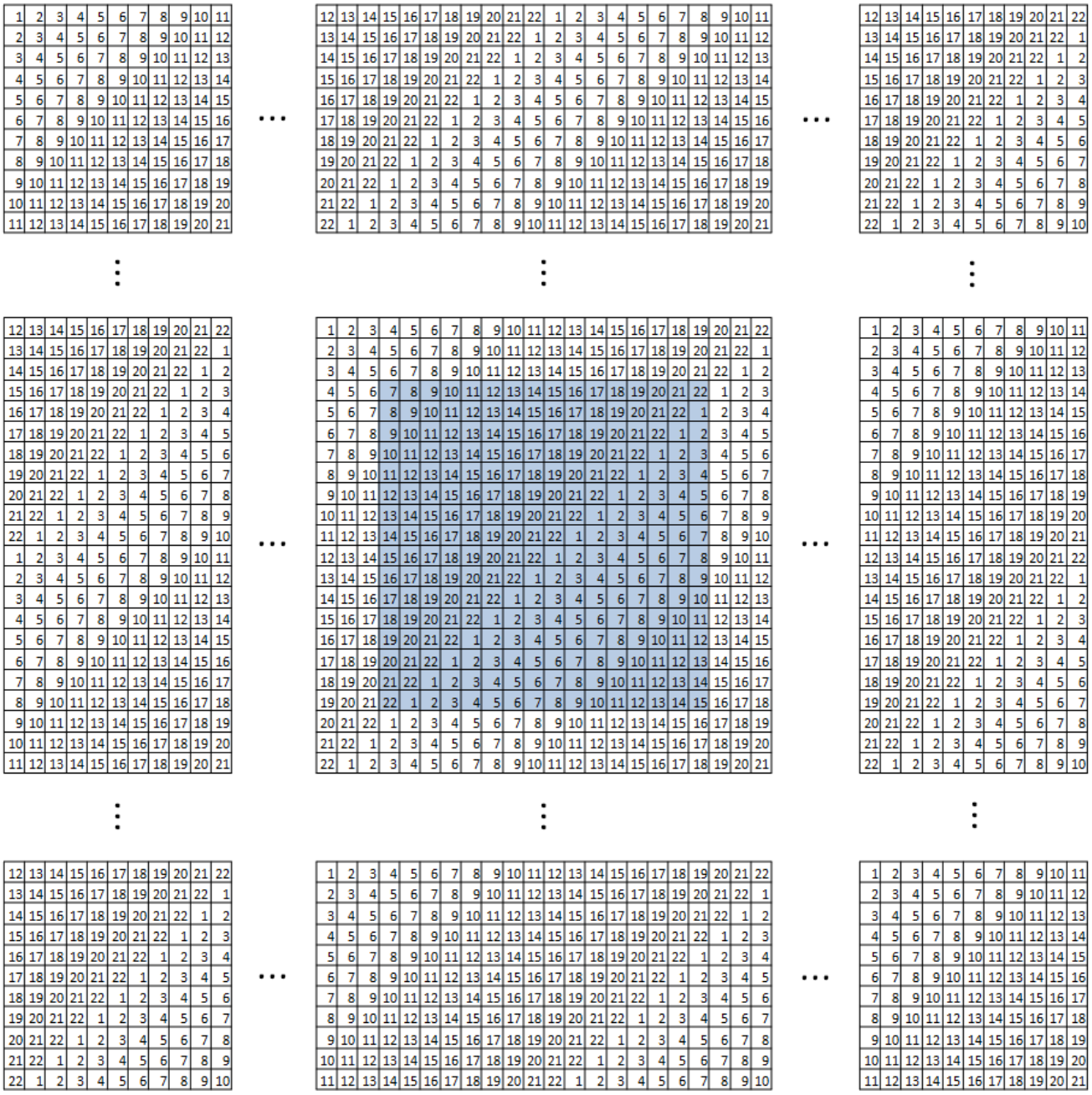


Figure 4.7: Diagonal Placement in SW

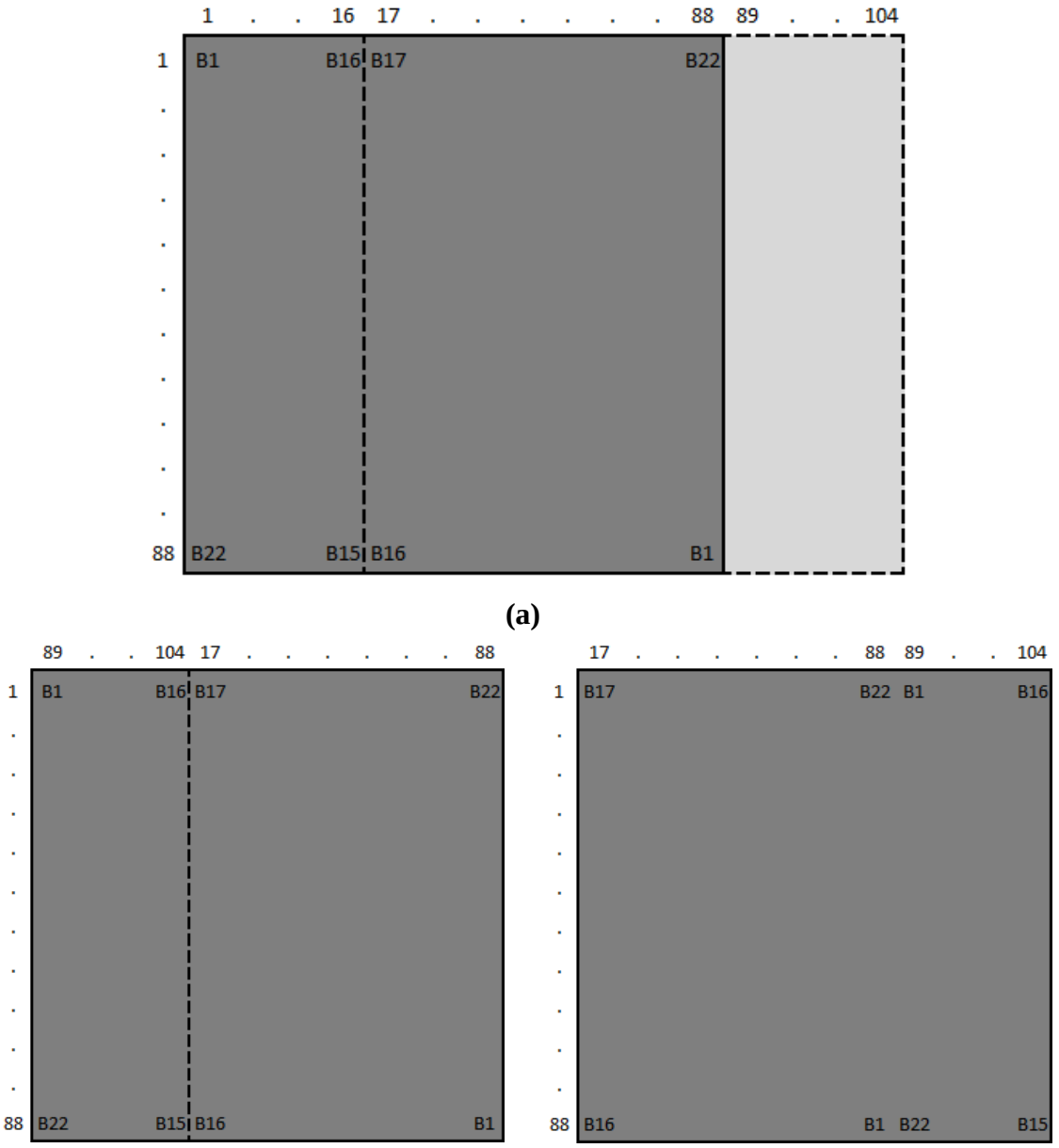


Figure 4.8: Address Rotation for SW (a) Overlapping pixels in SWs (b) Actual placement of pixels (c) Perceived placement of pixels.

In ATME algorithm, the access pattern for the SW depends on the values of the MVs that are evaluated. For example, if the MVs that will be evaluated for the CB are (1,0), (0,1) and (2,2), then PB is first accessed from the location one column right to CB. The PB is next accessed from one row below CB, and finally PB is accessed from two rows and two columns away from CB. Diagonal placement of pixels inside SW allows accessing any 22 pixel row or column in the SW in one cycle.

Address Generator calculates the starting addresses of the pixels that will be sent to the UW and reads them from 22 Block RAMs. Because of the SW address rotation, the starting address calculations are quite complex. For example, when the first pixel of the UW is read from the 15th Block RAM, next 7 pixels in the column should be read the same address of 16th to 22nd Block RAMs. However, the next 13 pixels in the column should be read from the next address of 1st to 14th Block RAMs. Since a Block RAM has 4 pixels from the same column, the addresses of the pixels in the following columns are calculated by adding 4 to the address of the pixel on the same row of the previous column.

In ATME Hardware with SW, large number of read accesses to off-chip SRAM is done in order to fill the on-chip SW memory. The number of pixels read from SRAM when the SW size is 66x66 pixels and frame size is 1080p HD is $\big((66x66x1) + (16x66x119)\big)x67 = \mathbf{8,711,340}$ pixels per frame. For 100 frames (98 frames processed) 8,711,340 x 98 $= \mathbf{853,711,320}$ pixels. The reason for 98 frames being processed is that MVs for the first frame pair are assigned randomly and the 100th frame is taken from the original video sequence, therefore SW is not filled in those two cases. Similarly, the number of pixels read from SRAM when the SW size is 88x88 pixels and frame size is 1080p HD is $\big((88x88x1) + (16x88x119)\big)x67 = \mathbf{11,744,832}$ pixels per frame. For 100 frames (98 frames processed) 11,744,832 x 98 $= \mathbf{1,150,993,536}$ pixels.

Therefore, using an on-chip SW memory for the ATME algorithm with a candidate set with small number of locations is not efficient. However, when a candidate set with large number of locations is used to obtain higher quality videos, using an on-chip SW memory becomes efficient especially for large frame sizes. For example, when ParkJoy1080p sequence is processed for 100 frames by an ATME algorithm with a candidate set with 14 locations and V_{th} = 2, the number of pixels accessed is 895,458,168. And, when InToTree1080p sequence is processed by the same ATME algorithm, the number of pixels accessed is 934,038,534. If a 66x66 pixel size SW is used, the number of accesses to off-chip SRAM is reduced for both examples.

The ATME Hardware with SW is implemented in Verilog HDL. However, the Verilog RTL code is not mapped to an FPGA.

Chapter 5

CONCLUSION AND FUTURE WORK

In this thesis, adaptive true motion estimation (ATME) algorithm based on 3-D Recursive Search algorithm is proposed for frame rate up-conversion. By using multi-objective genetic algorithm, an optimized set of candidate locations are obtained. The experimental results show that this optimized set improves the results of the 3-D Recursive Search algorithm up to 2 dB. In addition, an extended set of candidates is proposed to be used in cases where the results of the first set of candidates are unsatisfactory.

Several computational complexity reduction and redundancy removal techniques are used in ATME algorithm to reduce the number of SAD calculations. The first technique avoids the evaluations of the same MV candidates. The next technique avoids the evaluations of the similar MV candidates. The similarity of the MVs is determined by comparing their pair-wise distances to a predefined threshold value. When the threshold is set to zero, the same quality results are obtained with a 20% reduction in SAD calculations for a 3 candidate set and 38% reduction for an 8 candidate set. This reduction is further increased when multiple passes of the algorithm are done. When the threshold is set to a non-zero value, the number of SAD calculations is reduced up to 64% with an average PSNR loss of 0.2 dB.

A redundancy removal technique for multiple passes is used in the ATME algorithm. The probability of evaluating the MV, which is selected as the best matching candidate for a block in the first pass of the algorithm, in the next pass is quite high. Therefore, this technique stores the best SAD value obtained in the previous pass for each block and uses them in the next pass in order to avoid redundant SAD calculations. This multi-pass redundancy removal technique reduces the number of SAD calculations by 25% on average in 3 passes and 30% on average in 5 passes.

The experimental results show that the ATME algorithm produces higher PSNR results than reference algorithms while at the same time reducing SAD calculations up to 58%. Furthermore, ATME algorithm reduces SAD calculations up to 82% with up to 0.59 dB PSNR loss. There are several cases where there is more than 70% reduction in SAD calculations with less than 0.02 dB PSNR loss. Therefore, ATME algorithm produces high quality video sequences with significantly lower computational cost.

In addition, in this thesis, three efficient hardware architectures for ATME algorithm are proposed. The first hardware is a basic implementation of ATME algorithm. Off-chip SRAM accesses are costly both in terms of latency and power consumption. Therefore, the second hardware implements a data re-use scheme using a 22x22 pixel Update Window by exploiting the smoothness property of true motion vector fields. The third hardware uses a technique for loading the Update Window with only the pixels missing in the UW. An on-chip Search Window memory is used to efficiently implement this technique. The pixels are diagonally placed into 22 dual-port Block RAMs of the SW in order to provide single cycle access to any 22 pixel row or column inside the SW.

All three ATME hardware architectures are implemented in Verilog HDL. However, only two of them are mapped to Xilinx Virtex-4 FPGA. *Basic ATME Hardware* consumes 8 Block RAMs and 8% of the Slices in that FPGA. It works at 59.86 MHz and is capable of processing ~158 720p HD frames per second, which is sufficient for real-time processing. *ATME Hardware with Update Window* consumes 8 Block RAMs and uses 21% of the Slices in the same FPGA. It works at 62.63 MHz and is capable of processing ~168 720p HD frames per second.

As future work, the third ATME hardware can be mapped to an FPGA. The redundancy removal technique for multiple passes can be implemented and integrated into the ATME hardware. A complete FRUC system can be built by designing and implementing an MCI hardware and integrating it to the ATME hardware.

BIBLIOGRAPHY

[1] G. de Haan, *Video Processing for Multimedia Systems*. Univ. Press Eindhoven, ISBN 90-9014015-8, 2001.

[2] O. A. Ojo and G. de Haan, "Robust Motion-Compensated Video Upconversion," *IEEE Trans. Consum. Electron.*, vol. 43, no. 4, pp. 1045-1056, Nov. 1997.

[3] M. Tekalp, *Digital video processing.* Englewood Cliffs, NJ: Prentice Hall, 1995.

[4] N. Netravali and J. D. Robbins, "Motion-adaptive interopolation of television frames," *Proc. Picture Coding Symp.*, Jun. 1981, p. 115.

[5] K. A. Bugwadia, E. D. Petajan, and N. N. Puri, "Progressive-Scan Rate Up-Conversion of 24/30 Source Materials for HDTV," *IEEE Trans. Consum. Electron.*, vol. 42, no.3, pp. 312-321, Aug. 1996.

[6] K. A. Bugwadia, E. D. Petajan, N. N. Puri, "Progressive-Scan Rate Up-Conversion of 24/30 Hz Source Materials for HDTV," *IEEE Trans. Consum. Electron.*, vol. 42, no. 3, pp. 312-321, Aug. 1996.

[7] T. Koga, K.Iinuma, and T. Ishiguro, "Motion compensated interframe coding for video conferencing," *Proc. NTC*, pp. G5.3.1-G5.3.5, New Orleans, USA, Dec. 1981.

[8] A. Puri, H. M. Hang, and D. L. Schilling, "An efficient block matching algorithm for motion compensated coding," *Proc. IEEE ICASSP*, pp. 1063-1066, Apr. 1987.

[9] R. Li, B. Zeng, and M.L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 4, no.4, pp. 438–442, Aug. 1994.

[10] L. M. Po and W. C. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, no.3, pp. 313–317, Jun. 1996.

[11] 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 Trans. Circuits Syst. Video Technol.*, vol. 3, no. 5, pp. 368-379, Oct. 1993.

[12] F. Dufaux and F. Moscheni, "Motion Estimation Techniques for Digital TV: A Review and a New Contribution," *Proceedings of the IEEE*, vol. 83, no. 6, pp. 858–876, 1995.

[13] M. T. Orchard, "Predictive Motion-Field Segmentation for Image Sequence Coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 3, no. 1, pp. 54–70, 1993.

[14] V. Seferidis and M. Ghanbari, "Generalized Block-Matching Motion Estimation Using Quad-Tree Structured Spatial Decomposition," *IEEE Proc.-Vis. Image Signal Process*, vol. 141, no. 6, pp. 446–452, 1994.

[15] Y-K. Chen and S.Y. Kung, "Rate optimization by true motion estimation," *Proc. of IEEE Workshop on Multimedia Signal Processing*, June 1997, pp. 187-194.

[16] K. Deb, *Multi-objective optimization using evolutionary algorithm.* John Wiley & Sons, ISBN 0-471-87339-X, 2001.

[17] T. Chen, Y. Chen, S. Tsai, S. Chien, and L. Chen, "Fast Algorithm and Architecture Design of Low Power Integer Motion Estimation for H.264/AVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 5, May 2007.

[18] A. Netravalli and J. Robbins, "Motion compensated television coding," *Bell Systems Technical Journal*, no. 3, 1979, pp. 629-668.

[19] J.C. Greiner, R. Sethuraman, J. van Meerbergen, and G. de Haan, "A cost-effective implementaion of object-based motion estimation," *IEEE Workshop on Signal Processing Systems*, 2003.

[20] S.-C. Tai, Y.-R. Chen, Z.-B. Huang, C.-C. Wang, "A Multi-Pass True Motion Estimation Scheme With Motion Vector Propagation for Frame Rate Up-Conversion Applications", *IEEE/OSA J. Disp. Technol.*, Vol. 4, No. 2, pp. 188-197, June 2008

[21] G. de Haan, "Progress in motion estimation for consumer video format conversion," *IEEE Trans. Consum. Electron.*, vol. 46, no. 3, pp. 449-459, Aug. 2000.

[22] J. Astola, P. Haavisto and Y. Neuvo, "Vector median filters," *Proc. IEEE*, vol. 78, pp. 678-689, Apr. 1990.

[23] B.-W. Jeon, G.-I. Lee, S.-H. Lee, and R.-H. Park, "Coarse-to-fine frame interpolation for frame rate up-conversion using pyramid structure," *IEEE Trans. Consum. Electron.*, vol. 49, no.3, pp. 499-508, Aug. 2003.

[24] T.Y. Kuo and C.-C.J. Kuo, "Motion-compensated interpolation for low-bit-rate video quality enhancement," *Proc. SPIE Visual Communications and Image Processing*, vol. 3460, pp. 277-288, July 1998.

[25] A. Kaup and T. Aach, "Efficient prediction of uncovered background in interframe coding using spatial extrapolation," *Proc. ICASSP.*, vol. 5, pp, 501-504, 1994.

[26] R. J. Schutten and G. D. Haan, "Real-time 2–3 pull-down elimination applying motion estimation/compensation in a programmable device," *IEEE Trans. Consum.Electron.*, vol. 44, no. 3, pp. 501–504, Aug. 1998.

[27] B-T. Choi, S-H. Lee, and S-J. Ko, "New frame rate up-conversion using bi-directional motion estimation," *IEEE Trans. Consum. Electron.*, vol. 46, no.3, pp. 603-609, Aug. 2000.

[28] B-D. Choi, J-W. Han, C-S. Kim, and S-J. Ko, "Motion-compensated frame interpolation using bilateral motion estimation and adaptive overlapped block motion compensation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 4, pp. 407-416, Apr. 2007.

[29] S-J. Kang, K-R. Cho, and Y. H. Kim, “Motion compensated frame rate up-conversion using extended bilateral motion estimation,” *IEEE Trans. Consum. Electron.*, vol. 53, no.4, pp. 1759-1767, Nov. 2007.

[30] S-J. Kang, D-G. Yoo, S-K. Lee, and Y. H. Kim, “Multiframe-based bilateral motion estimation with emphasis on stationary caption processing for frame rate up-conversion,” *IEEE Trans. Consum. Electron.*, vol. 54, no.4, pp. 1830-1838, Nov. 2008.

[31] M. Orchard and G. Sullivan, “Overlapped block motion compensation: An estimation-theoretic approach,” *IEEE Trans. Image Processing*, Vol. 3, May 1994, pp. 693–699.

[32] ITU-T, Draft ITU-T Recommendation H.263, “Video Coding for Low Bit Rate Communication,” 1997.

[33] Beric, G. de Haan, R. Sethuraman, and J. van Meerbergen, “An efficient picture-rate up-converter,” *Journal of VLSI Signal Processing*, vol. 41, pp. 49-63, 2005.

[34] S.-C. Tai, Y.-R. Chen, Z.-B. Huang, and C.-C. Wang, “A multi-pass true motion estimation scheme with motion vector propagation for frame rate up-conversion applications,” *IEEE J. Display Technol.*, vol. 4, no. 2, Jun. 2008.

[35] Burak Erbağcı and Özgür Karakaya, “Bilateral Motion Estimation Algorithm and Hardware Design”, *BS Graduation Project Final Report*, Sabancı University, Jun. 2009.

[36] Zafer Tevfik Özcan and Çağla Çakır, “Overlapped Block Motion Compensation Software and Hardware Design”, *BS Graduation Project Final Report*, Sabancı University, Jun. 2009.

[37] R. W. Hall, “Efficient spiral search in bounded spaces,” *IEEE Trans. Pattern Anal. Mach. Intell.* vol. PAMI-4, no. 2, pp. 208-215, Mar 1982.

[38] Video Quality Experts Group Benchmark Videos, ftp://vqeg.its.bldrdoc.gov/