Temporal Denoising of High Resolution Video

Temporal Denoising of High Resolution Video

By

Gang XUE

M.Sc.

A Thesis

Submitted to the School of Graduate Studies

In Partial Fulfilment of the Requirements

for the Degree of

Master of Applied Science

McMaster University

MASTER OF APPLIED SCIENCE (2009) McMaster University

(Electrical and Computer Engineering) Hamilton, Canada

TITLE: Temporal Denoising of High Resolution Video

AUTHOR: Gang XUE, M.Sc. (Fudan University, China)

SUPERVISOR: Dr. Xiaolin Wu

COSUPERVISOR: Dr. Sorina Dumitrescu

NUMBER OF PAGES: x, 59

Abstract

This thesis is concerned with the removal or reduction of noises in high resolution video sequences. Many video denoising techniques have been published in the past two decades, with or without motion compensation. They vary in a wide range of complexity, performance, and implementation cost. Also, many existing video denoisers make simplistic assumptions on noise statistics and motion type, and hence their performance depends on the validity of the assumed noise and motion models. To improve the performance and robustness of existing methods, we propose a new joint spatial-temporal video denoising algorithm that combines multihypothesis interframe motion compensation and directional intra-frame filtering. The algorithm takes into account general compound motions, including both global camera motion and individual object motion(s). An affine motion model is used to characterize the global camera movement, whereas a blockwise translational motion model is used to approximate local object motions. Quadtree data structure is used to organize and speed up the computations of block-based motion estimation. Quadtree-structured diamond search is conducted so that a large area can be examined in motion estimation at a low computational cost.

In order to achieve the best possible visual quality we augment motion-compensated temporal interframe denoising operation by an intra-frame denoising operation of adaptive directional filtering. The directional filter is designed for the local signal waveform and noise level, and it has the advantage of effectively suppressing noises

without blurring edges.

The proposed video denoising algorithm is implemented and tested extensively on high-resolution digital cinema contents. The experimental results demonstrate the competitive advantages of the new algorithm in both visual quality and processing throughput.

Acknowledgement

I would like to take this opportunity to thank and acknowledge the many individuals who have made the completion of this thesis possible.

First and foremost, I would like to express my appreciation to my supervisor Dr. Xiaolin Wu. It is an honor as well as pleasure to work with him. His guidance, encouragement and keen insights are highly appreciated and will always be remembered. I am also grateful to my co-supervisor Dr. Dumitrescu Sorina for her patience and suggestions.

Special thanks goes to my readers, Dr. Jiankang Zhang and Dr. Shahram Shirani for their valuable input. Thanks to Cheryl, Helen and all other Electrical and Computer Engineering administrative staffs for their friendly assistance in the past few years. Sincere thanks goes to my research group Xiaohan, Xiangjun, Zhe, Ning and Nathan at the Multimedia Signal Processing Laboratory. Their help and friendship have made this an experience to remember and cherish.

Finally, to my family, no words can describe the gratefulness I feel for your unconditional love and support. I truly hope that you will be proud of what I have accomplished.

Contents

Abstract iii

Acknowledgement v

List of Figures viii

List of Tables ix

List of Abbreviations x

1 Introduction 1

2 Intra-frame Denoising 7

2.1 Noise Level Estimation 7

2.2 Edge Classification 8

2.3 One-pixel-width Edges 10

2.4 The Directional Filter 11

2.5 Fast Implementation of Directional Median Filter 13

3 Motion Estimation 17

3.1 Fast Global Motion Estimation 18

3.1.1 Model of Global Motion 18

3.1.2 Optimization Problem 19
3.1.3 Extraction of Feature Points 21
3.1.4 Solving the Optimization Problem: Gradient Descent Method 22
3.1.5 Exclusion of Regions With Local Motion 23
3.2 Local Motion Estimation 24
3.2.1 Local Motion Segmentation 24
3.2.2 Multi-Resolution Structure 25
3.2.3 Directional Diamond Search 25
3.2.4 Quad-tree Structure 27
3.2.5 Cost Function 29
3.2.6 Steps of Local Motion Estimation 30
3.2.7 Fault Motion Removal 32

4 Temporal Denoising **33**
4.1 Weighted Temporal Filter 34
4.2 Median Temporal Filter 35

5 Implementation And Experimental Results **37**
5.1 Algorithm Implementation 37
5.2 Experimental Results 38

6 Conclusions **47**

A Fast Implementation of Median Filter **53**
A.1 5-point Median Filter 53
A.2 3-point Median Filter 55

B Fast Implementation of Block-based SAD **57**

List of Figures

1.1 Temporal denoising method 5

2.1 Intra-denosing results of three methods: the median filter, the directional median filter, and our directional median filter with one-pixel-width edge. 14

3.1 The mapping grid of a current pixel in the reference frame 20

3.2 Flow chart for global motion estimation subsystem 24

3.3 Diamond search pattern with searching step = r 28

3.4 Quad-tree pattern 29

4.1 Part of background. 35

5.1 Dark background sequence. 41

5.2 Blow-up part of Swing Angel sequence. 42

5.3 Waving hand sequence. 44

5.4 Detail blow-up of waving hand sequence. 45

5.5 Animation robot sequence. 46

List of Tables

2.1 The total number of instructs *min* and *max* for the N-median filter . 15

2.2 SSE2(R) instructs supports parallelized vector operations *min*() and *max*(). 15

5.1 Descriptions of high resolution test video sequences. 39

5.2 Processing time comparison between proposed method and Neat Video Denoiser (NVD). 39

List of Abbreviations

i.i.d.	independent identically distributed
DS	Diamond Search
RGB	Red, Green, and Blue Color Channels
SAD	the Sum of Absolute Difference
SIMD	Single Instruction Multiple Data
SNR	Signal-to-noise Ratio

Chapter 1

Introduction

Video denoising plays an important role in video processing. It is critical, since the presence of noise in a video sequence degrades both its visual quality and the effectiveness of subsequent processing tasks. For example, the compression performance of a video codec decreases in the presence of noises. This is because noises can greatly increase the entropy of the contaminated video sequence. Therefore, video denoising can improve not only the visual quality but also the performance of subsequent processing tasks such as analysis, coding and interpretation.

A video sequence can be viewed as a three-dimensional data set, i.e., a temporal sequence of 2D frames. The noise degradation model of this thesis can be described by

$$I(x, y, t) = R(x, y, t) + n(x, y, t) \tag{1.1}$$

where $I(x, y, t)$ and $R(x, y, t)$ are the observed and the original pixel values, respectively, of the t^{th} frame at spatial location (x, y), and $n(x, y, t)$ is independent additive zero-mean white Gaussian noise.

The video denoising problem is to find an estimate $\tilde{R}(i, j, t)$ of the true video sequence $R(x, y, t)$ based on the noisy observation $I(x, y, t)$. This problem belongs to

the class of *Inverse Problems* in image/video processing, which aims to recover a high quality image/video signal from a degraded version of it. A video denoising algorithm can operate in both spatial and temporal dimensions, and remove noises by exploiting sample correlations within a frame as well as between neighboring frames. Temporal 3D denoising methods are more effective than spatial 2D denoising methods, but at a significantly higher computational cost.

In this thesis we focus on temporal denoising of high-resolution video sequences, such as those of digital cinema. The problem becomes more challenging for high-resolution videos due to the following difficulties.

1. High resolution video is more noisy than low resolution video because the signal to noise ratio decreases in pixel size. The problem is particularly acute when a high-resolution video camera operates in low luminance conditions.

2. High-resolution video is meant to capture fine high-frequency details in a scene. For a video denoiser detail preservation and noise reduction are a pair of conflicting requirements, and this conflict is more difficult to be resolved as spatial resolution increases.

3. The computational complexity of video denoising increases quadratically in spatial resolution.

The existing video denoising algorithms can be classified into two classes: spatial filtering methods and temporal filtering methods. In spatial methods [16] each frame is filtered individually, ignoring temporal correlations between video frames. This intraframe approach tends to introduce artifacts into the filtered video sequence due to temporal inconsistency. Even the most advanced spatial denoisers, such as Wiener [26] and wavelet filtering [27], can not deliver good video denoising results. Therefore, pure spatial denoisers methods are not appropriate for video.

Temporal video denoising methods [5], [6] can alleviate the artifacts caused by spatial methods by tracking object motions through frames and thus ensuring temporal consistency. The temporal denoising approach exploits both spatial and temporal correlations. By registering a patch of noisy pixels of the current frame are registered with its counterparts in reference frames via motion estimation, one can perform 3D low-pass filtering to suppress noises. A spatiotemporal joint filtering scheme (JNT) was proposed in [20]. The noisy video was first filtered by a temporal Kalman filter and a spatial Wiener filter separately, and then the two denoising results were combined to improve the video quality further. However, in case of complex motions the pixel registration is error prone and consequently the temporal denoising methods are susceptible to oversmoothing of the video frames.

There are two types of temporal denoising techniques: motion adaptive and motion registered. Motion adaptive methods only check whether a pixel patch is static between two frames. If there is lack of motion, a temporal low-pass filter is used; otherwise, a spatial denoising filter is used. The advantage of motion adaptive methods is their low computational complexity, while the disadvantage is that they can not make full use of temporal correlations. Motion registration (MR) methods estimate object and/or camera motions between the current frame and reference frames. The motion registered blocks of pixels are then low-pass filtered through the estimated motion trajectories. This approach incurs higher computational cost but it produces better results, if motions can be estimated accurately.

The goal of this research is to develop a fast motion-registered denoising method. To fully utilize inter-frame correlations we track the motion through multiple reference frames. This generates multiple estimates of a noisy pixel in the current frame. These estimates and the current noisy observation are fused by a least square weighting scheme. Compared to denoising techniques relying on a single reference frame [17]

[18] [19] [20], our multiframe motion estimation technique makes the denoising results more robust, cleaner and sharper. To improve motion estimation precision and speed up motion search, a general motion model is adopted that allows both affine camera motion and translational object motions.

To estimate the true clean pixel value $R(x, y, t)$ in (1.1), we adopt a joint spatial-temporal denoising approach of the following form:

$$\begin{aligned}\tilde{R}(i,j,t) = \mathbf{T}(&\mathbf{S}(I(i_{t-2}, j_{t-2}, t-2)), \mathbf{S}(I(i_{t-1}, j_{t-1}, t-1)), \mathbf{S}(I(i,j,t)),\\ &\mathbf{S}(I(i_{t+1}, j_{t+1}, t+1)), \mathbf{S}(I(i_{t+2}, j_{t+2}, t+2)))\end{aligned} \tag{1.2}$$

where $\mathbf{S}(\cdot)$ is a spatial denoiser, $\mathbf{T}(\cdot)$ is a temporal denoiser, and (i_{t+k}, j_{t+k}), for $k = \pm 1, \pm 2$ is the pixel location of frame $t+k$ that is motion registered with location (i, j) of the current frame. $I(i, j, t)$ denotes the noisy pixel value of location (i, j) of frame t.

Figure1.1 is a schematic description of our spatiotemporal denoising algorithm. The algorithm consists of two phases: spatial and temporal denoising processes. In the spatial denoising phase, the noise energy level of each frame is first estimated. Then the frame is denoised by a low-pass directional filter whose parameters are tuned to the estimated noise level and to the gradient of the current pixel. The resulting sequence of spatially denoised frames are further processed in the temporal denoising phase. The goal is to improve the results of the spatial denoiser by exploiting interframe correlations. This is achieved by registering multiple frames via motion estimation and low-pass filtering of the current pixel along motion trajectory. For best possible visual quality both global camera motion and local object motions are considered.

This thesis is organized as follows. In Chapter 2, we develop an efficient intra-frame denoising algorithm and describe its implementation. Chapter 3 details fast algorithms for global motion estimation and block-based local motion estimation. The

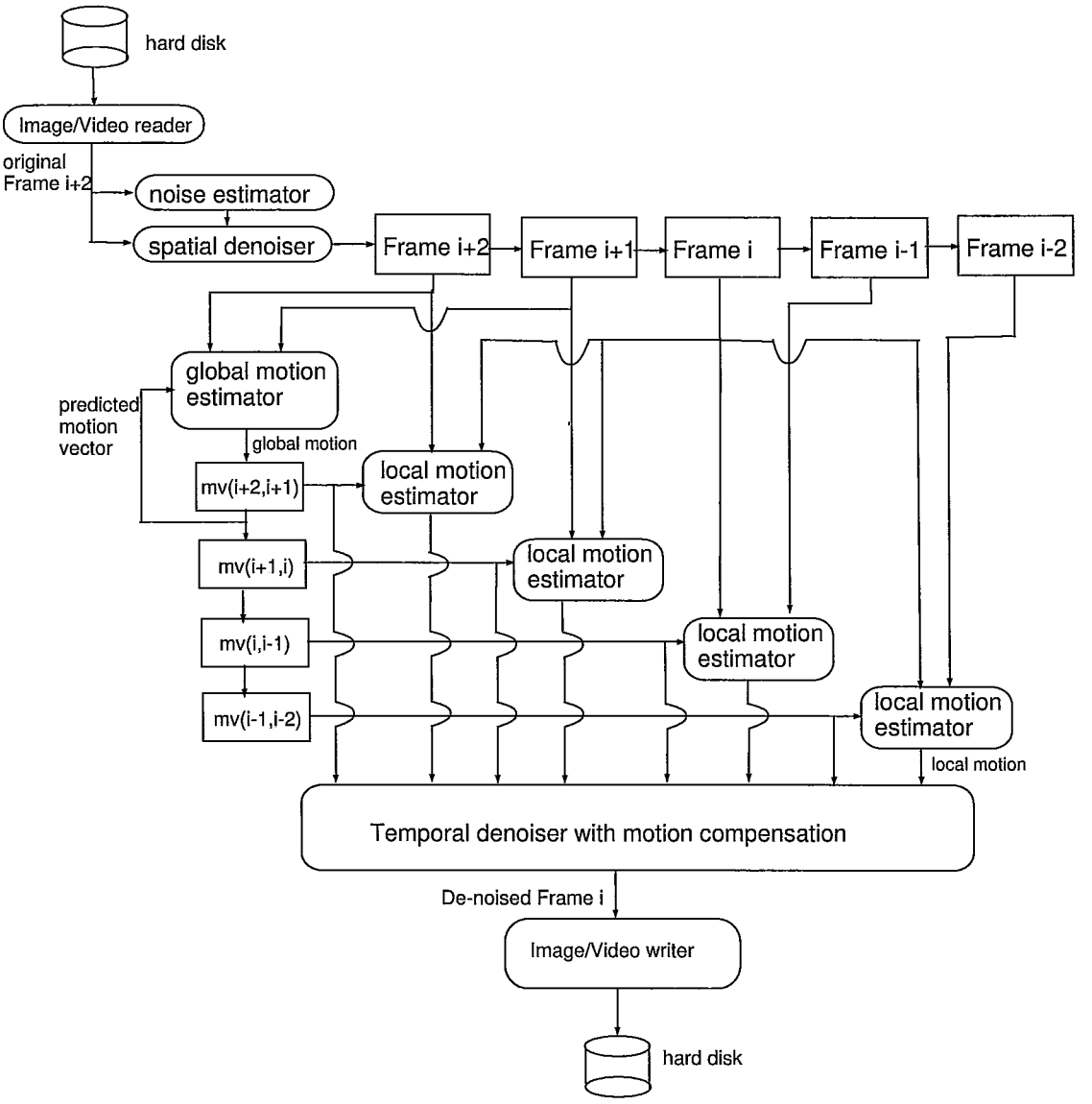


Figure 1.1: Temporal denoising method

temporal denoising algorithm is introduced in Chapter 4. We present all experimental results in Chapter 5, and finally conclude the thesis in Chapter 6.

Chapter 2

Intra-frame Denoising

In this chapter, we present the details of the spatial denoising phase of the proposed spatiotemporal video denoising system. The main result is an adaptive directional intra-frame denoiser. The purpose of intra-frame denoising is to facilitate the subsequence temporal denoising process by alleviating the effects of noises on motion estimation.

2.1 Noise Level Estimation

In the design of any denoiser, it is important to have a statistical model of the noise. In this thesis, we assume that the noise in a video frame is additive white Gaussian of zero mean. Therefore, our first task is to estimate the variance (i.e., the energy level) of the noises. The estimated noise variance is an important parameter when designing our adaptive directional denoiser. For instance, it can set an adaptive threshold for edge points amid noises.

For motion estimation, the estimated noise level can help the searching iterations converge fast, and stop when the residual block consists of only noise, which means the matching is done, and no further searching is required.

According to the spatial local continuous characteristics of the frame content, we fit the pixels in each 16 × 16 region with a bi-cubic function and estimate the noise using the residual.

$$\sigma = \min\{\sigma_x, \sigma_y\}; \tag{2.1}$$

$$\sigma_x = \frac{1}{|\Omega|} \sum_{(i,j)\in\Omega} |I_{i,j} - \bar{I}^x_{i,j}|; \tag{2.2}$$

$$\sigma_y = \frac{1}{|\Omega|} \sum_{(i,j)\in\Omega} |I_{i,j} - \bar{I}^y_{i,j}|; \tag{2.3}$$

$$\bar{I}^x_{i,j} = \frac{1}{16}(-1 \quad 9 \quad 9 \quad -1)(I_{i,j-2} \quad I_{i,j-1} \quad I_{i,j+1} \quad I_{i,j+2})^T; \tag{2.4}$$

$$\bar{I}^y_{i,j} = \frac{1}{16}(-1 \quad 9 \quad 9 \quad -1)(I_{i-2,j} \quad I_{i-1,j} \quad I_{i+1,j} \quad I_{i+2,j})^T; \tag{2.5}$$

where $\bar{I}^x_{i,j}$ is the estimated value of pixel (i, j) by cubic fitting in horizontal direction, $\bar{I}^y_{i,j}$ is the estimated value of pixel (i, j) by cubic fitting in the vertical direction, Ω is a 16 × 16 block, and σ represents the estimated noise level. The notation $\min\{a, b\}$ is for the minimum of a and b.

The noise level estimation can be implemented very fast, because bi-cubic fitting is linear, and can be computed parallelly. We can assume that the noise level is stationary in neighbor video frames. Therefore, the noise level estimation only need to be applied once for a series of frames.

2.2 Edge Classification

Edge Classification is required before applying the directional filter. It can be shown that under rather general assumptions for an image formation model, discontinuities in image intensity are likely to correspond to:

- discontinuities in depth,
- discontinuities in surface orientation,
- changes in material properties and
- variations in scene illumination.

Our edge classification relies on the computation of image gradients. The computation of gradients is sensitive to the noise. Therefore, we apply a Gaussian low-pass filter on the image to reduce noise sensitivity, and then compute the gradients based on the low-passed image. These two procedures can be integrated in the following gradient operators (2.6).

$$\begin{aligned} \Delta h_{i,j} &= \frac{1}{4}(-1 \quad -2 \quad 2 \quad 1)(I_{i,j-2} \quad I_{i,j-1} \quad I_{i,j+1} \quad I_{i,j+2})^T; \\ \Delta v_{i,j} &= \frac{1}{4}(-1 \quad -2 \quad 2 \quad 1)(I_{i-2,j} \quad I_{i-1,j} \quad I_{i+1,j} \quad I_{i+2,j})^T; \end{aligned} \tag{2.6}$$

where $\Delta h_{i,j}$ and $\Delta v_{i,j}$ denote the horizontal and vertical gradient at location (i, j) respectively.

Given such estimates of first-order derivatives, the gradient magnitude is then estimated as

$$d_{i,j} = \max\{|\Delta h_{i,j}|, |\Delta v_{i,j}|\} \tag{2.7}$$

Once we have computed a measure of edge strength $d_{i,j}$, the next stage is to apply a threshold, to decide whether edges are present or not at an image point. The lower the threshold, the more edges will be detected. Thus the result will be increasingly susceptible to noise, and also to picking out irrelevant features from the image. Conversely, a high threshold may miss subtle edges, or result in fragmented edges. We set the threshold based on noise level σ. Therefore, the decision if a pixel

is on an edge is taken as follows.

$$\begin{aligned} &\texttt{if } d_{i,j} > \lambda\sigma, \quad (i,j) \texttt{ is on an edge;} \\ &\texttt{otherwise,} \quad (i,j) \texttt{ is not on an edge.} \end{aligned} \tag{2.8}$$

where $\lambda > 1$ is a thresholding parameter, and σ is the noise level estimated in (2.1). We can choose an appropriate thresholding parameter λ, and a suitable thresholding values may vary over the image.

Further, we estimate the gradient direction using the first-order derivatives, then rounding off the gradient direction to multiples of 45 degrees. Let $\tau = \frac{\Delta v_{i,j}}{\Delta h_{i,j}}$, then all edges are classified into four different angles, $0°, 45°, 90°$, and $135°$ as follows.

$$\begin{cases} 0° & \texttt{if } \tau \in [-\tan\frac{\pi}{8}, \tan\frac{\pi}{8}]; \\ 135° & \texttt{if } \tau \in (\tan\frac{\pi}{8}, \tan\frac{3\pi}{8}]; \\ 45° & \texttt{if } \tau \in [-\tan\frac{3\pi}{8}, -\tan\frac{\pi}{8}); \\ 90° & \texttt{otherwise .} \end{cases} \tag{2.9}$$

2.3 One-pixel-width Edges

From (2.6) and (2.9), we notice that the first-order derivatives can not correctly estimate the edge direction for one-pixel-width edges. Because both of horizontal and vertical gradients are zeros on the peak of one-pixel-width edges, a non-directional filter will be applied on them, which causes an oversmoothing effect.

We can recognize this one-pixel-width edges, and classify them into two classes by the following method.

Let

$$\begin{aligned} a_{i,j} &= \frac{1}{2}\big|\,|2I_{i,j} - (I_{i-1,j-1} + I_{i+1,j+1})| - |2I_{i,j} - (I_{i+1,j-1} + I_{i-1,j+1})|\,\big|, \\ b_{i,j} &= \frac{1}{2}\big|\,|2I_{i,j} - (I_{i-1,j} + I_{i+1,j})| - |2I_{i,j} - (I_{i,j-1} + I_{i,j+1})|\,\big|. \end{aligned} \tag{2.10}$$

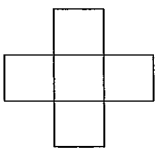
Then

$$\begin{cases} \texttt{if } a_{i,j} > \lambda\sigma \quad (i,j), \texttt{ is on a diagonal one-pixel-width edge;} \\ \texttt{if } b_{i,j} > \lambda\sigma \quad (i,j), \texttt{ is on a horizontal or vertical one-pixel-width edge;} \\ \texttt{otherwise} \quad (i,j) \texttt{ is not on an one-pixel-width edge.} \end{cases} \tag{2.11}$$

2.4 The Directional Filter

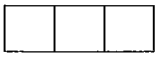
Once we know the direction of the edge, we can apply the directional filter, including the following cases. In frame I, let $I_{i,j}$ be the intensity of pixel (i,j), and $\hat{I}_{i,j}$ be the filtered value. We have

- No edge:



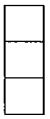
$$\hat{I}_{i,j} = \texttt{MEDIAN}\begin{pmatrix} & I_{i-1,j} & \\ I_{i,j-1} & I_{i,j} & I_{i,j+1} \\ & I_{i+1,j} & \end{pmatrix} \tag{2.12}$$

- Horizontal edges (0°):



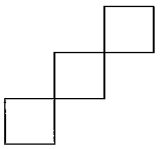
$$\hat{I}_{i,j} = \texttt{MEDIAN}(I_{i,j-1},\ I_{i,j},\ I_{i,j+1}) \tag{2.13}$$

- Vertical edges (90°):



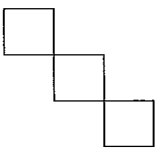
$$\hat{I}_{i,j} = \texttt{MEDIAN}(I_{i-1,j},\ I_{i,j},\ I_{i+1,j}) \tag{2.14}$$

- 45° diagonal edges:



$$\hat{I}_{i,j} = \texttt{MEDIAN}(I_{i-1,j+1},\ I_{i,j},\ I_{i+1,j-1}) \tag{2.15}$$

- 135° diagonal edges:



$$\hat{I}_{i,j} = \texttt{MEDIAN}(I_{i-1,j-1},\ I_{i,j},\ I_{i+1,j+1}) \tag{2.16}$$

- One-pixel-width edges:

  Copy the original intensity.

$$\hat{I}_{i,j} = I_{i,j}. \tag{2.17}$$

where $\texttt{MEDIAN}(\cdot)$ is the operation to compute the median.

Generally, the median filter performs a good noise reduction without blurring edges. In case of one-pixel-width edges, the median filter will lose the sharpness,

because the gradient value is underestimated and the leak value on one-pixel-width edges will be lost by median operation. Therefore, we keep the original pixel values for one-pixel-width edges. We compared three different filters: non-directional median filter, directional median filter, and our directional median filter with one-pixel-width edges. The visual results are shown in Figure 2.1.

Analysis: the non-directional median filter(Figure 2.1(b)) introduces oversmoothing effect around the hair. The directional median filter(Figure 2.1(c)) provides more clear and sharper hair than the non-directional median filer. Among the three filters, our method (Figure 2.1(d)) gives the most sharp single hair, and keeps the highlight of the thin hair, which is very sensitive to human eyes.

2.5 Fast Implementation of Directional Median Filter

The directional median filter can be implemented by two stages:

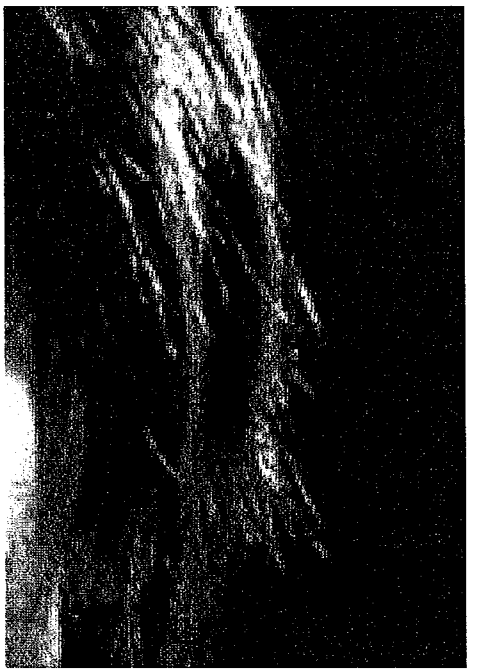
1. Denoise frame I by a 5-point median filter to obtain I_t;
2. Re-examine frame I, classify edges, choose directional filter for edges and replace the previous ones in I_t.

To improve the processing speed of the median filter, we minimize the number of comparisons and memory accesses (reading and writing memory), and establish parallel computing technique.

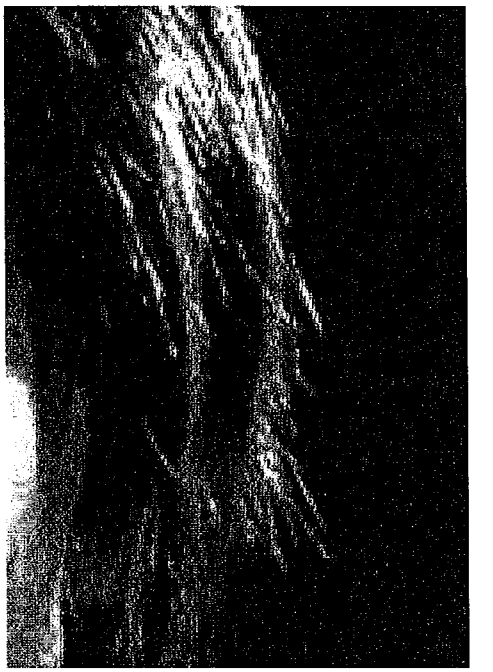
John L. Smith [2] implemented the 9-point median filter with the minimum exchange network to produce a median from nine input pixels by the FPGA. Inspired by his hardware implementation, we develop a fast software implementation using the fundamental operations "min" and "max". The minimum number of comparisons

(a) The original image

(b) Result of the median filter



(c) Result of the directional median filter



(d) Result of our directional median filter

Figure 2.1: Intra-denosing results of three methods: the median filter, the directional median filter, and our directional median filter with one-pixel-width edge.

Table 2.1: The total number of instructs *min* and *max* for the N-median filter

N	3	5	7
number of *min*() and *max*()	4	10	20

Table 2.2: SSE2(R) instructs supports parallelized vector operations *min*() and *max*().

content in a register	*min*()	*max*()	Latency (cycles)
16 8-bit values	PMINSB	PMAXSB	2
8 16-bit values	PMINSW	PMAXSW	2
4 32-bit values	PMINSD	PMAXSD	2

("min" and "max") is listed in Table.2.1. For example, a 5-point median filter needs at least ten comparisons among five input elements to find the median.

In order to make full use of parallel computing ability of the CPU, we employ SSE2 (Streaming SIMD Extensions 2) instruction set. SSE2 is a Single Instruction Multiple Data (SIMD) instruction set designed by Intel. It enables the parallel computing for vectors and can operate on 128-bit registers, which accelerates the computing speed significantly. The SSE2 instruction set provides operations $min(a, b)$ and $max(a, b)$, both of which are in high parallelism and have a latency of only two cycles on a Pentium 4 CPU (See Table 2.2). In Table.2.2, we list the SSE2 instructions for *min* and *max*. Instructs PMINSB and PMAXSB compute parallel minimum and maximum between two sets of sixteen 8-bit signed values. Instructs PMINSW and PMAXSW compute parallel minimum and maximum between two sets of eight 16-bit signed values. Instructs PMINSD and PMAXSD compute parallel minimum and maximum between two sets of four 32-bit signed values. The detailed operations of

$min(a, b)$ and $max(a, b)$ are described in (2.18).

$$\begin{aligned} &\min(a, b) = \{\min(a_i, b_i)\} \\ &\max(a, b) = \{\max(a_i, b_i)\} \\ &\texttt{for } i = 1, 2, \cdots, N. \end{aligned} \tag{2.18}$$

For example, one SSE2 instruction, PMINSW, computes the scalar minimums of two sets of eight 16-bit signed values within two clock cycles. SSE2 only supports the comparison operation for signed values. Thus, a conversion from unsigned to signed values is required, since we input 16-bit unsigned pixel values.

Moreover, our median filter has the minimized number of memory accesses. During the computation of the median, our implementation sequentially reads input data from the memory to a register only once, sequentially writes the output data from a register to the memory once, and all other operations are manipulated inside registers, which significantly improves CPU's computational efficiency and memory cache efficiency.

For a 5-point median filter, we compared our optimized code against the non-optimized code. They are applied on a frame with resolution 2048×1556, RGB 48-bit depth. Our optimized code spends 0.08 seconds, while the non-optimized code spends 1.34 seconds, about sixteen times longer than our optimized code. We conclude that the parallel computing technology improves the computational efficiency significantly.

Chapter 3

Motion Estimation

In this chapter, we describe our motion estimation approach with two stages: global motion and local motion estimation. Motion estimation is the process of determining the motion vectors that describe the transformation from one 2D image to another; usually from adjacent frames in a video sequence. It is an ill-posed problem as the motion is in three dimensions but the images are a projection of the 3D scene onto a 2D plane. The motion vectors may relate to the whole image (global motion estimation) or specific parts, such as rectangular blocks, arbitrarily shaped patches or even per pixel. The motion vectors may be represented by a translational model or many other models that can approximate the motion of a real video camera, such as rotation and translation in all three dimensions and zoom.

The motion existing in a scene can be mainly considered as arising from local motions superimposed to the camera motion (ie., the global motion). The former is produced by the movement of objects in the scene, and the latter results from the motion of camera. The global motion can be removed to provide a more precise subsequent local motion evaluation, if the camera motion is estimated. The foreground consists of objects in local motion, and the background consists of objects in global motion. The global motion should be estimated only on the background, prevent-

ing from the error caused by local motions. The segmentation of background and foreground contributes to the accuracy of global motion.

The two stages motion estimation approach is presented as follows. In the first stage, we obtain the background mask and its initial guess for global motion parameters either by first estimation applied on edges in frames or by tracking and prediction of the previous mask and parameters. The global motion is then applied on the background. Next, the global motion compensation is performed, and the background mask is refined by eliminating more foreground regions from the previous background mask. Then, the global motion estimation and background refinement are done iteratively, until no more foreground can be found in the mask. In the further stage, the local motion estimation can be performed on the foreground regions.

3.1 Fast Global Motion Estimation

In this section, we describe the detail of global motion estimation which uses the generalized motion estimation [1].

3.1.1 Model of Global Motion

Global motion is considered a kind of motion caused by the camera. To describe this complex motion, a perspective model is introduced.

$$\begin{aligned} x_2 &= \frac{a_0 + a_2 x_1 + a_3 y_1}{a_6 x_1 + a_7 y_1 + 1} \\ y_2 &= \frac{a_1 + a_4 x_1 + a_5 y_1}{a_6 x_1 + a_7 y_1 + 1} \end{aligned} \tag{3.1}$$

where $a_0, \cdots, a_7$ are the motion parameters, (x_1, y_1) denotes the spatial coordinates of a pixel in the current frame, and (x_2, y_2) denotes the coordinates of corresponding pixel in the reference frame.

In most cases, global camera motion in movies can be simplified to an affine model, which corresponds to setting $a_6 = a_7 = 0$ in formula (3.1).

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} a_2 & a_3 \\ a_4 & a_5 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} \tag{3.2}$$

In this thesis, we consider that the camera motion obeys an affine model.

3.1.2 Optimization Problem

The global motion estimation is designed to achieve the minimum of the sum of squared differences between the current frame I and the motion compensated reference frame I_k.

$$\min_{\mathbf{a}} \sum_{(x,y)\in I} [I(x,y) - Int(I', f(x,y,\mathbf{a}), \mathbf{a})]^2 \tag{3.3}$$

where $\mathbf{a} = (a_0, \cdots, a_5)$ denotes the affine parameter vector, $f(x, y, \mathbf{a})$ is the coordinates of the corresponding pixel in reference frame I' by the affine transformation (3.2) using $\mathbf{a}$, and the function $Int(\cdot)$ estimates the value of pixel (x, y) based on the motion parameter $\mathbf{a}$ and the reference frame I'. In our work, we choose area-based interpolation [1]. This interpolation method is optimal if box point spread function (PSF) and no correlation between neighboring pixels are assumed.

In order to reduce the sensitivity to noise, we only choose pixels near edges as feature points. Thus, we find edges by the gradient magnitude in the current frame I.

There is a problem that edges with higher gradient will take greater weight in the cost function (3.3), but it effects the accuracy of the motion estimation. Therefore,

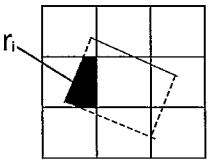


Figure 3.1: The mapping grid of a current pixel in the reference frame

we normalize the weight of all extracted edges, and thus, (3.3) is adjusted to (3.4).

$$\min_{\mathbf{a}} F(\mathbf{a}) = \sum_{(x,y)\in E} \frac{[I(x,y) - Int(I', f(x,y,\mathbf{a}), \mathbf{a})]^2}{\sigma_{x,y}}, \tag{3.4}$$

where $\sigma_{x,y}$ denotes the gradient magnitude of pixel (x, y), and E represents the set of pixels near edges. $\mathbf{a} = (a_0, \cdots, a_5)$ denotes the affine parameter vector, $f(x, y, \mathbf{a})$ is the coordinates of the corresponding pixel in reference frame I' by the affine transformation (3.2) using $\mathbf{a}$, and the function $Int(\cdot)$ estimates the value of pixel (x, y) based on the motion parameter $\mathbf{a}$ and the reference frame I'.

The other benefit of (3.4) is that we reduce the weight of difference caused by noise, since the gradient intensity is proportional to the noise.

The estimation of intensity value of current pixel is illustrated in Figure 3.1. Our estimation function $Int(\cdot)$ can be described as follows.

$$\begin{aligned} Int(\cdot) &= \frac{1}{R}\sum_i r_i I'(i) \\ R &= \sum_i r_i \end{aligned} \tag{3.5}$$

where R is the area of the pixel of the current frame on the grid of the reference frame I', r_i is the area of intersection of current grid and reference grids, and $I'(i)$ denotes the intensity value of pixel i in frame I'.

3.1.3 Extraction of Feature Points

The global motion estimation only relies on the background information. In the background, smooth areas do not have discrimination power in motion estimation, due to lack of features. Consequently, those areas without features can be safely disregarded in the motion estimation without loss of precision. This significantly reduces the computational complexity. In this work the feature selection is performed only in the current frame. Motion estimation is done by matching selected pixels (with features) of the current frame with pixels of the reference frames. We extract large-scale edges as good reliable features. The edge map is subject to an erosion operation to exclude isolated pixels selected by the high-pass edge detector because there is a high possibility that the corresponding features have noise origin. This is followed by a dilation step to allow pixels adjacent to the edges to participate in motion estimation. The last step is intended to make the algorithm robust against inaccurate edge detection. For video frames with negligible noise the edge detection step can be replaced by inexpensive high-pass filtering and thresholding. We found that less than 10 percent of the total number of pixels can be selected for very accurate registration of video sequences. The matching technique is applied only on the pixels near the edges, which reduces significantly the computational cost.

We can compute the gradients to find the set E of feature points in the current frame I .

$$E = \{(x, y) | (\frac{\partial I}{\partial x})^2 + (\frac{\partial I}{\partial y})^2 > \eta\} \tag{3.6}$$

where η is the threshold of edge detection.

3.1.4 Solving the Optimization Problem: Gradient Descent Method

Generally, the cost function (3.4) is non-convex. However, if the initial guess is close enough to the global minimum, a gradient descent algorithm can provide a numerical solution. Since the motion in a scene is continuous, the previous motion between frames I_{i-1} and I_i can be considered a good initial guess for the motion between I_i and I_{i+1}. In this thesis a modified Newton-Raphson method is applied for solving the optimization problem of (3.4). If $\nabla_{\mathbf{a}} F$ is the gradient of the cost function w.r.t. motion parameter vector $\mathbf{a}$ and $\nabla^2_{\mathbf{a}} F$ is the corresponding Hessian, then one iteration of the modified Newton-Raphson algorithm is given by

$$\mathbf{a}_{k+1} = \mathbf{a}_k - \alpha_{k+1} \nabla^2_{\mathbf{a}_k} F^{-1} \nabla_{\mathbf{a}} F \tag{3.7}$$

where $\mathbf{a}_{k+1}$ and $\mathbf{a}_k$ are the estimated motion vectors in the $(k+1)^{th}$ and k^{th} iteration, respectively, and α_{k+1} is a constant such that $0 < \alpha_{k+1} \leq 1$. The algorithm starts with an initial estimate $\mathbf{a}_0$ obtained from previous affine parameter. The following steps are required in each iteration.

1. Initialization: Compute $F(\mathbf{a}_0)$, $\nabla F(\mathbf{a}_0)$ and $\nabla^2 F(\mathbf{a}_0)$. Set $k = 0$ and $\gamma = 1$.

2. Compute

$$\hat{\mathbf{a}} = \mathbf{a}_k - \gamma \nabla^2 F(\mathbf{a}_k)^{-1} \nabla F(\mathbf{a}_k).$$

   Reduce γ, if $\mathbf{a}$ violates any bound on parameters and re-compute $\hat{\mathbf{a}}$.

3. If the absolute value of $(\mathbf{a}_k - \hat{\mathbf{a}})$ is too small for all parameters, set $\mathbf{a}_n$ as the solution. Exit function.

4. If $F(\hat{\mathbf{a}}) < F(\mathbf{a}_k)$, set $\mathbf{a}_{k+1} = \hat{\mathbf{a}}$ and $\alpha_{k+1} = \gamma$. Goto step 8.

5. Let $\gamma = \frac{1}{4}\gamma$, Goto step 3.

6. If $|F(\mathbf{a}_{k+1}) - F(\mathbf{a}_k)| < \zeta_1$ and $|\nabla^2 F(\mathbf{a}_k)| < \zeta_2$, $\mathbf{a}_{k+1}$ is the solution. Exit function.

7. $k = k + 1$. If $k > MaxIteration$, $\mathbf{a}_{k+1}$ is the solution, and Exit function.

8. $\gamma = \min(1, 2\gamma)$, and goto step 3.

We use the thresholds $\zeta_1 = 0.0001$ and $\zeta_2 = 0.001$. The algorithm converges within less than eight function evaluations ($k < 8$) for all video sequences in our experiment.

Next, we exclude the local motion regions, and do the estimation again, until no more local motion region can be found.

3.1.5 Exclusion of Regions With Local Motion

Because the extracted feature points P may contain edges with both local motion and global motion, feature points in local motion segments will degrade the accuracy of global motion estimation. It is necessary to exclude the local motion segments from feature points P. We check the normalized difference between aligned pixels, and find feature points in the local motion segments L by thresholding the normalized difference as follows.

$$L = \{\mathbf{i} | \left| \frac{I(\mathbf{i}) - I_k(G(\mathbf{i}))}{Grad(\mathbf{i})} \right| > \varepsilon, \mathbf{i} \in P\}, \tag{3.8}$$

where L denotes the feature points located in the local motion segments, the notation $Grad(\cdot)$ computes the gradient intensity, $G(\mathbf{i})$ represents the motion vector of pixel $\mathbf{i}$, I and I_k denote the current frame and the reference frame respectively, and $\varepsilon > 0$ is a threshold parameter.

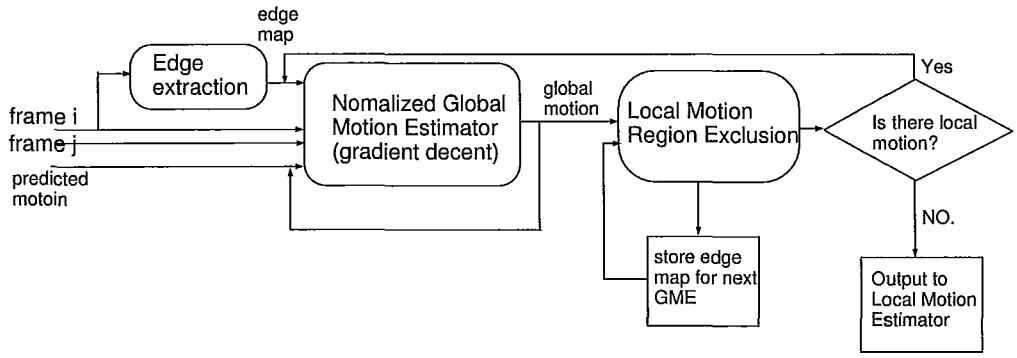


Figure 3.2: Flow chart for global motion estimation subsystem

After excluding segments in local motions, we solve the optimization problem (3.4) again to improve the global motion estimation. We repeat the iteration until no segment in local motions can be excluded.

The whole procedure of our recursive global motion estimation is illustrated in Figure 3.2. In the work flow, the last 4 global motion vectors are required to be stored for the subsequent processing.

3.2 Local Motion Estimation

Different from global camera motion, local object motions can be approximately considered as blockwise translation motions. This type of motions is well estimated by variable block-based motion estimation.

3.2.1 Local Motion Segmentation

Once the global motion is estimated, we segment the frame into regions with either global motion or local motion. First, we transform the reference frame into a compensated frame by using the estimated global motion. Second, we analyze the difference between the current frame and the compensated frame. It is also very important to

reduce the effect of noise. At last, we obtain a map indicating the segments with global motion and with local motion.

$$\mathbb{L} = \{(x,y) | \frac{\sum_{(i,j)\in R(x,y)} |I(i,j) - I_c(i,j)|}{Grad(x,y)} > \varepsilon\} \tag{3.9}$$

where $\mathbb{L}$ is the set of local motion regions, (x,y) denotes the coordinates of a pixel, I is current frame, I_c is the motion compensated frame, ε is a positive constant, $R(x,y)$ is a set of neighborhoods of (x,y), and $Grad(x,y)$ represents the gradient at (x,y).

3.2.2 Multi-Resolution Structure

To further improve the robustness and efficiency, a hierarchical scheme is implemented. A three-level pyramid of the image is built by using a three-tap filter with coefficients $[1/4, 1/2, 1/4]$. The time complexity of block-based motion estimation is $O(whM_xM_y)$, where w, h indicate respectively the width and height of a block, and M_x, M_y denote the maximum motion along x and y direction respectively. In the case of high resolution, such as 4096×2048, M_x, M_y are about 100 for slow motion, and larger than 250 for fast motion. The computational cost is unacceptable if exhausted motion search is performed on the original resolution. By using the pyramid structure, we can do motion search with large range for the top of the pyramid, e.g. lowest resolution, then in other resolutions, we do motion search with small range constrained near the motion vector obtained from the previous lower resolution. The computation is considerably reduced without precision loss.

3.2.3 Directional Diamond Search

The Diamond Search algorithm (DS), proposed by Shan Zhu and Kai-Kuang Ma [11], is based on the study of motion vector field distributions of a large population

of video sequences. Diamond search (i.e., other than exhaustive search) reduces the number (or the complexity) of "sum of absolute difference" (SAD) computations. The diamond pattern as presented in Figure 3.3 is derived from the probability distribution function as those locations corresponding to the highest probability of finding the matching block in the reference frame. The algorithm starts in the co-located macro-block in the reference frame and performs eight additional SAD's around the diamond center. This contrasts to full search (FS) which computes all possible overlapping SAD's. Once the minimum SAD location is found, the diamond center is displaced to the optimum location and a new diamond search is executed. The new search will require fewer SAD computations. The search stops once the position of the minimum SAD is located in the center of the diamond. On average the best matching block will be found within a few diamond search iterations, thus requiring fewer SAD's per macro-block than FS. The algorithm offers the advantage of extending the search support area, allowing more reference frame coverage with fewer computations. However, due to the sparse nature of the diamond, one may miss an optimum matched block near the center. Various algorithms based on DS have been proposed to do a hierarchical search, so that once the best matching macro-block is found an inner diamond search is performed to cover points interior to the diamond.

Furthermore, we employ dynamic initial searching step to increase the speed of searching the minimum SAD as follows.

$$\begin{aligned} s_0 &= 2^{\mu}; \\ s_{i+1} &= s_i/2; \\ \mu &= \min(\lceil \frac{SAD_0}{\phi \sigma A_b} \rceil, 4). \end{aligned} \tag{3.10}$$

where ϕ is a constant, A_b denotes the area of the block b, SAD_0 denotes the SAD at the center of first diamond, and μ denotes the exponent part of initial searching step.

The steps of the algorithm are presented as follows.

1. Input: frame f_i, f_j, block b, predicted motion vector v_0;
2. Compute $SAD_0 = SAD(v_0)$ for block b.
3. Compute $(s_0, s_1, \cdots)$ by formula (3.10).
4. Set $k = 0$, $r = s_0$, $v = v_0$.
5. Search the local minimum SAD in Diamond pattern with searching step r centered at v, and motion vector $\hat{v}$ is obtained.
6. We draw a conclusion:

   If $r == 1$, $\hat{v}$ is the solution, return $\hat{v}$. Done.

   If $||\hat{v}|| > MaxRange$, motion vector can not be obtained in the searching range, and return failure.

   If $v == \hat{v}$, set $k = k + 1$, $r = s_k$, $v = \hat{v}$, and goto step 5.

   Otherwise, set $v = \hat{v}$, and goto step 5.

   where $MaxRange$ denotes the maximum dynamic range we handle. We set $MaxRange = (300, 150)$, which means that the max dynamic range is $[-300, +300]$ at X-axis, and $[-150, +150]$ at Y-axis.

3.2.4 Quad-tree Structure

Constant size block-based motion estimation is insufficient to describe complex motion, such as rotation and zooming. On one hand, the true motion can be simulated more accurately by smaller blocks. On the other hand, evaluating motion using smaller blocks results in more local minima without sufficient constraints.

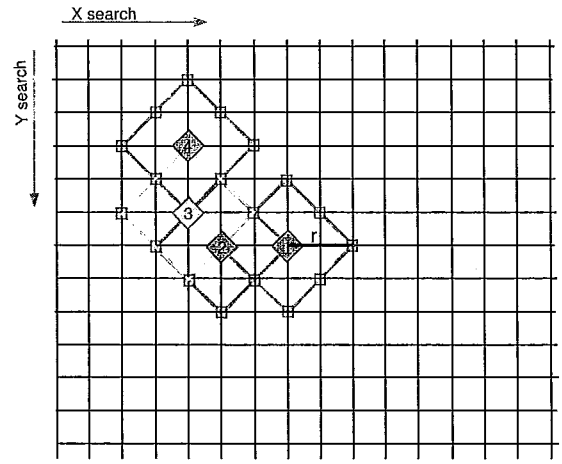


Figure 3.3: Diamond search pattern with searching step = r

During block based motion estimation, we apply the quad-tree structure of variable block size. Figure 3.4 illustrates the quad-tree pattern. Firstly, a motion vector is computed initially by some large block size. If the SAD is larger than the threshold, 4 sub-blocks are generated to do the block-based search recursively.

Secondly, quad-tree blocks provide a trade-off between precision and convergence of block-based motion estimation. On the top level, block-based matching is applied to a big block, and we obtain the first motion vector v_1. Then, if SAD is not lower than some threshold, four sub-blocks are split from their father block. Using v_1 as the prediction (initial motion vector), we employ the matching on those sub-blocks. And this process is performed iteratively, until block's SAD either is lower than the threshold or converges.

Since each quad-tree is independent from each other, motion estimation can be applied to each quad-tree in parallel threads.

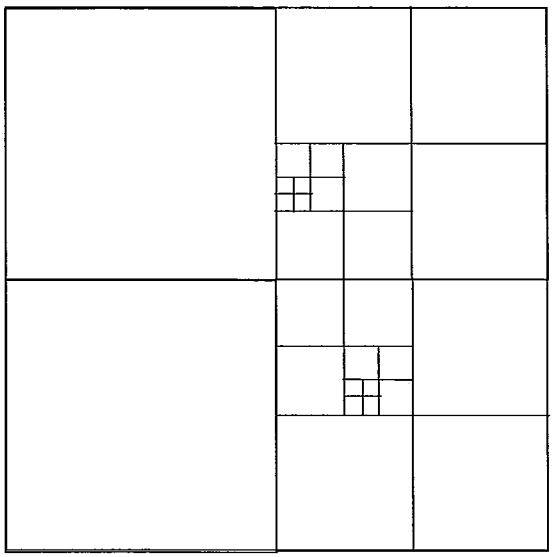


Figure 3.4: Quad-tree pattern

3.2.5 Cost Function

The cost function of motion search is described as follows.

$$\begin{aligned} &\min_{i,j} \; SAD(i,j) + \texttt{position penalty}; \\ &s.t. \quad SAD(i,j) = \sum_{m=0}^{M}\sum_{n=0}^{N} |I(m,n) - I_k(m+i,n+j)|; \\ &\qquad \texttt{position penalty} = w_x(|v_x| + |v_x - v_{x0}|) + w_y(|v_y| + |v_y - v_{y0}|); \\ &\qquad w_x + w_y = \frac{1}{2}, \end{aligned} \tag{3.11}$$

where w_x, and w_y are fixed weights for horizontal and vertical components, (v_x, v_y) is motion vector of the candidate block and (v_{x0}, v_{y0}) is the predicted motion vector of the candidate block. I and I_k are the current frame and reference frame, respectively. A position penalty [10] is added to the cost function (3.11) to regulate the motion field.

With the cost function containing position penalty, there are more chances of

choosing a motion vector with minimum distortion as the position penalty is uniformly distributed. Furthermore, it forces the motion vectors to promise a small motion vector difference between predicted motion and obtained motion.

3.2.6 Steps of Local Motion Estimation

The local motion estimation consists of the following steps.

1. Input: **a** (Affine global motion vector), f_i, f_j (2 spatial denoised frames, Y component required).

2. Compute the local motion region $\mathbb{L}$ by the method described in Section 3.2.1.

3. Compute the pyramid structure for frame f_i and f_j by the method described in section 3.2.2. Only Y component is computed.

4. Push all blocks $b_k \in \mathbb{L}$, its pyramid level ($l_k = 2$, top level), and its predicted motion vector $v_k = (0, 0)$ into a stack $H = \{(b_k, l_k, v_k)\}$.

5. If stack is not empty, pop the top value (b, l, v_p) from the stack H. Otherwise, done.

6. Compute the median vector [25] v_m among v_p and neighborhoods' motion vectors of block b. v_m is the predicted motion vector for Diamond searching. Try Diamond Searching (Section 3.2.3).

$$\begin{aligned} (e, v) &= DS(f_i, f_j, b, l, v_m) \\ \bar{e} &= \frac{e}{Area(b, l)} \end{aligned} \tag{3.12}$$

where e denotes the minimum SAD corresponding to the motion vector v, $DS(\cdot)$ denotes diamond searching, $\bar{e}$ denotes the average absolute difference, and $Area(\cdot)$ computes the area of the block.

7. Let $\bar{e}_{last}$ represent the average absolute difference of the previous diamond searching.

   Different conditions are listed below:

   $|\bar{e} - \bar{e}_{last}| < \epsilon\bar{e}_{last}$: Reach convergence, and no more composition is required. ϵ is a positive constant. If $l > 0$, go down to the lower level to do matching, Push $(b, l-1, 2v)$ into stack H, and goto step 5; If $l == 0$, go to the next condition.

   $\bar{e} <= \varepsilon$ **and** $l == 0$ **(lowest level):** store the motion vector v for block b. End of current block b. Goto step 5;

   $\bar{e} <= \varepsilon$ **and** $l > 0$ **(top level or intermediate level):** push $(b, l-1, 2v)$ into stack H, and goto step 5;

   $\bar{e} > \varepsilon$ **and** $Area(b, l) <= 16$ **and** $l == 0$**:** mark block b "not matched". Goto step 5;

   $\bar{e} > \varepsilon$ **and** $Area(b, l) < 16$ **and** $l > 0$**:** go down to the lower level to do matching. Push $(b, l-1, 2v)$ into stack H. Goto step 5;

   Otherwise: decompose b into four sub-blocks (b_1, b_2, b_3, b_4). Push $(b_k, l, v), k = 1, 2, 3, 4$ into stack H. Goto step 5.

   where ε is proportional to the estimated noise strength, and ϵ denotes a small positive constant.

In step 4, we choose $(0, 0)$ as the initial predicted motion vector, and we also can set it as the motion vector for the block derived from global motion.

3.2.7 Fault Motion Removal

Finally, we try to remove the fault motion vectors from the motion fields obtained above. Because motion estimation can introduce artifacts in the areas where motion is not trackable, for example in the newly exposed or occluded areas. To remove those undesired artifacts of motion vectors, we develop the following method according to the consistent characteristics of directional local motion fields, i.e., motion vectors of pixels inside a rigid object are consistent. The method to remove fault motion vectors can be described as follows.

$$M_{fault} = \{(x,y) | \sum_{(i,j)\in R_{x,y}} ||v_{x,y} - v_{i,j}|| > \mathcal{T}_v\} \tag{3.13}$$

where $v_{x,y}$ denotes the motion vector on pixel (x,y), $R_{x,y}$ is the set of neighborhoods of pixel (x,y), $||\cdot||$ denotes the normal operator, $\mathcal{T}_v$ represents the threshold of motion vector's difference, and M_{fault} denotes the set of pixels with fault motion estimation. All pixels in M_{fault} are marked as non-trackable, and will not be utilized in the following compensation process.

Chapter 4

Temporal Denoising

In general, video signals are more temporally redundant, and temporal denoising provides better performance. Use of spatial denoising is only limited to the area where temporal denoising is not possible (i.e., no match is found for the area in its neighbor frames). As both global camera motion and local object motions are estimated, we can compensate all reference frames for the current frame. If there are two forward frames and two backward frames, we can obtain four corresponding compensated frames. Then a temporal denoising filter will be applied to these four compensated frames and the current frame.

Assume that we have a frame sequence $S = \{I^i | i = 1, \cdots, N\}$ including N frames. Without losing generality, we apply our algorithm on the sub-sequence $\{I^{k-2}, I^{k-1}, I^k, I^{k+1}, I^{k+2}\}$ to de-noise the current frame I^k. For each pixel $I^k_{x,y}$ in a frame, we can establish a pipe as follows.

$$\begin{aligned} &\{I^{k-2}_{T_k(-2,x,y)}, I^{k-1}_{T_k(-1,x,y)}, I^k_{x,y}, I^{k+1}_{T_k(1,x,y)}, I^{k+2}_{T_k(2,x,y)}\} \\ &T_k(j,x,y) = (x,y) + MV_{k,k+j}(x,y) \end{aligned} \tag{4.1}$$

where $I^k_{x,y}$ represents the intensity value of pixel (x, y) in frame I^k, $MV_{k,k+j}(x, y)$ de-

notes the motion vector of pixel $I^k(x,y)$ between frames I^k and I^{k+j}, and $T_k(j,x,y)$ denotes the coordinates of the compensated pixel in reference frame I^{k+j} corresponding to the current pixel (x,y) in frame I^k.

4.1 Weighted Temporal Filter

First, the pixels in each neighboring frame are registered (compensated) according to the estimated motion vectors, to produce temporally redundant copies of the current frame. Second, for each compensated frame, they are used for weight calculation, such that a better match has more weight in making the temporally denoised frame.

The average absolute difference per pixel (ADPP) measures the match quality, and determines the weight of each compensated pixel. ADPP of each compensated pixel can be obtained during motion estimation. ADPP for the current frame is set to that of its closest match among the neighbors.

Our weighted temporal denoising filter can be described as follows.

$$\hat{I}^k_{x,y} = w_{-2} I^{k-2}_{T_k(-2,x,y)} + w_{-1} I^{k-1}_{T_k(-1,x,y)} + w_0 I^k_{x,y} + w_1 I^{k+1}_{T_k(1,x,y)} + w_2 I^{k+2}_{T_k(2,x,y)} \tag{4.2}$$

where $\hat{I}^k_{x,y}$ denotes the denoised value of pixel (x,y) in frame I^k, and (w_j), for $j = -2,-1,0,1,2$ is the weight kernel which can be obtained by (4.3).

$$\begin{aligned} w_j &= \frac{t_j}{\sum_{q=-2}^{2} t_q}, \\ w_0 &= \max_j\{w_j\}, \quad \texttt{for } j = -2,-1,1,2. \\ t_j &= \begin{cases} (\frac{SAD_j}{Area} + \frac{\sigma}{10})^\beta, & \texttt{if compensation is available in frame } I^{k+j}; \\ 0, & \texttt{if compensation is unavailable in frame } I^{k+j}. \end{cases} \end{aligned} \tag{4.3}$$

where $\frac{SAD_j}{Area}$ denotes the average absolute difference for the current pixel (x,y), and

(a) original part of background (b) weighted temporal filter (c) median temporal filter

Figure 4.1: Part of background.

β is a constant used to control the shape of filter kernel. We recommend $\beta = -2$. w_j is non-zero when there is an available compensated block in reference frame I^{k+j}, otherwise $w_j = 0$.

4.2 Median Temporal Filter

Weighted temporal filter is very efficient for the data corrupted by Gaussian noise. However, because of the nonstationarity of the temporal signals, local distributions within a temporal window are generally not Gaussian in case of real data captured by digital camera or scanner. Therefore, a median filter provides a better estimate of the true signal. Temporal median filter is very suitable to remove impulsive noise, because most temporal signals without noise are root signals of a median filter [24]. The median filter is much more efficient to remove impulse noise than the weighted filter. Referring to Figure 4.1, the weighted filter averages the strength of impulse noise, and introduces oversmoothing effect, while the median filter provides a much more clear background.

Our median temporal denoising filter is described by (4.4).

$$\hat{I}^k_{x,y} = Median5\{I^{k-2}_{T_k(-2,x,y)}, I^{k-1}_{T_k(-1,x,y)}, I^k_{x,y}, I^{k+1}_{T_k(1,x,y)}, I^{k+2}_{T_k(2,x,y)}\} \tag{4.4}$$

where $\hat{I}^k_{x,y}$ denotes the denoised value of pixel (x, y) in frame I^k, and $Median5(\cdot)$ denotes a 5-point median filter.

It should be noticed that there are cases of less than four available compensated neighbors (i.e., motion compensation is not available in some reference frames). To account for such cases, our temporal filter is developed as follows:

4 compensated neighbors are found: 5-point median filter.

3 compensated neighbors are found: the weighted temporal filter described by (4.2).

2 compensated neighbors are found: 3 point median filter.

1 compensated neighbor is found: mean filter on two values.

no compensated neighbor is found: Copy the previous spatial denoised pixels for high deviation regions, and apply Gaussian filter for low deviation regions.

With the help of fast implementation of the median filter (See Appendix A), we can do the median temporal denoising more efficiently.

Chapter 5

Implementation And Experimental Results

In this chapter, we will detail our implementation and explain our experiment results. First, we introduce our experimental environment and detail our algorithm implementation.

5.1 Algorithm Implementation

We implemented our proposed algorithm by C/C++ and assembler language. The PC configuration in our experiment is listed as follows.

Operating System: Windows XP with sp2;

CPU: Intel Duo Core 2.2G;

Memory: 2Gb, DDR533;

Next, we present some highlights of our implementation as follows.

1. Multi-threading technology takes the computational advantage of Duo Core CPU. The computational resource of CPU can be taken full use of by our algorithm, and I/O operations and computing operations are processed in parallel.

2. Denoising a sequence can be divided into several processing threads each of which applied our fast denoising algorithm. The advantage is the parallelization of I/O operation and CPU computation. The denoising process has a large number of I/O operations, such as input frame reading, output frame writing, etc. These operations are independent between different threads.

3. For local motion estimation, there are two facts allowing parallelization. First, local motion estimation can be divided into two independent processes: forward motion, and backward motion. Forward motion and backward motion estimation are independent with each other, and can be computed parallelly. Second, the motion searching of blocks is independent with each other.

4. We use SSE2 instructions to optimize some time-consuming parts, such as median filter, and SAD computation. Parallel computing technology highly accelerates the processing speed.

5.2 Experimental Results

We set up our experiments with high-resolution videos, which mainly include some video sequences from digital cinema camera and film scanner that incorporate different types of motion, including one slow motion sequence, one fast motion sequences, one natural scene, and one animation sequence. A more detailed description of the sequences used can be found in Table.5.1.

In order to demonstrate the higher processing throughput and better performance, we compare our proposed method with "Neat Video" Denoiser [15], one of the most

Table 5.1: Descriptions of high resolution test video sequences.

Sequence	Number of Frames	Resolution and color space
Swing Angel	40	4096 × 2048 (RGB48)
Waving Hand	40	2048 × 1556 (RGB30)
Animation Roberts	40	1920 × 1436 (RGB30)
Dark Background	40	2048 × 1556 (RGB30)

Table 5.2: Processing time comparison between proposed method and Neat Video Denoiser (NVD).

Video Sequences	Our Algorithm		Neat Video Denoiser	
	Total time (seconds)	Average time per frame (sec.)	Total time (seconds)	Average time per frame (sec.)
Swing Angel	197.1	4.93	997	24.9
Waving Hand	81.6	2.04	383	9.6
Animation Roberts	77.5	1.87	357	8.9
Dark Background	82.7	2.07	409	10.2

well-known commercial denoisers. "Neat Video" denoiser (NVD) is a digital video filter. Its main function is to reduce noise in digital video sequences. We use the preset "hard noise" in the NVD.

The overall and average processing time of proposed method and NVD are listed in Table.5.2. Obviously, the processing time results show that the proposed method is almost five times faster than NVD.

Next, we analyze the performance of the proposed temporal denoiser, and compare visual quality with that of NVD. Because these noisy test sequences are directly captured by digital cinema cameras and there are not corresponding true video sequences without noise, we can not evaluate the performance by peak signal-to-noise rate (PSNR). Therefore, we evaluate the performance by the visual quality of denoised video frames.

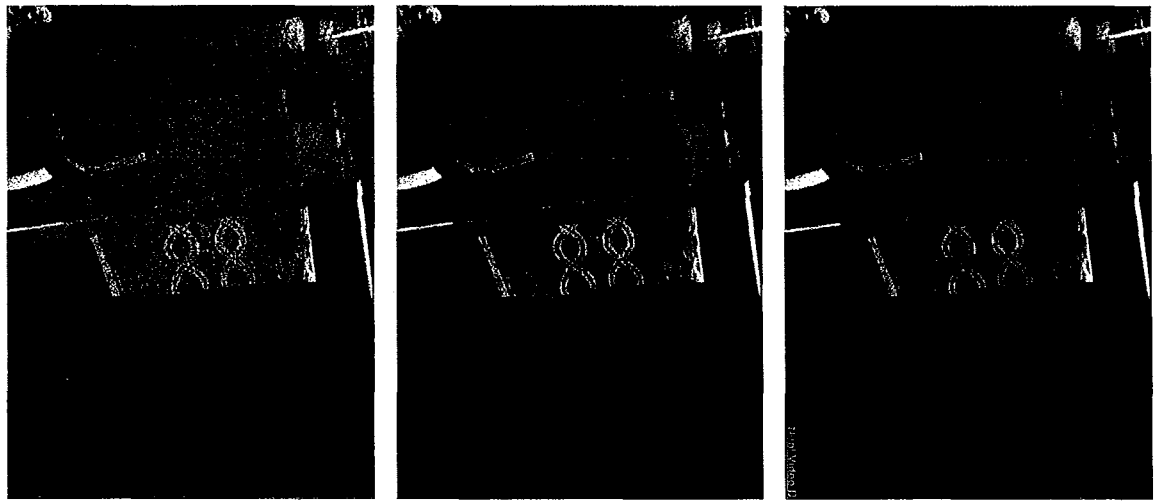
1. Dark background. Resolution 2048 × 1556, 30-bit depth.

In this sequence, the digital camera captured an indoor scene at night. The

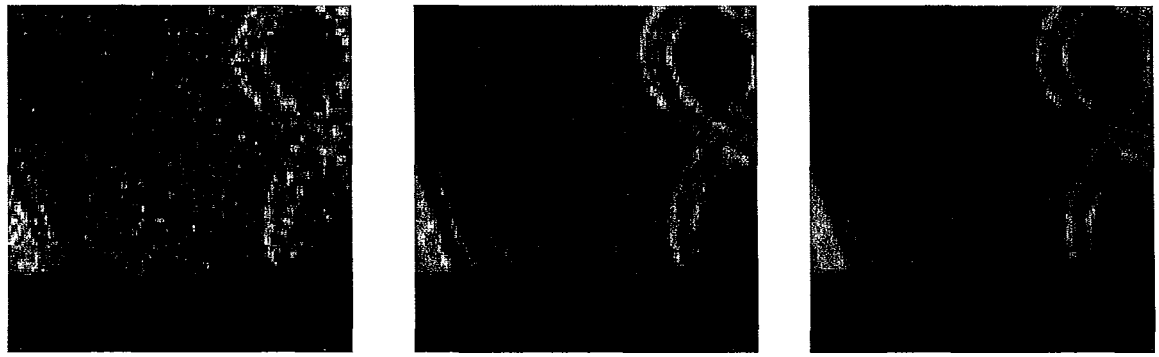
signal-to-noise rate is serious low. The global motion and local object motion are slow. Figure.5.1(a) shows part of a frame. The noise is much higher in the dark background, for example the floor carpet. We can not recognize the pattern on the carpet in the original frame, because the signal-to-noise rate is too low. The global motion is caused by the camera motion, and there are some random local motion for some lights. Figure.5.1(b) and Figure.5.1(c) show the result of our proposed method and Neat Video Denoiser (NVD), respectively. They both reduce noise, and provide good visual quality. The former gives a bit more crisp pictures, and prevents edges from blurred. The latter introduces oversmoothing effects, and make the edges blurred. The differences between results of two methods are obviously shown in the detail blow-up Figure 5.1(e), 5.1(h)), 5.1(f) and 5.1(i). There are some visual improvements in the proposed result: the remoter on the desk is more clear than that of NVD, and the pattern of the tablecloth is sharper than that of NVD. The average processing time of proposed method is 2.07 seconds per frames, which is much less than that of NVD, 10.2 seconds.

2. Swing Angel. Resolution 4096×2048. 48-bit depth.

This sequence contains high-level noise. In this sequence, a girl wearing two artificial swings is sitting at the front of a blue background. There is a slight motion for her body and swings. The motion is slow. We show a part of a frame (Figure 5.2(a)), including a red gem on her swings. The red gem is surrounded by several small white diamonds. Figure 5.2(b) is the result of proposed denoising algorithm, and Figure 5.2(c) is the result of NVD. The former removes noise, and keeps excellent sharpness of edges and highlighted small white diamonds. The latter removes noise, but at the same time introduces oversmoothing effects, and blurs edges and some details, for example the white small diamonds below the red gem. Therefore, the proposed method removes noise and preserves more details than NVD. For this high



(a) original frame (b) Result of proposed method (c) NVD's result



(d) Original detail 1 (e) Proposed method's result (f) NVD's result

(g) Original detail 2 (h) Proposed method's result (i) NVD's result

Figure 5.1: Dark background sequence.

(a) A part of a original frame (b) Result of proposed method (c) Result of NVD

Figure 5.2: Blow-up part of Swing Angel sequence.

resolution video (a 4K video), the proposed algorithm obtains a high speed, 4.93 seconds per frame on average, while NVD consumes 24.9 seconds per frame.

3. Waving Hand. Resolution 2048×1556, 30-bit depth.

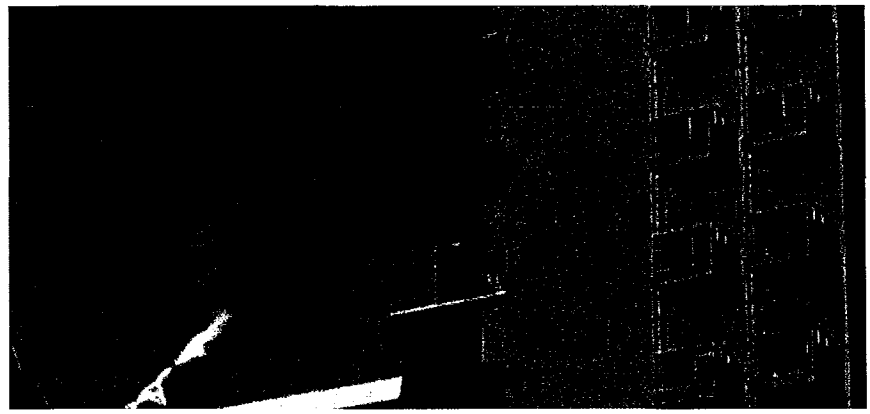
This sequence is characterized by high-level noise, fast object motion. The camera captured a scene in which a girl in white is waving her hands in front of a building. The camera motion is slow, but the motion of waving hand is fast. Figures 5.3(a - e) show part of five frames, including her right hand and the building. The building obeys global camera motion, and the waving hand introduces local motion. There are three facts making the local motion estimation very hard. Firstly, the waving hand is moving fast, and the motion search range is very large (about 107 pixels between Figure 5.3(c) and 5.3(e)). Secondly, the waving hand is not a rigid object, i.e. the shape of her hand is keeping changing. Thirdly, there is motion blur effect around her hand. Our proposed method overcomes these difficulties: Global motion estimation will estimate the global motion of the building; The local motion area can be excluded from global motion area; The local motion estimator is capable of track the motion with large range. The hand is decomposed into small blocks by quad-tree structure to simulate its reshaping, and fault motion vectors are removed

by motion analysis. Compared with NVD's result (Figure 5.3(h), result of proposed method (Figure 5.3(g)) removes noise without blurring edges. From the detail blow-up (Figure 5.4 (c - f)), proposed method preserves sharper high frequency barrier. For this 2K video, proposed method reaches the speed at 2.04 seconds per frame on average, which is almost five times faster than NVD.

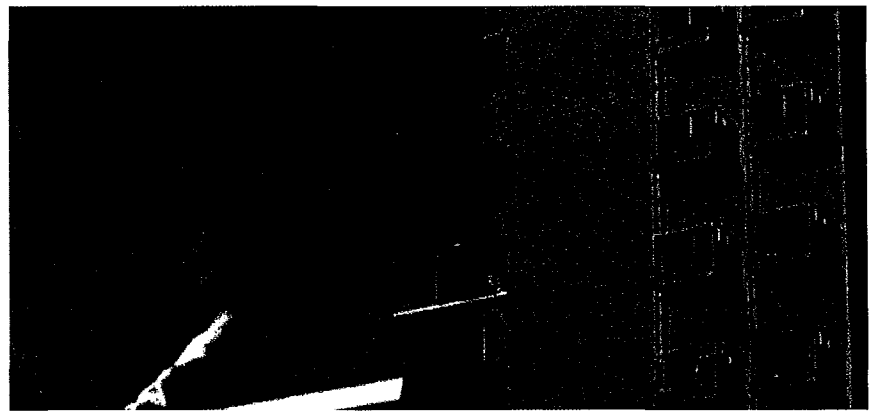
4. Animation robots. Resolution 1920×1436, 30-bit depth.

This sequence was digitized from films by a motion picture scanner. The video sequence is characterized by high-level noise, and fast/incontinuous motion in relatively small areas. In this sequence, an artifact robot is shooting. The motion of the flashing light is incontinuous, and is hard to accurately estimate. With the help of motion analysis, we can find unmatched regions near the flashing light. We only apply the intra-frame filter on those regions. Figures 5.5(a - e) show part of first five frames. Figure 5.5(g) and 5.5(h) show the result of proposed method and NVD, respectively. Both methods provide good visual quality for this animation sequence, and their performance are similar. The advantage of the proposed method is its high processing speed, which is more than four times faster than NVD.

(a) Original frame 1(b) Original frame 2(c) Original frame 3(d) Original frame 4(e) Original frame 5



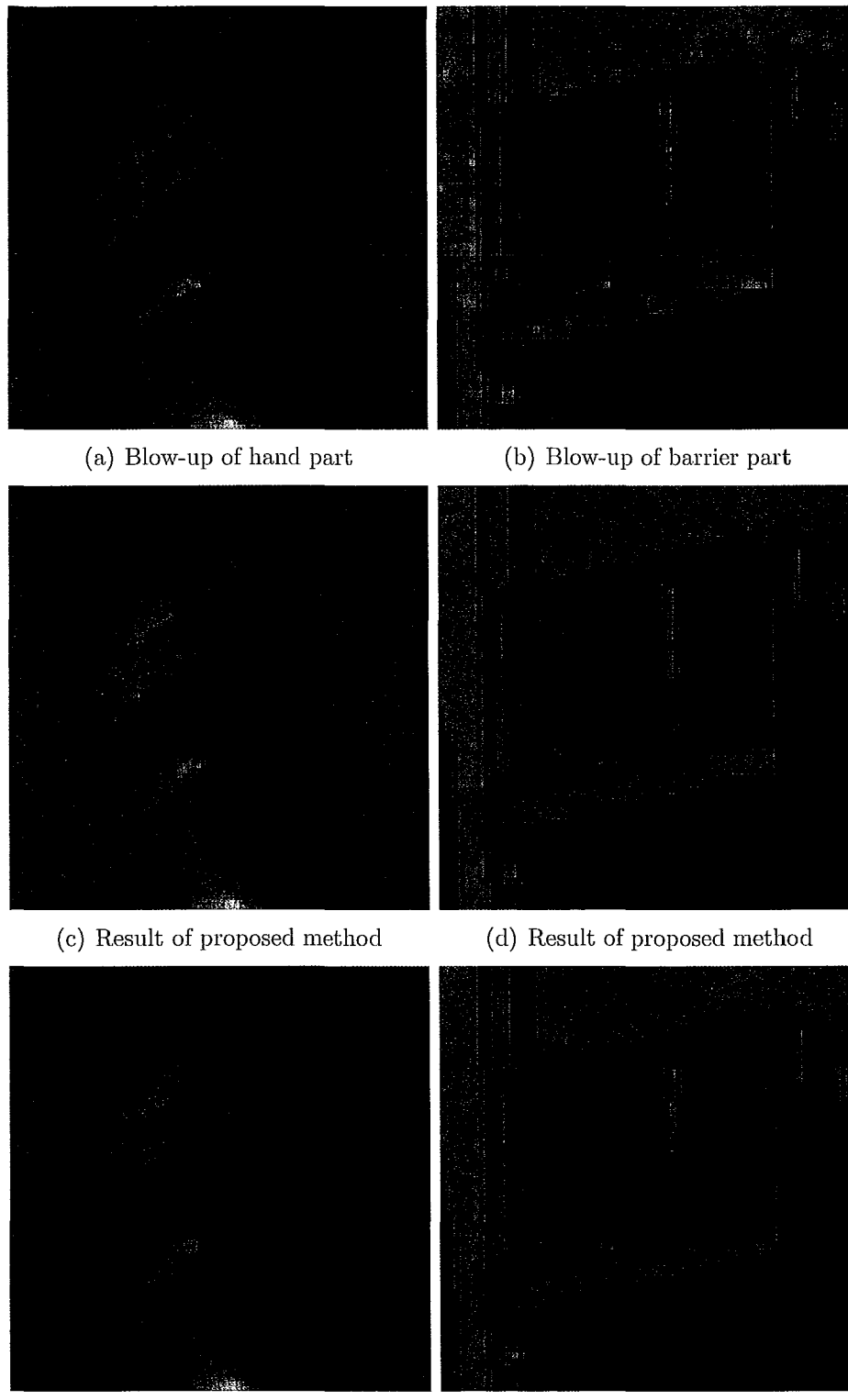
(f) Original frame 3



(g) Result of proposed method

(h) Result of NVD

Figure 5.3: Waving hand sequence.



(a) Blow-up of hand part (b) Blow-up of barrier part

(c) Result of proposed method (d) Result of proposed method

(e) NVD's result (f) NVD's result

Figure 5.4: Detail blow-up of waving hand sequence.

(a) Original frame 1(b) Original frame 2(c) Original frame 3(d) Original frame 4(e) Original frame 5

(f) Original frame 3 (g) Result of proposed method

(h) NVD's result

(i) Original detail

(j) Proposed method

(k) NVD

Figure 5.5: Animation robot sequence.

Chapter 6

Conclusions

In this thesis we investigated the problem of video denoising, which plays an important role in many applications of video processing. Funded by an industrial research grant on digital cinema, we are primarily interested in the removal of noises in high-resolution movie contents, either scanned film or the raw data of digital cinema cameras. In pursue of best possible visual quality, we adopted the approach of motion-compensated temporal video denoising. Our study exposed some weaknesses of existing video denoising algorithms, such as their incapability of modeling and tracking complex motions, the use of overly simplistic noise models, and high computational cost.

To overcome these weaknesses and improve the performance and robustness of existing video denoising methods, we proposed a new joint spatial-temporal video denoising algorithm that combines multiframe inter-frame compensation and directional intra-frame filtering. In the algorithm a noisy video is cleaned in two phases: spatial (intra-frame) denoising followed by temporal (inter-frame) denoising. In the spatial denoising phase, each video frame is denoised by a low-pass directional filter whose parameters are tuned to the estimated noise level and to the gradient of the current pixel. The directional filter has the advantage of effectively suppressing noises

without blurring edges. The resulting sequence of spatially denoised frames is further processed in the temporal inter-frame denoising phase to improve the results of the intra-frame denoiser by exploiting inter-frame correlations. This is achieved by registering multiple frames via motion estimation and low-pass filtering of the current pixel along motion trajectory. During motion estimation, general compound motions are taken into account, including both global camera motion and local object motions. We use an affine motion model to characterize the global camera motion, and a blockwise translational motion model to approximate local object motions.

The proposed video denoising algorithm is implemented and tested extensively on high-resolution digital cinema contents. Great efforts are devoted to achieving high throughput of our video denoiser. In particular, parallel processing techniques at machine instruction level are adopted to remove computation bottlenecks in temporal denoising, such as median filtering and block matching in motion estimation. We are able to achieve a throughput five times higher than a commercial video denoising package, and at the same time obtain better visual quality of the denoised video contents. In our experiments the proposed new video denoising algorithm also exhibits an improved level of robustness. Its performance does not dramatically deteriorate in difficult cases of discontinuous and sporadic motions and object occlusions.

Bibliography

[1] Abhijit Sinha, Xiaolin Wu, "Fast Generalized Motion Estimation and Superresolution",Image Processing, 2007, ICIP 2007. Vol.5, pp.413 - pp.416.

[2] John Smith, "Implementing median filters in XC4000E FPGAs", http://www.xilinx.com/xcell/xl23/xl23_16.pdf

[3] F. Cocchia, S. Carrato, and G. Ramponi, "Design and real-time implementation of a 3-D rational filter for edge preserving smoothing", IEEE Transactions on Consumer Electronics, vol.43, no.4, pp.1291 - 1300, Nov. 1997.

[4] R. Kleihorst, R. Lagendijk, and J. Biemond, "Noise reduction of image sequences using motion compensation and signal decomposition", IEEE Trans. on Image Processing, vol.4, no.3, pp. 274 - 284, Mar. 1995.

[5] R. Rajagopalan and M. Orchard, "Synthesizing processed video by filtering temporal relationships", IEEE Trans. on Image Processing, vol. 11, no. 1, pp. 26 - 36, Jan. 2002.

[6] G. Haan, "Ic for motion-compensated de-interlacing, noise reduction and picture rate conversion", IEEE Trans. on Consumers Electronics, vol. 45, no. 3, pp. 617 - 623, Aug. 1999.

[7] G. Haan, et al.,"True-Motion Estimation with 3-D Recursive Search Block Matching", IEEE Transactions on Circuits and Systems for Video Technology, vol. 3, No. 5, pp. 368 - 379, Oct. 1993.

[8] J. Canny, "A computational approach to edge detection", IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 9, pp. 679 - 698, Nov. 1986.

[9] P. L. Rosin, "Thresholding for change detection", in Proc. IEEE Int. Conf. of Computer Vision, India, pp. 274 - 279, Jan. 1998.

[10] H. Yeo, C. A. Gonzales, et. al.,"A cost function with position penalty for motion estimation in mpeg-2 video coding", IEEE Intl. Conf. of Multimedia And Expo, 2000, NY, USA, pp. 1755 - 1758.

[11] Shan Zhu, Kai-Kuang Ma, "A new diamond search algorithm for fast block-matching motionestimation", IEEE Trans. Image Processing, Vol. 9, no. 2, pp. 287 - 290, Feb. 2000.

[12] A. N. Avanaki, "A Spatiotemporal Edge-preserving Denoising Method for High-quality Video", in Proc. IEEE ISSPIT, 2006.

[13] Dalsa(R) digital cinema website:

`http://www.dalsa.com/dc/4K_products/4K_products.asp`

[14] Imagica motion picture scanner website:

`http://www.imagica-technologies.com/english/imagerXE/index.html`

[15] Neat video denoiser home page:

`http://www.neatvideo.com/index.html`

[16] A. K. Katsaggelos, Ed., "Digital Image Restoration", vol. 23., chapt. 1, New York: Springer-Verlag, 1991.

[17] A. J. Patti, A. M. Tekalp, and M. I. Sezan, "A new motion-compensated reduce-order model Kalman filter for space-varying restoration of progressive and interlaced video", IEEE Trans. Image Process., vol. 7, no. 4, pp. 543C554, Apr. 1998.

[18] E. J. Balster,Y. F. Zheng, and R. L. Ewing, "Combined spatial and temporal domain wavelet shrinkage algorithm for video denoising", IEEE Trans. Circuits Syst. Video Technol., vol. 16, no. 2, pp. 220C230, Feb. 2006.

[19] A. Buades, B. Coll, and J. M. Morel, "Denoising image sequences does not require motion estimation", in Proc. IEEE Int. Conf. on Adv. Video and Signal Based Surveillance, Sep. 2005, pp. 70C74.

[20] R. Dugad and N. Ahuja, "Video denoising by combining Kalman and wiener estimates", in Proc. IEEE Int. Conf. on Image Process., Oct. 1999, pp. 152C156.

[21] H.-H. Nagel, "On the estimation of optical flow: relations between different approaches and some new results", Artificial Intelligence, vol.33, no. 3, pp.298 – 324, Nov. 1987.

[22] J. K. Kearney, W. B. Thompson, and D. L. Boley, "Optical flow estimation: an error analysis of gradient-based methods with local optimization", IEEE Trans. on Pattern Analysis and Machine Intelligence, vol.9, no.2, pp.229 – 244, 1987.

[23] A. Verri, T. Poggio, "Motion Field and Optical Flow: Qualitative Properties", IEEE Trans. on Pattern Analysis and Machine Intelligence, vol.11 no.5, pp.490-498, May 1989.

[24] S. S. H. Naqvi, N. C. Gallagher, and E. J. Coyle, "An application of median filters to digital television", in Proc. IEEE Int. Conf Acoust., Signal Process., Tokyo, Japan, 1986, pp. 2451 - 2454.

[25] L. Alparone, M. Barni and F. Bartolini,"Adaptively weighted vector-median filters for motion-fields smoothing", ICASSP, pp.2267-2270, 1996.

[26] A.K. Jain, "Foundimentals of Digital Image Processing", Englewood Cliffs, NJ, Prentice-Hall, 1989.

[27] C.Q. Zhan and I.J. Karam, "Wavelet-Based Adaptive Image Denoising With Edge Preservation", Proc. of the IEEE 2003 Int. Conf. on Image Process., MA-L2, Sept. 2003.

[28] A. K. Katsaggelos, R. P. Kleihorst, and et al., "Adaptive image sequence noise filtering methods", in SPIE Proc. Vis. Comm. and Image Process., Nov. 1991, Boston, MA, vol. 1606, pp. 716-727.

Appendix A

Fast Implementation of Median Filter

A.1 5-point Median Filter

The 5-point median filter is implemented by assembler language employing SSE2 instructions. RGB pixels are planar, and each component is 16-bit depth.

```
const __declspec(align(16)) unsigned short MIN_SHORT_ARRAY[8] = {
    0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000
};
 inline void med5_1(unsigned short *p, unsigned short *d) {
    __asm {
        movdqa      xmm7, [MIN_SHORT_ARRAY]
        push        esi
        push        edi
        push        ecx
        mov         esi, [p]
        mov         edi, [d]
        mov         ecx, 96 - 16
        L_LOOP:
```

```
//read 8 red value from the first block, 16-bit depth
movdqa      xmm0, [esi + ecx + 0*96]
//convert unsigned short to signed short
psubw       xmm0, xmm7
//read 8 red value from the second block, 16-bit depth
movdqa      xmm1, [esi + ecx + 1*96]
psubw       xmm1, xmm7
//read 8 red value from the third block, 16-bit depth
movdqa      xmm2, [esi + ecx + 2*96]
psubw       xmm2, xmm7
//read 8 red value from the fourth block, 16-bit depth
movdqa      xmm3, [esi + ecx + 3*96]
psubw       xmm3, xmm7
//read 8 red value from the fifth block, 16-bit depth
movdqa      xmm4, [esi + ecx + 4*96]
psubw       xmm4, xmm7
//save a copy
movdqa      xmm5, xmm0
//min operation with 8 value parallelly
pminsw      xmm0, xmm1
//max operation with 8 value parallelly
pmaxsw      xmm5, xmm1
movdqa      xmm6, xmm3
pminsw      xmm3, xmm4
pmaxsw      xmm6, xmm4
pmaxsw      xmm3, xmm0
pminsw      xmm5, xmm6
movdqa      xmm0, xmm5
pminsw      xmm5, xmm2
pmaxsw      xmm0, xmm2
pminsw      xmm0, xmm3
//get the median value of 5 value with 8 parallelly.
```

```
        pmaxsw      xmm0, xmm5
        //convert signed short to unsigned short
        paddw       xmm0, xmm7
        //write 8 median values to destination.
        movdqa      [edi + ecx], xmm0
        sub         ecx, 16
        jns         L_LOOP
        pop         ecx
        pop         edi
        pop         esi
    }
}
```

A.2 3-point Median Filter

The 3-point median filter is implemented by assembler language employing SSE2 instructions. RGB pixels are planar, and each component is 16-bit depth.

```
inline void med3_1(unsigned short *p, unsigned short *d) {
    __asm {
        movdqa      xmm7, [MIN_SHORT_ARRAY]
        push        esi
        push        edi
        push        ecx
        mov         esi, [p]
        mov         edi, [d]
        mov         ecx, 96 - 16
        L_LOOP:
        //read 8 pixel value from the first block
        movdqa      xmm0, [esi + ecx + 0*96]
        //convert unsigned short to signed short
```

```
        psubw       xmm0, xmm7
        //read 8 pixel value from the second block
        movdqa      xmm1, [esi + ecx + 1*96]
        psubw       xmm1, xmm7
        //read 8 pixel value from third block
        movdqa      xmm2, [esi + ecx + 2*96]
        psubw       xmm2, xmm7
        movdqa      xmm5, xmm0
        pminsw      xmm0, xmm1
        pmaxsw      xmm1, xmm5
        pminsw      xmm1, xmm2
        pmaxsw      xmm0, xmm1
        paddw       xmm0, xmm7
        movdqa      [edi + ecx], xmm0
        sub         ecx, 16
        jns         L_LOOP
        pop         ecx
        pop         edi
        pop         esi
    }
}
```

Appendix B

Fast Implementation of Block-based SAD

The computation of SAD between two 8x8 blocks is implemented by assembler language employing SSE2 instructions. RGB pixels are planar, and each component is 16-bit depth.

```
unsigned int SAD_A_8x8(TIMG *r, TIMG *c, unsigned int pitch) {
__asm{  push        esi
        push        edi
        push        ecx
        pxor        xmm5, xmm5
        pxor        xmm6, xmm6
        pxor        xmm7, xmm7
        mov         esi, [r]
        mov         edi, [c]
        mov         eax, [pitch]
        lea         ecx, [eax * 8]
        sub         ecx, eax
        movdqu      xmm1, [esi + ecx * 2]
        movdqu      xmm0, [edi + ecx * 2]
```

```
L_LOOP: movdqa     xmm2, xmm1
        psubusw    xmm1, xmm0
        psubusw    xmm0, xmm2
        por        xmm0, xmm1
        sub        ecx, eax
        movdqa     xmm3, xmm0
        movdqa     xmm4, xmm0
        movdqu     xmm1, [esi + ecx * 2]
        movdqu     xmm0, [edi + ecx * 2]
        punpcklwd  xmm3, xmm7
        punpckhwd  xmm4, xmm7
        paddd      xmm5, xmm3
        paddd      xmm6, xmm4
        jnz        L_LOOP
        movdqa     xmm2, xmm1
        psubusw    xmm1, xmm0
        psubusw    xmm0, xmm2
        por        xmm0, xmm1
        paddd      xmm6, xmm5
        movdqa     xmm3, xmm0
        movdqa     xmm4, xmm0
        punpcklwd  xmm3, xmm7
        punpckhwd  xmm4, xmm7
        paddd      xmm6, xmm3
        paddd      xmm6, xmm4
        movdqa     xmm0, xmm6
        psrldq     xmm6, 8
        paddd      xmm6, xmm0
        movdqa     xmm0, xmm6
        psrldq     xmm6, 4
        paddd      xmm6, xmm0
        movd       eax, xmm6
```

```
        pop         ecx
        pop         edi
        pop         esi
    }
}
```