TAMPERE UNIVERSITY OF TECHNOLOGY

JUNSHENG FU

A Real-time Rate-distortion Oriented Joint Video Denoising and Compression Algorithm

Master of Science Thesis

Subject approved in the Department Council meeting on the 23rd of August 2011

Examiners: Prof. Karen Egiazarian
Dr. Alessandro Foi

Preface

In September 2010, I started to work as a research assistant in Transforms and Spectral Techniques Group in the Department of Signal Processing at Tampere University of Technology. The purpose of this thesis is to enhance the compression performance of a video coding standard using better filtering strategies. The main parts of this thesis are implemented and documented in 2011.

I would first like to acknowledge my supervisor and examiner Prof. Karen Egiazarian for the opportunity of working in such a nice group and his guidance during my thesis work. I would also like to show my gratitude to Dr. Evgeny Belyaev, for his invaluable supports and the numerous fruitful discussions during the development of this thesis. Further, I thank my second examiner Alesssandro Foi, and other colleagues in Transforms and Spectral Techniques Group for their assistance and friendliness.

Looking back, the two years studying in Finland, have been an unforgettable experience in my life - fellowship, wonderful friends, "Tech. salon", badminton club and Finnish sauna.

Finally, I am deeply indebted to my father Jianhua Fu, mother Qiulian Xia and my best friend Mengting He for their unconditional love, encouragement and support. I could not have become what I am now without them.

Tampere, Finland
November 2011
Junsheng Fu

Abstract

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
JUNSHENG FU : A Real-time Rate-distortion Oriented Joint Video Denoising and Compression Algorithm
Master of Science Thesis: 58 pages
September 2011
Major: Signal Processing
Examiner: Prof. Karen Egiazarian, Dr. Alessandro Foi
Keywords: real-time filter, pre-filtering, in-loop filtering, H.264/AVC

This thesis proposes a real-time video denoising filter, a joint pre-filtering and compression algorithm, and a joint in-loop filtering and compression algorithm.

A real-time video denoising filter: a great number of digital video applications motivate the research in restoration or enhancement methods to improve the visual quality in the presence of noise. Video Block-Matching and 3D collaborative filter, abbreviated as VBM3D, is one of the best current video denoising filters. We accelerate this filter for real-time applications by simplifying the algorithm as well as optimizing the codes, while preserving its good denoising performance.

A joint pre-filtering and compression algorithm: pre-filtering and compression are two separate processes in traditional systems and they do not guarantee optimal filtering and quantization parameters with respect to rate-distortion framework. We propose a joint approach with pre-filtering by VBM3D and compression by H.264/AVC. For each quantization parameter, it jointly selects the optimal filtering parameter among the provided filtering parameters. Results show that this approach enhances the performance of H.264/AVC by improving subjective visual quality and using less bitrates.

A joint in-loop filtering and compression algorithm: in traditional video in-loop filtering and compression systems, a deblocking filter is employed in both the encoder and decoder. However, besides blocking artifacts, videos may contain other types of noise. In order to remove other types of noise, we add a real-time filter as an enhancing part in the H.264/AVC codec after the deblocking filter. Experiments illustrate that the proposed algorithm improves the compression performance of

the H.264/AVC standard by providing frames with increased PSNR values and less bitrates.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays there are a great number of practical applications involving digital videos, but digital videos can be easily corrupted by noise during acquisition, processing or transmission. A lot of research has been carried out in video restoration and enhancement solutions to improve the visual quality in the presence of noise. Video Block-Matching and 3D collaborative filter [1], abbreviated as VBM3D, is one of the best current video denoising filters, and it achieves state-of-the-art denoising performance in terms of both peak signal-to-noise ratio and subjective visual quality.

However, due to the computational complexity of the algorithm, the speed at which the current implementation of VBM3D executes makes it hard to be used for real-time applications. In this thesis, we define the real-time requirement as: the filter should have at least 25 fps for processing frames with a resolution of 640 $\times$ 480 under computer platform with Intel Core 2 Duo 3 GHz and 3.2 GB of RAM. To meet this requirement, while preserving the good denoising performance, we balance between complexity and speed, optimize the code and propose an integer implementation.

In the current video compression systems, the most essential task is to fit a large amount of visual information into a narrow bandwidth of transmission channels or into a limited storage space, while maintaining the best possible visual perception for the viewer [2]. H.264/AVC is one of the most commonly used video compression standards in areas of broadcasting, streaming and storage. It has achieved a significant improvement in rate-distortion efficiency over previous standards [3].

The noise in video sequences not only degrades the subjective quality, but also affects compression processes. The H.264/AVC codec uses only a filter to decrease blocking artifacts. To enhance the compression performance, some filtering strategies are usually employed, such as pre-filtering, in-loop filtering and post-filtering. In this thesis, we focus on pre-filtering and in-loop filtering.

In traditional video pre-filtering and compression systems, pre-filtering and compression are two separate processes and do not guarantee optimal filtering and quantization parameters. It has been suggested that joint pre-filtering and compression

algorithm improves the performance of the compression by producing compressed video frames, with increased PSNR values and less compression artifacts, at the same bitrates, compared to standard compression[4]. We continue this research of joint parameters selection, and propose a joint algorithm with pre-filtering by VBM3D and compression by the H.264/AVC encoder.

In traditional video in-loop filtering and compression systems, a deblocking filter is employed to remove blocking artifacts introduced in the compression process. However, videos may contain other types of noise, and it is desirable to remove them as well. The method presented in the literature [5] suggests that adding a spatial-temporal filter in the H.264/AVC codec improves the compression performance. We continue this research and present a joint in-loop filtering and compression algorithm by adding the proposed real-time filter as an enhancing part into the H.264/AVC codec. The joint scheme is designed, tested and analyzed.

This thesis is structured as follows:

- Chapter 2 briefly describes the H.264/AVC standard. The reader is guided through characteristics of video codec, main functional parts of H.264/AVC and its compression performance in the presence of noise.

- Chapter 3 discusses some general video denoising methods with a focus on VBM3D. This helps the reader to understand general video denoising strategies and how VBM3D achieves state-of-the-art denoising performance in terms of both peak signal-to-noise ratio and subjective visual quality. Further, a real-time integer implementation of the simplified VBM3D is proposed.

- Chapter 4 illustrates traditional video filtering and compression schemes as well as their drawbacks. Then two joint filtering and compression algorithms are proposed: one is a joint pre-filtering and compression algorithm; the other is a joint in-loop filtering and compression algorithm. Finally, results of both algorithms are analyzed.

- Chapter 5 summarizes the results of this study and provides suggestions for the future work.

Chapter 2

Video Compression using the H.264/AVC Standard

2.1 Main Characteristics of Video Codec

2.1.1 Introduction

Video codec is a software that compresses and decompresses digital videos. By using it, a large amount of visual information can be put into a limited storage space or a narrow bandwidth of transmission channel. Many different kinds of codecs were designed in the last twenty years. In order to compare different codecs, three main characteristics need to be taken into consideration: *visual quality of compressed video, bitrate*, and *complexity*. In this section, these three characteristics will be introduced one by one, and a brief overview of the widely used video compression standard H.264/AVC will be presented.

2.1.2 Visual quality

In order to evaluate and compare video codecs, it is necessary to estimate the visual quality of compressed video frames displayed to the viewer.

Video visual quality is actually subjective and viewers' opinions of visual quality can be various. So usually it is more complex and difficult to use subjective criteria to obtain the measurement of video visual quality. On the other hand, objective quality measurement method gives accurate and repeatable results and has low complexity. It is widely used in video compression and processing systems. In this thesis, we use *Peak-Signal-to-Noise Ratio* to measure the visual quality of video frames. The *Mean Squared Error* and *Peak-Signal-to-Noise Ratio* are discussed below.

The *Mean Squared Error*, abbreviated as *MSE*, is one common way to measure the difference between two signals. It is the average of the square of the difference between the desired response and the actual system output. In 2D images, if I is an

original image and I' is the same image corrupted by a noise, *MSE* can be expressed as:

$$MSE = \frac{1}{|X|} \sum_{x \in X} (I(x) - I'(x))^2, \tag{2.1}$$

where $x \in X \subset \mathbb{Z}^2$, $I(x)$ is a pixel of I at position of x.

The *Peak-Signal-to-Noise Ratio*, abbreviated as *PSNR*, indicates the ratio between the maximum possible power of a signal and the power of corrupting noise. Usually, *PSNR* is expressed in the term of a logarithmic decibel scale. It is defined as :

$$PSNR = 10 \log_{10} \Big(\frac{MAX^2}{MSE} \Big), \tag{2.2}$$

where MAX is the maximum possible value of the signal, e.g. if each pixel is represented by 8 bits, then $MAX = 255$.

In order to compare different lossy compression codecs, *PSNR* is the most commonly used quality measurement. In this case, the signal is the original data, and the noise is the error introduced by the compression. However, it should be noticed that high *PSNR* values do not always guarantee high human visual quality perception [8]. In this thesis, *PSNR* is used as a quality measurement due to its simple calculation and clear physical meaning.

2.1.3 Bitrate

In video coding, *bitrate* is the number of bits generated by the a codec in a unit of time, usually a second. Therefore, *bitrate* can be measured in "bits per second" (bit/s), or in conjunction with a metric prefix, e.g., kilo (kbit/s).

2.1.4 Complexity

Complexity of a video codec can be expressed as the number of arithmetic operations used in processing a video. But in real applications, the number of operations do not show the full complexity because they do not include the memory accesses and logical operations. Therefore, we consider *complexity* as the number of processed frames in a unit of time, for the given frame resolution and computer platform.

2.2 General Scheme of H.264/AVC

H.264/AVC (Advanced Video Coding) is one of the most commonly used video compression standards. It is a block-oriented motion-compensation-based codec standard developed by the ITU-T Video Coding Experts Group (VCEG) together with the ISO/IEC Moving Picture Experts Group (MPEG). It is published jointly as Part 10 of MPEG-4 and ITU-T Recommendation H.264 [6, 7].

Some important terminologies in the H.264/AVC standard is discussed before briefing the scheme of H.264/AVC.

A video sequence can be divided into several *groups of pictures* (GOP). Each *group of pictures* may contain several frame types: *I-frames, P-frames* and *B-frames*. An *I-frame* is an "intra-coded frame". It is the least compressible frame and decodes itself without the aid of other frames. A *P-frame* is a "predicted frame". It uses data from previous frames to decompress and is more compressible than *I-frame*. A *B-frame* is a "bidirectional predicted frame". It uses both previous and forward frames as references to achieve the highest amount of data compression. The coding and display orders of frames are not necessary the same. Figure 2.1 illustrates one example of *group of pictures, I-frame, P-frames*, *B-frames* and *decoupled coding order and display order*.

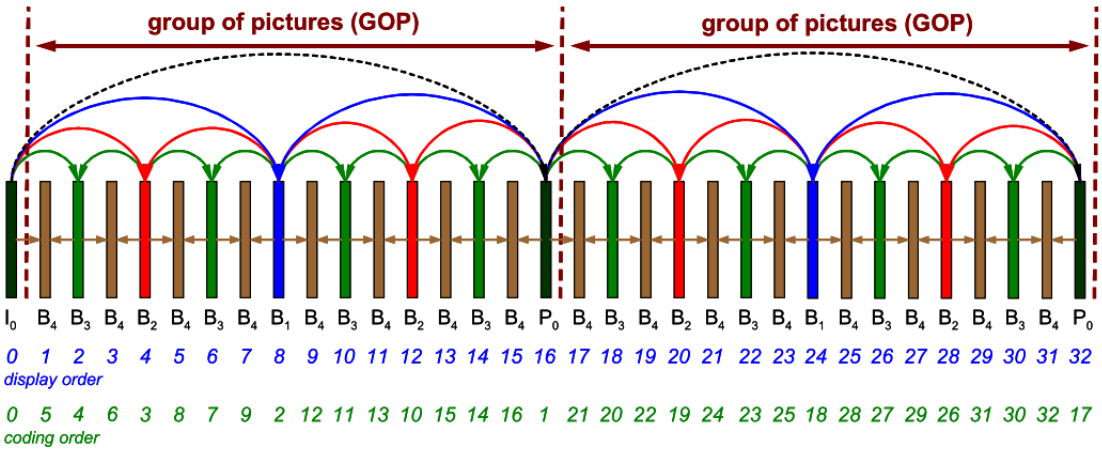


Figure 2.1: Example of I,P,B-frames, group of pictures and decoupled coding order and display order

A coded frame consists of a number of *macroblocks*. Within each frame, a *slice* is made of a set of *macroblocks* in raster-scan order. Generally, an *I-slice* contains *I-macroblock*, a *P-slice* may contain *P* and *I-macroblocks* and *B-slice* may contain *B, P* and *I-macroblocks*. *I-macroblocks* are predicted using intra prediction from decoded samples in the current slice. *P-macroblocks* are predicted using inter prediction from previous reference frame(s) in display order. *B-macroblocks* are predicted using inter prediction from both previous and forward reference frames.

The H.264/AVC standard defines only the syntax of an encoded video bitstream and the decoder. The Encoder and Decoder of the H.264/AVC standard are respectively shown in Figure 2.2 and Figure 2.4. As we can see, there is a "decoding loop" inside the Encoder. So we can say that the Encoder has two data-flow paths: "forward path" and "reconstruction path".

Below is a brief description of data flow in encoder and decoder.

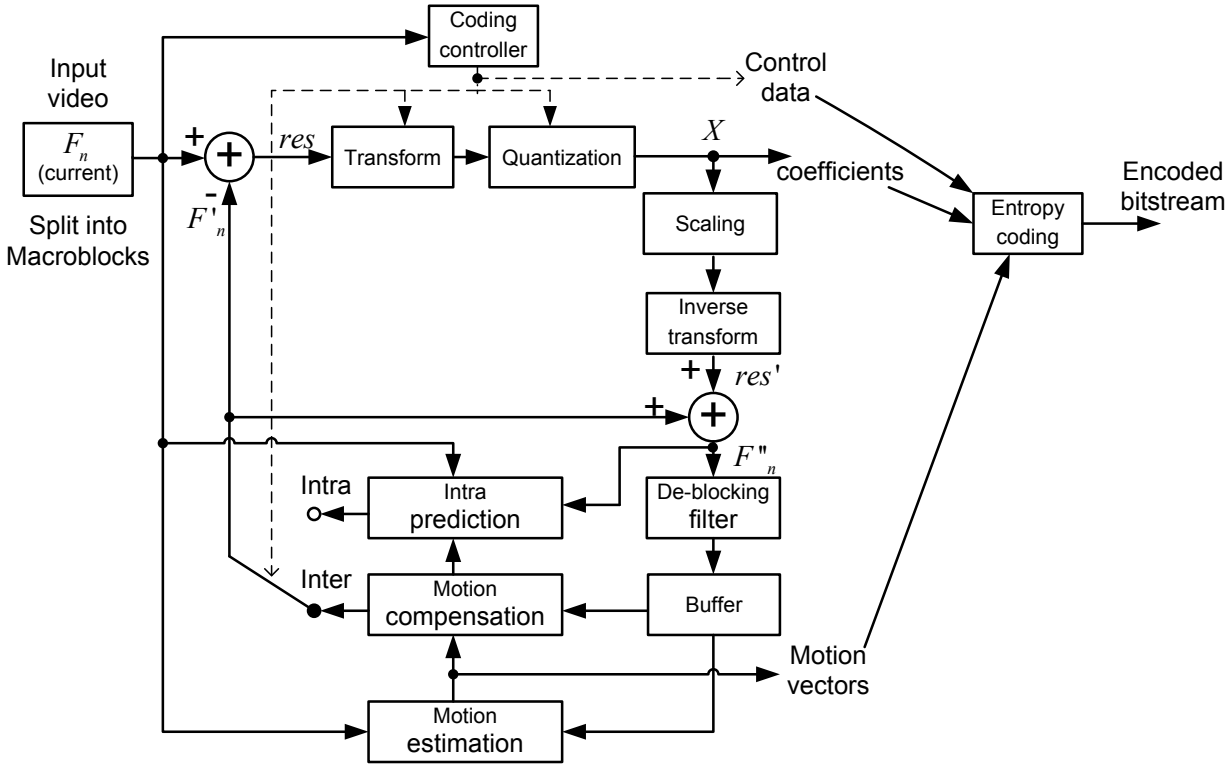
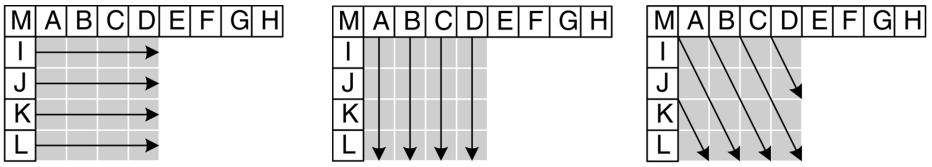


Figure 2.2: Flowchart of the H.264/AVC encoder

In **encoder**, an input frame is processed in macroblock-wise manner, and each macroblock is encoded in *intra* or *inter* mode, which is determined by the coding controller. In *inter mode*, a prediction F'_n is obtained based on motion estimation, motion compensation and previous reconstructed samples (motion estimation and motion compensation will be presented in Section 2.4). Then F'_n is subtracted from the current block F_n to produce a residual res that is transformed and quantized to create X. The coefficients X take two paths: the first path leads to entropy encoding, in which the coefficients X together with the side information required in decoder (e.g. encoding mode, quantizer, and motion vectors) are entropy coded and transmitted as the output of encoder; the second path is the "reconstruction path", where the coefficients X are scaled and inverse transformed to produce a reconstructed residual res' for the current block. The prediction F'_n is added to res' to give the reconstructed block F''_n. After applying the deblocking filter, F''_n is preserved in the buffer for further prediction. In *intra mode*, the only difference is the way of creating the prediction F'_n: intra block prediction is used instead of motion estimation and compensation. Basically, a prediction F'_n is formed based on the samples locate above or on the left. These samples are already encoded and reconstructed without the deblocking filter (see Figure 2.3).

The **decoder** receives the compressed bitstream and obtains a set of quantized coefficients X after entropy decoding. Then quantized coefficients X are scaled and inverse transformed to produce res', which is identical to the res' in "reconstruc-



(a) Horizontal (b) Vertical (c) Vertical right

Figure 2.3: Examples of intra prediction.

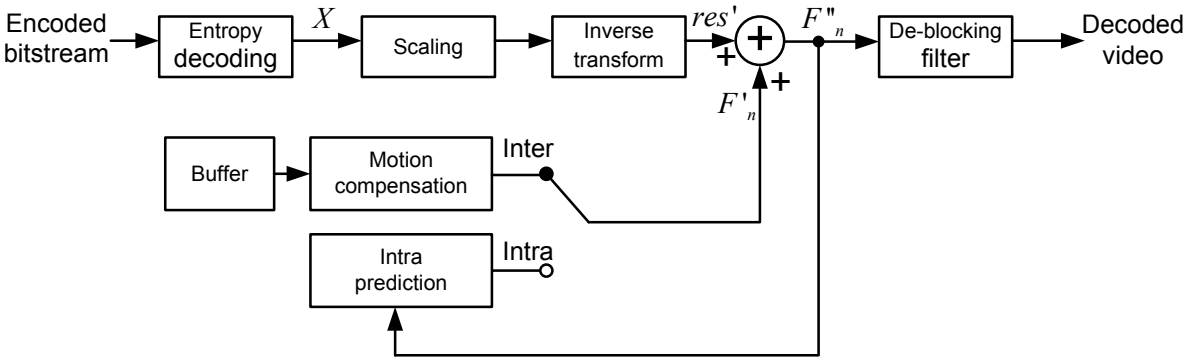


Figure 2.4: Flowchart of the H.264/AVC decoder

tion path" inside encoder. By utilizing the header information obtained from the bitstream, the decoder creates a prediction F'_n in inter or intra mode. Then F'_n is added to res' to produce F''_n which is filtered to create each decoded block.

2.3 Integer Transform and Quantization

Generally the H.264/AVC codec uses block transforms with three different sizes: 4×4, 8×8, and 16×16 transform. All of these three transforms are integer transforms. Some scales multiplication in transform are integrated into quantization. Since the general idea of these three types of transforms are similar, we just discuss the 4×4 *DCT-based transform and quantization* here.

This 4×4 *DCT-based transform* is applied to 4×4 blocks of residual data res. Compared with Discrete Cosine Transform, this DCT-based transform has some advantages [8]:

1. The core part of transform can be implemented by only additions and shifts.
2. Scaling multiplication inside transform can be integrated into quantization, reducing the total number of multiplications.
3. It's an integer transform, so it can produce platform independent results (unlike floating point implementation that has slightly different results by running the same codes in different platforms).

Evolution of the 4×4 DCT-based integer transform from the 4×4 DCT transform is shown below [23]: Discrete Cosine Transform (DCT) is a basis in various lossy compression standards for multimedia signals, such as MP3, JPEG and MPEG. Here, for data X, a 4×4 DCT can be written as:

$$Y = AXA^T = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \begin{bmatrix} X \end{bmatrix} \begin{bmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{bmatrix}, \tag{2.3}$$

where $a = \frac{1}{2}$, $b = \sqrt{\frac{1}{2}} cos(\frac{\pi}{8})$, and $c = \sqrt{\frac{1}{2}} cos(\frac{3\pi}{8})$.

This equation can be modified and expressed in the following form:

$$Y = (CXC^T) \otimes E = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{bmatrix} \begin{bmatrix} X \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & d \\ 1 & d & -1 & -1 \\ 1 & -d & -1 & 1 \\ 1 & -1 & 1 & -d \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix}, \tag{2.4}$$

where CXC^T is the core part of 2D transform. E is a matrix of scaling factors. The symbol $\otimes$ indicates element-wise matrix multiplication. In other words, each element of CXC^T is multiplied by the scaling factor at the same position in matrix E. The constant a, b and c are the same as in Equation 2.3, and $d = \frac{c}{d} \approx 0.414$.

In order to simplify the core part of 2D transform, d is set to 0.5. But b needs to be modified to ensure that the transform remains orthogonal. So, these modified constants are as follows:

$$a = \frac{1}{2}, \quad b = \sqrt{\frac{2}{5}}, \quad d = \frac{1}{2}. \tag{2.5}$$

The core part of 2D transform CXC^T is further simplified by multiplying a scalar 2 to the 2nd and 4th rows of matrix C and the 2nd and 4th columns of matrix C^T. Matrix E is also scaled for compensation. Finally, we get the simplified version of the forward transform:

$$Y = (CXC^T) \otimes E = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} X \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & ab \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix}. \tag{2.6}$$

Therefore, the core part of the transform CXC^T can be implemented with integer arithmetic using only additions and shifts. Note, the result of this 4×4 DCT-based

transform will not be identical to the 4×4 DCT due to changes of factors d and b. Besides, the scaling matrix E can be integrated into quantization since it requires element-wise multiplications (explained in the quantization part).

The *inverse transform* is also defined as arithmetic operations in the H.264 standard [7], and it is illustrated below,

$$X = C_i^T(Y \otimes E_i)C_i = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \left(\begin{bmatrix} Y \end{bmatrix} \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \right) \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix}, \tag{2.7}$$

where Y is the decoded data and is multiplied with scaling matrix E_i, C_i^T and C_i are inverse transform matrices, and X is the inverse transformed data. Note, the factors $\pm\frac{1}{2}$ in C_i^T and C_i can be implemented by a right-shift without a significant accuracy loss.

Quantization is a process of mapping a range of values X to a smaller range of values Y. Since the possible range of a signal is smaller after quantization, it should be possible to represent signal Y with less bits than original signal X. A scalar quantization is used in H.264, and it is a lossy process since it is impossible to determine the exact value of the original fractional number from the rounded integer. The basic forward quantizer can be expressed as:

$$Z_{i,j} = round\Big(\frac{Y_{i,j}}{Qstep}\Big), \tag{2.8}$$

where $Y_{i,j}$ is the transformed coefficients, $Qstep$ is the quantization step, and $Z_{i,j}$ is the quantized data.

The standard H.264 [7] defines 52 values of *quantization steps* (Qstep), indexed by *quantization parameters* (QP) from 0 to 51. Both post- and pre-scaling multiplications are integrated into forward and inverse quantization to avoid floating point operation in transform domain. The forward quantization is given as:

$$Z_{i,j} = round\Big(W_{i,j} \cdot \frac{PF}{Qstep}\Big), \tag{2.9}$$

where $Z_{i,j}$ is the quantized coefficient, $W_{i,j}$ is the unscaled coefficients obtained from the core transform CXC^T, PF is one of three scalars $\frac{ab}{2}$, $\frac{b^2}{4}$ and a^2, according to the position (i, j) in the matrix E (see Equation 2.6), $Qstep$ is the quantization step. Table 2.1 shows parts of $Qsteps$ and QPs used in the H.264 codec.

In order to simplify the arithmetic, the factor $\frac{PF}{Qstep}$ is implemented in the reference software [25] as a multiplication by a factor MF and a right-shift, avoiding

Table 2.1: Parts of quantization steps and quantization parameters used in H.264 codec

QP	0	1	2	3	4	5	6	7	8	9	10
Qstep	0.625	0.6875	0.8125	0.875	1	1.125	1.25	1.375	1.625	1.75	2
QP	24	...	30	...	36	...	42	...	48	...	51
Qstep	10	...	20	...	40	...	80	...	160	...	224

any division operation:

$$Z_{i,j} = round\Big(\big(W_{i,j} \cdot MF\big) \gg qbits\Big), \tag{2.10}$$

where

$$qbits = 15 + floor\left(\frac{QP}{6}\right). \tag{2.11}$$

2.4 Block-based Motion Estimation and Compensation

Inter-frame predictive coding is used to eliminate the large amount of temporal and spatial redundancy that exists in video sequences. It tries to reduce the redundancy between transmitted frames by sending a residual which is formed by subtracting a predicted frame from the current frame. The more accurate the prediction is, the less energy is contained in the residual frame. To get an accurate prediction, good motion estimation and compensation are very important. A widely-used method is *block-based motion estimation and compensation*, which is adopted in various video coding standards, such as H.262, H,263 and H.264.

Block-based motion estimation is the process of searching within an area in the reference frame to find the *best match* for a given block. The reference frame is a previously encoded frame from the sequence and may be before or after the current frame in display order. Motion estimation is carried out by comparing current block with some or all possible blocks in a search window and finding the block which is the *best match*. For a given $M \times N$ block S_n in a frame with frame number n, the process is to find a $M \times N$ block $S_k^{'}$ in a reference frame with the frame number k to minimize,

$$J(\mathbf{v}) = \sum_{(x,y)\in S_n} (s_n(x,y) - s_k^{'}(x+v_x, y+v_y))^2, \tag{2.12}$$

where $\mathbf{v} = (v_x, v_y)$ is a *motion vector*, $|v_x| \leq r$ and $|v_y| \leq r$, r is the search radius, $s_n(x,y)$ is a luminance value of the pixel with coordinate (x,y) in the block S_n, $s_k^{'}(x+v_x, y+v_y)$ is a luminance value of the pixel with coordinate $(x+v_x, y+v_y)$ in the block $S_k^{'}$ (see Figure 2.5).

There are many motion estimation algorithms, and we discuss *full search*, *diamond search* [11] and *hexagon-based search* [12] in this section.

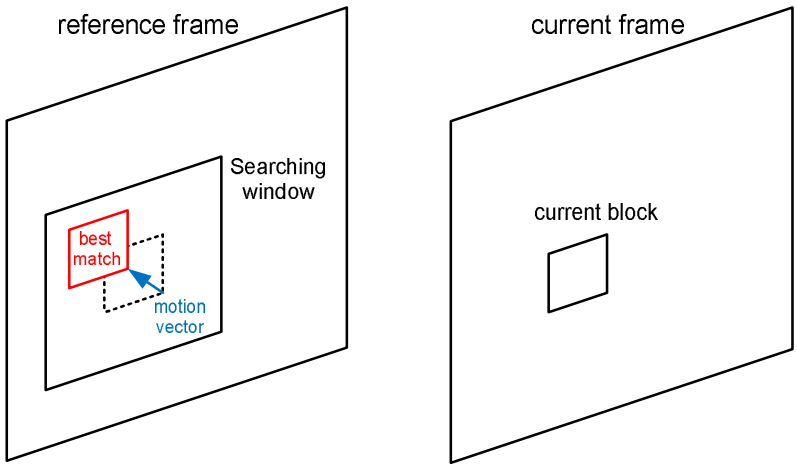


Figure 2.5: Block-based motion estimation

Full search is a commonly used motion estimation method, and for each block it searches exhaustively for the *best match* within a search window. On one hand, it obtains the most precise match since it compares all possible blocks in a reference frame. As a result, the best prediction can be provided, residual will be small and less data need to be transmitted. On the other hand, practical applications of *full search* is limited due to its high computationally intensity. For each block, if only one reference frame is used, the number of search points is,

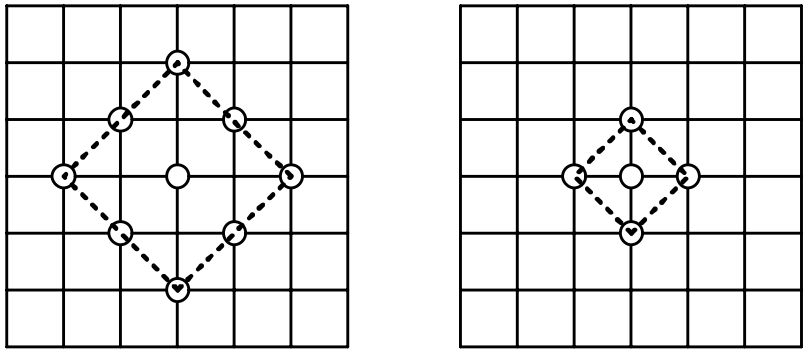
$$N_{FS} = (2 \times r + 1)^2, \tag{2.13}$$

where r is the radius of search window, and the number of search points for full search N_{FS} is proportional to r^2, $N_{FS} \propto r^2$.

In real applications, some fast motion estimation algorithms are commonly used, such as *diamond search* and *hexagon-based search*.

Diamond search [11] is a fast motion estimation method, which employs two search patterns as shown in Figure 2.6. One pattern is called *large diamond search pattern* (LDSP), consisting of 9 check points from which 8 points surround the center one to create a *diamond shape* (Figure 2.6a). The other pattern named *small diamond search pattern* (SDSP), comprising of 5 check points (Figure 2.6b). The main algorithm can be summarized in a few steps:

Step 1. Center a large diamond search pattern (LDSP) at a predefined search window, and compare 9 check points to find minimum block distortion, abbreviated as MBD. If the MBD point is found to be at the center, jump to Step 3; otherwise, go to Step 2.



(a) Large diamond search pattern (b) Small diamond search pattern

Figure 2.6: Two patterns in diamond search algorithm

Step 2. Create a LDSP centred at the position of MBD point from previous search, and search within new check points. If the MBD point among these check points is found to be at the center, jump to Step 3; otherwise, repeat this step.

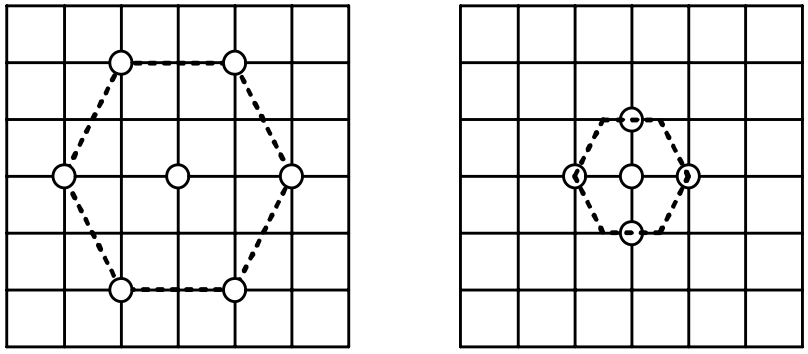
Step 3. Switch the search pattern from large diamond search pattern to small diamond search pattern, and create a small diamond search pattern at the position of MBD point from the previous search. The minimum block distortion among check points is the final solution.

Hexagon-based search [12] is another widely used fast motion estimation algorithm. It has two hexagon-based search pattern as illustrated in Figure 2.7. The first pattern consists of 7 check points with 6 endpoints surrounding the center one to compose a *hexagon* shape (Figure 2.7a). The six endpoints are approximately distributed around the center, which is desirable to achieve the fast search speed [12]. The second pattern (Figure 2.7b) composes of 5 check points (left, right, up, and down dots around the center with distance 1). The Hexagon-based search algorithm is described below:

Step 1. Put a large hexagon search pattern at the center of the predefined search window, and evaluate 9 check points to find minimum block distortion, abbreviated as MBD. If the MBD point is found at the center of the hexagon, jump to Step 3; otherwise, go to Step 2.

Step 2. Create a new large hexagon search pattern centred at the position of MBD point from previous search, and compare new check points. If the MBD is found at the center of this hexagon, jump to Step 3; otherwise, repeat Step 2.

Step 3. Switch the search pattern from the large hexagon pattern to the small one, and center this pattern at the position of the previous MBD point. Compare these 5 check points in a small hexagon pattern, and the MBD among them is the final solution.



(a) Large hexagon search pattern (b) Small hexagon search pattern

Figure 2.7: Two patterns in Hexagon-based search algorithm

Compared with *full search*, both *diamond search* and *hexagon-based search* are much more computationally efficient in motion estimation. If we set N_{DS} and N_{HEXBS} as the average number of search points per block with respect to *diamond search* and *hexagon-based search*, then $N_{DS} \propto r$, $N_{HEXBS} \propto r$ [12] and $N_{FS} \propto r^2$ (Equation 2.13), where r is the radius of the search window. Therefore, compared with *full search*, *diamond search and hexagon-based search* have significantly reduced complexity.

Block-based motion compensation is a process of improving the prediction accuracy by utilizing motion between the current block and reference block(s). Once the best match is found, it becomes the predictor for the current $M \times N$ block. The predictor is subtracted from the current block to produce a residual $M \times N$ block. Then the residual is encoded and transmitted to the decoder, together with the information required in the decoder to repeat the prediction process. *Block-based motion compensation* is a popular technique because it is straightforward, and fits well with rectangular video frames and block-based image transforms. However, there are also some disadvantages. For instance, moving objects in a real video seldom have neat edges that match rectangular boundaries, and objects may move by a fractional number of pixels between frames. Some types of motion, such as rotation and warping, are difficult to compensate by using block-based methods. Therefore, some other methods are used to improve compensation, like variable block-size motion compensation, motion vector with sub-pixel accuracy, and improved coding modes (e.g. skip mode and direct mode) [7].

2.5 Rate-distortion Optimization

The H.264/AVC standard has various candidate modes to code each macroblock, such as Inter Mode 16×16, Inter Mode 16×8, Inter Mode 8×16, Intra modes and so on. A *coding controller* is used to help encoder to make the decision about

applying specific mode for each block. Generally, mode decision depends on two main criteria:

- The mode with least distortion.
- The mode with lowest bitrate.

However, these two criteria cannot be fulfilled at the same time. For example, in block-based motion estimation, smaller size block (8×4, 4×4) may give a lower-energy residual (less distortion) after motion compensation but usually requires a larger number of bits (higher bitrates) to present the motion vectors and choice of partitions, and vice versa.

To solve this problem, a method called *rate-distortion optimization* is employed in the *coding controller* of the H.264/AVC encoder. *Rate-distortion optimization* uses Lagrange multiplier [14] to change this problem into a constrained problem - optimize the distortion subject to bitrates constraint. For a current block s_i coding controller chooses the mode M^* by the following equation:

$$M^* = arg \min_{M \subseteq \{M\}} \big(D(s_i, M) + \lambda_{MODE} \cdot R(s_i, M)\big), \tag{2.14}$$

where $D(s_i, M)$ is the distortion between original macroblock s_i and reconstructed macroblock under coding mode M, $R(s, M)$ is the bitrates of coding block s_i under coding mode M. λ_{MODE} is a Lagrange multiplier, which reveals the tradeoff between distortion and bitrates. For a given QP, the Lagrange multiplier is determined by,

$$\lambda_{MODE} = 0.85 \times 2^{\big(\frac{QP-12}{3}\big)}. \tag{2.15}$$

2.6 Visual Artifacts in Compression

The H.264/AVC standard is widely employed in video coding applications to achieve good compression performance with high visual perception quality [8]. However, the lossy compression techniques used in the H.264/AVC standard may result various visual artifacts in compression, such as *blocking, ringing, blurring, color bleeding, mosquito noise*, etc [15].

Blocking artifacts are defined as the discontinuities found at the adjunct blocks in a decoded frame. There are several causes of *blocking artifacts* in compression. First, each frame is divided into macroblocks and each macroblock can be further divided into variable size blocks in compression process. Second, as it was mentioned in Section 2.3, both transform and quantization used in the H.264/AVC standard are block-based procedures. The coarse quantization of transformed coefficients leads to *blocking artifacts* among adjunct blocks. One example of occurrence of *blocking*

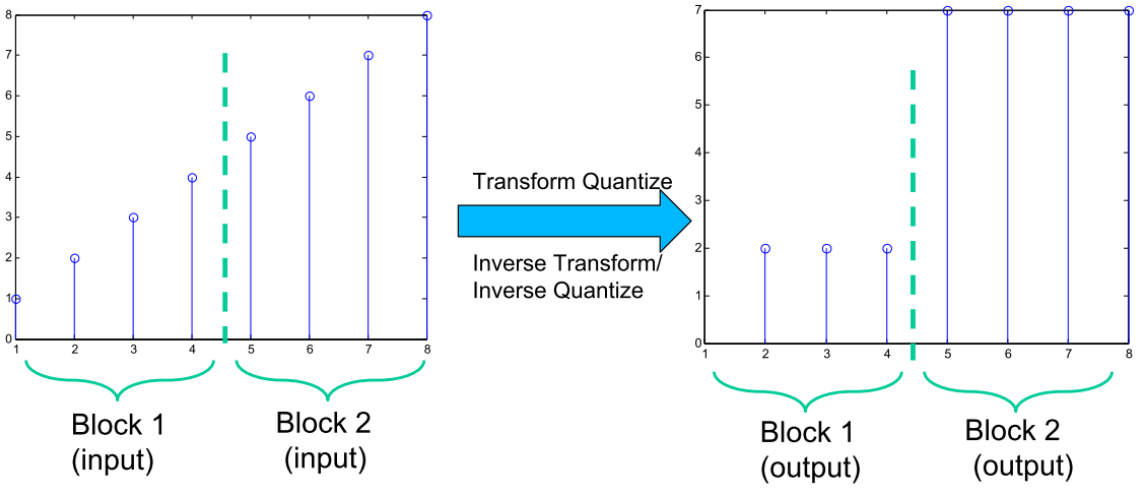


Figure 2.8: Occurrence of blocking artifacts

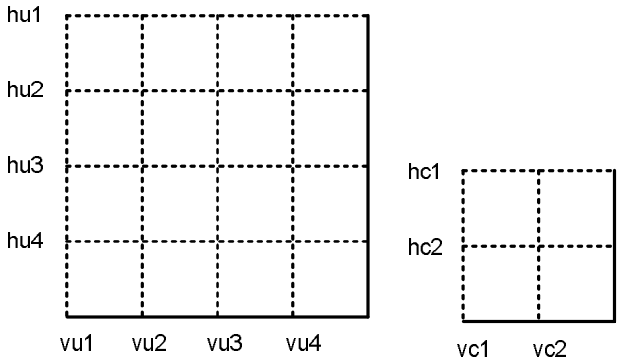
artifacts is shown in Figure 2.8. In the H.264/AVC standard, a deblocking filter (see Section 2.7) is adopted in both encoder and decoder to reduce the *blocking artifacts*.

Ringing artifacts are spurious signals near sharp edges in the frame. They are caused by the loss of some high frequency coefficients. High frequency coefficients play an important role in representation of object edges. But after transformation and coarse quantization some high frequency coefficients are quantized to zeros, resulting in errors in the reconstructed block.

Blurring artifacts are a loss of spatial detail and a reduction in sharpness of edges in frames [15]. They are due to the attenuation of high spatial frequencies, which occurs in quantization (similar to *ringing artifacts*). In H.264/AVC, *blurring artifacts* become obvious when the deblocking filter becomes heavier at low bitrates.

Color bleeding is an artifact where a color component "bleeds" into other areas with different color. Usually, it is caused by color subsampling and heavy quantization of chrominance components [15].

Mosquito noise is an artifact seen mainly in smoothly textured regions as fluctuations of luminance or chrominance around high contrast edges, or moving objects in a video sequence. This effect is related to the high-frequency distortions introduced by both *ringing artifacts* and prediction error produced during motion estimation and compensation [15].



(a) 16×16 luminance (b) 8×8 chrominance

Figure 2.9: Edge filtering order in a macroblock

2.7 Deblocking Filter

The H.264/AVC standard employs a deblocking filter after the inverse transform in both encoder and decoder (see Figures 2.2 and 2.4). The filter is applied to each macroblock to reduce the blocking artifacts without decreasing the sharpness of the frame, so the filtered frame is frequently a more reliable reproduction of the original frame than an unfiltered one. Therefore, video compression performance can be improved by using filtered frame for motion-compensated prediction.

Filtering is applied to vertical or horizontal edges of each block except for slice boundaries. One example of filtering a macroblock is shown in Figure 2.9. First, four vertical edges of luminance components (vu1, vu2, vu3 and vu4) are filtered. Second, it filters four horizontal edges of luminance components (hu1, hu2, hu3 and hu4). Then, two vertical and horizontal edges of chrominance components (vc1, vc2 and hc1, hc2) are filtered. It is also possible to change the filter strength or to disable the filter. Each filtering operation affects up to three samples on either side of the boundary. Figure 2.10 shows four samples on vertical edges and horizontal edges in adjacent blocks p and q. p0, p1, p2, and p3 are four horizontal adjacent pixels in block p, and respectively q0, q1, q2, and q3 are four horizontal adjacent pixels in block q.

The operation of the deblocking filter can be divided into three main steps: *filter strength computation, filter decision* and *filter implementation* [13, 7].

Filter strength for a block is indicated by a parameter named *boundary strength* (BS). The boundary strength depends on the current quantizer, macroblock type, motion vector and gradient of image samples across the boundary. The boundary strength (BS) can be selected as any integer from 0 to 4, according to the rules illustrated in Table 2.2. Note that the BS values for chrominance edges are not independently calculated, and the same values calculated for luminance edges are applied. Application of these rules results in strong filtering at places where there is

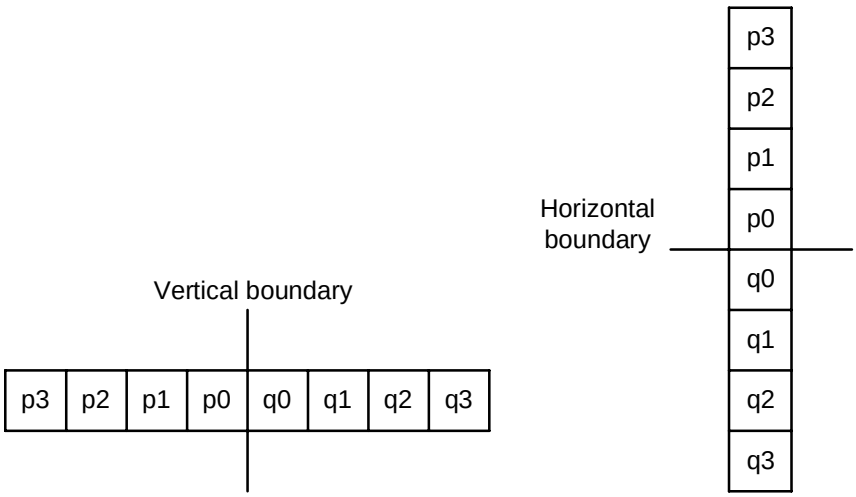


Figure 2.10: Adjacent samples at vertical and horizontal edges

likely to be significant blocking distortion, such as the boundary of the intra coded macroblock or the boundary between blocks which contain coded coefficients.

Table 2.2: Boundary strength (BS) in different conditions

Condition	BS
One of the blocks is Intra coded and the boundary is a macroblock boundary	4
Two blocks are intra coded and the boundary is not a macroblock boundary	3
Both blocks are not intra coded and contain coded coefficients	2
Both blocks are not intra coded and do not contain coded coefficients	1
Both blocks are not intra coded; their motion compensation is from different reference frames *or* their motion vector values that differ by one or more lumimance samples	1
Else	0

Filter decision depends on both boundary strength and gradient of image samples across the boundary. The main reason is that image features with sharp transitions (e.g. object edges) should be preserved rather than to be filtered. When pixels values do not change much across the edge, it should be a smooth region and deblocking filtering is desirable.

If a set of samples (p_2, p_1, p_0 and q_0, q_1, q_3) is filtered, the following conditions must be satisfied.

1. $BS > 0$.

2. $|p_0 - q_0| < \alpha$, $|p_1 - p_0| < \beta$ and $|q_1 - q_0| < \beta$,

where α and β are thresholds defined in the standard [7], and they increase with the average quantization parameter (QP) of the two blocks.

When QP is small, the small transition across the boundary may cause by image features rather than blocking artifacts. The transition should be preserved, so thresholds α and β should be low. When QP is large, blocking artifacts is likely to be much noticeable. Thresholds α and β should be high, so that more boundary samples can be filtered.

Filter implementation can be mainly divided into two modes [13]: one mode is applied when $BS \in \{1, 2, 3\}$; the other mode is a stronger filtering compared to the first mode, and is applied when BS is equal to 4. Those two blocks shown in Figure 2.10 are used as examples for edges filtering, and the filtering process for luminance is described below.
(1) Filtering for edges with $BS \in \{1, 2, 3\}$.
(a) On the boundary, the filtered values $p_0^{'}$ and $q_0^{'}$ are calculated as:

$$p_0^{'} = p_0 + \triangle_0^{'}, \tag{2.16}$$

$$q_0^{'} = q_0 - \triangle_0^{'}, \tag{2.17}$$

where $\triangle_0^{'}$ is calculated in two steps. First, a 4-tap filter is applied with inputs p_1, p_0, q_1 and q_0 to get $\triangle_0$, where

$$\triangle_0 = (4\,(q_0 - p_0) + (p_1 - q_1) + 4) \gg 4. \tag{2.18}$$

Second, the value $\triangle_0$ is clipped to obtain $\triangle_0^{'}$, defined by

$$\triangle_0^{'} = Min\,(Max\,(-c_0, \triangle_0)\,, c_0)\,, \tag{2.19}$$

where c_0 is a parameter that is determined based on a table in H.264 standard [7]. The purpose of clipping is to avoid blurring. Since the intermediate value $\triangle_0$ is directly used in filtering operation, it would result in too much low-pass filtering [13].
(b) The values of p_1 and q_1 are modified only if the following two conditions are satisfied. Otherwise the values of p_1 and q_1 are not changed.

$$|p_2 - p_0| < \beta, \tag{2.20}$$

$$|q_1 - q_0| < \beta. \tag{2.21}$$

If Equation (2.20) is true, then the filtered value of $p_1^{'}$ is calculated as:

$$p_1^{'} = p_1 + \triangle_{p1}^{'}, \tag{2.22}$$

where $\triangle_{p1}^{'}$ is obtained in two steps as well. First, the a 4-tap filter is applied as follows:

$$\triangle_{p1} = (p_2 + ((p_0 + q_0 + 1) \gg 1) - 2p_1) \gg 1. \tag{2.23}$$

Second, similar with clipping process in (a), this value $\triangle_{p1}$ is clipped by:

$$\triangle_{p1}^{'} = Min\left(Max\left(-c_1, \triangle_{p1}\right), c_1\right), \tag{2.24}$$

where c_1 is also a parameter that is determined based on a table in H.264 standard [7].

If Equation (2.21) is true, filtered value $q_1^{'}$ is calculated in the same way, by substituting q_2 and q_1 for p_2 and p_1 respectively. As for a chrominance, only the values of p_0 and q_0 are modified, and there is no need to clip the value.
(2) Filtering for edges with $BS = 4$
(a) If $|p_2 - p_0| < \beta$ and $|p_0 - q_0| < (\alpha \gg 2) + 2$, then,

$$p_0^{'} = (p_2 + 2p_1 + 2p_0 + 2q_0 + q_1 + 4) \gg 3, \tag{2.25}$$

$$p_1^{'} = (p_2 + p_1 + p_0 + q_0 + 2) \gg 2, \tag{2.26}$$

$$p_2^{'} = (2p_3 + 3p_2 + p_1 + p_0 + q_0 + 4) \gg 3, \tag{2.27}$$

else only p_0 is modified according to the following equation, and p_1 and p_2 are left unchanged:

$$p_0^{'} = (2p_1 + p_0 + q_1 + 2) \gg 2. \tag{2.28}$$

(b) Similarly, the values of the q block are modified, by substituting Equation $|q_2 - q_0| < \beta$ for $|p_2 - p_0| < \beta$ and replacing p_i by q_i and vice versa.

2.8 Influence of Source Noise to Compression Performance

Digital video sequences can easily be corrupted by noise during acquisition, recording, processing or transmission. Figure 2.11 describes an original video $x(t)$ that is corrupted by noise $n(t)$ to produce a noisy video $y(t)$. Then $y(t)$ is given to the H.264/AVC codec.

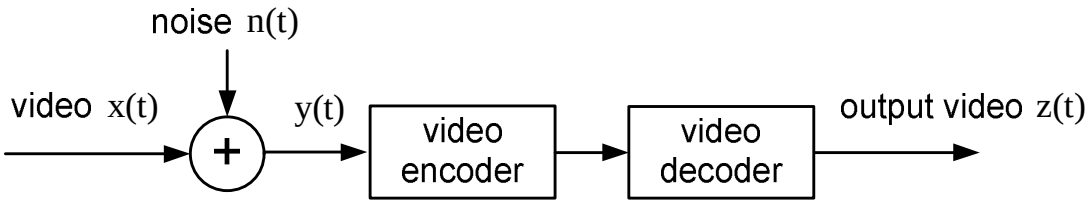


Figure 2.11: Compress noisy video

In this case, $y(t)$ can be expressed as,

$$y(t) = x(t) + n(t). \tag{2.29}$$

As for the noise $n(t)$, one of the most common mode is a *Gaussian noise*. *Gaussian noise* is a common noise in images or videos and has Gaussian probability density

function. Generally, the noise component n is defined as an independent and identically distributed zero-mean Gaussian random variable with a variance σ^2. The zero-mean Gaussian noise model can be expressed as,

$$n(\cdot) \sim N\left(0, \sigma^2\right). \tag{2.30}$$

In order to see the influence of source noise in the H.264/AVC codec, two experiments have been carried out.

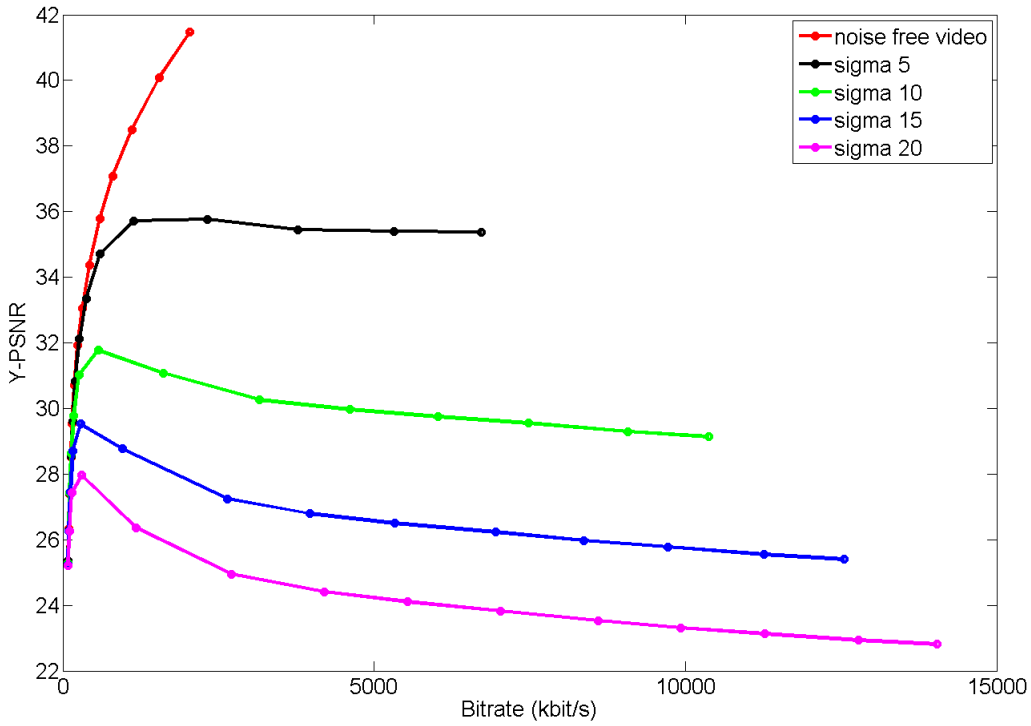


Figure 2.12: Rate-distortion curves for noisy video compression: video *foreman* corrupted with different level of Gaussian noise is compressed by the H.264/AVC codec with different QPs ($QP \in \{20, 22, 24, ...46\}$).

In the first experiment, we compare the system outputs $z(t)$ for an input video sequence corrupted by Gaussian noise at different variances. Video *foreman* (352×288) is corrupted by Gaussian noise with different variances ($\sigma^2 \in \{5^2, 10^2, 15^2, 20^2\}$), and these noisy videos are compressed by the H.264/AVC reference software JM V.17.1 with different quantization parameters ($QP \in \{20, 22, 24, ...46\}$). The rate-distortion curves are shown in Figure 2.12.

This Figure 2.12 reflects that noise contained in the video decreases the compression performance of the codec, and the stronger noise is the lower PSNR values are. The curves of noisy videos show that the bitrates increases very fast as the QP decreases. In addition, when the noise level is high and QP is low, we find that as the bitrates are increased, the PSNR values get smaller. This kind of feature is not acceptable in real applications, and it should be avoided. The reason for this feature is as follows: when QP is high, the video is highly quantized. This process

smoothens the frames and decreases the noise. However when QP is small, the codec tries to preserve the noise in the corrupted video.

Table 2.3: Percentages of inter and intra coded macro-blocks when video *hall* is corrupted by Gaussian noise with different variances

Mode \ Sigma	0	5	10	15	20
Inter coded blocks	98.4%	94.5%	54.4%	38.8%	28.2%
Intra coded blocks	1.6%	5.5%	45.6%	61.2%	71.8%

In the second experiment, the percentages of inter and intra coded macroblocks is recorded while compressing a input video sequence corrupted by Gaussian noise at different variances. *Hall* (352×288) is a test video with a static background and it is corrupted by Gaussian noise with different variances ($\sigma^2 \in \{5^2, 10^2, 15^2, 20^2\}$). These videos are compressed by the H.264/AVC reference software JM V.17.1 at QP=28. The results are shown in Table 2.3.

This table tells that the number of intra coded macroblocks increases with sigma. Our analysis shows that with the increase of noise, motion estimation does not work well. The coding controller finds the energy contained in residual which is equal or even higher than the original block. So it chooses intra mode to code blocks even if the video has a static background between frames.

2.9 Conclusion

The H.264/AVC is an excellent video coding standard in terms of both coding efficiency and flexibility for different applications. However, the H.264/AVC standard does not perform well in the presence of noise, and the stronger noise is the lower PSNR values are. In addition, motion estimation does not work well when the noise level is high in the video, and the codec tends to code macroblocks at intra mode. Furthermore, the bitrates increase very fast due to many intra coded data that are transmitted. Therefore, we need to find some methods to reduce the noise level in videos before compression, and we discuss some common video denoising methods in the next chapter.

Chapter 3

Video Denoising using Block-Matching and 3D filtering

3.1 Introduction

Digital video sequences are almost always corrupted by noise during acquisition, recording, processing or transmission. The noise in video sequences not only degrades the subjective quality, but also affects the effectiveness of further processing (Section 2.8). Therefore, video denoising is important, because it improves the quality of perceived video sequences and enhances subsequent processes in video coding (e.g. motion estimation).

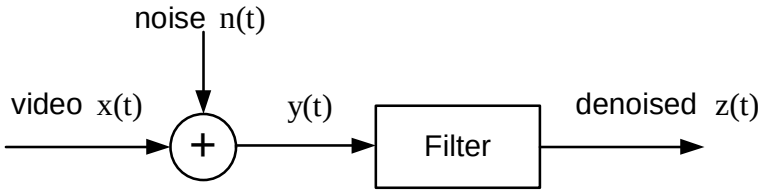


Figure 3.1: Typical flowchart of video denoising

A general case of original video $x(t)$ corrupted by noise $n(t)$ is shown in Figure 3.1, and the noisy video can be expressed as:

$$y(t) = x(t) + n(t). \tag{3.1}$$

The task of video denoising is to filter corrupted video sequence $y(t)$ so as to minimize the difference between filtered output $z(t)$ and original video $x(t)$. The noise $n(t)$ represents the *Gaussian noise* (see in Section 2.8).

In this chapter, contents are organized as follows: Section 3.2 gives a brief overview of basic video denoising methods and then Section 3.3 discusses the Video Block-Matching and 3D filtering algorithm. Furthermore, Section 3.4 proposes a real-time implementation of the simplified version of VBM3D.

3.2 Classification of Video denoising Algorithms

A large number of research has been carried out on video restoration and enhancement, and many different algorithms and principles have been presented during the past several decades ([26]-[39]). These approaches basically can be classified into four categories:

- Spatial domain video denoising;
- Temporal domain video denoising;
- Spatio-temporal domain video denoising;
- Transform domain video denoising.

Many different kinds of filters are designed based on various denoising strategies, then some of the denoising methods are illustrated here:

Spatial domain denoising is a way of utilizing spatial correlation of video content to suppress noise. It is normally implemented with a weighted local 2D or 3D windows, and the weights can be either fixed or adapted based on the image content. 2D Wiener filter [27], 2D Kalman filter [28], non-local means [29] and wavelet shrinkage [30] denoising methods were proposed in the last few decades. However, spatial-only denoising is rarely considered in real applications, as it often leads to visible artifacts.

Temporal domain denoising is an approach of exploiting temporal correlations to reduce noise in a video sequence. A video sequence contains not only spatial correlation but also temporal correlation between consecutive frames. Temporal denoising methods [31, 32] utilize temporal correlations to achieve video denoising. Normally, motion estimation methods, which can be based on block matching [1, 36] or optical flow [38, 39], are employed to find the prediction of the reference block. For each reference block, its temporal predictions are combined with the block itself to suppress noise.

Spatio-temporal denoising exploits both spatial and temporal correlations in video sequence to reduce noise. It is generally agreed that in many real video applications, spatio-temporal filtering performs better than temporal filtering [26], and the best performance can be achieved by exploiting information from both past and future frames. 3D Kalman filter [33], spatio-temporal shrinkage [34], 3-D non-local means [35] and VBM3D [1] are some spatio-temporal denoising methods.

Transform domain denoising methods first decorrelate the noisy signal using a linear transform (e.g. DCT or wavelet transform [37]), and then recover the transform coefficients (e.g. by hard thresholding [1]). Then this signal is subjected to inverse transform to get the signal back to spatial domain. Typically, transform domain methods are used together with temporal or spatial domain denoising methods.

3.3 Video Block-Matching and 3D filtering

3.3.1 General Scheme of the Video Block-Matching and 3D filtering

As it was mentioned in the previous section, spatio-temporal domain filtering, transform domain filtering, and motion information can be used together to improve the filtering performance. There are some filtering approaches that exploit correlations using combined filtering strategies. In this section, we present Video Block-Matching and 3D filtering [1], which is one of the best current video denoising filters.

Video Block-Matching and 3D filtering is an effective video denoising method based on highly sparse signal representation in local 3D transform domain [1]. It is an extension of Block-Matching and 3D filtering for images [16], and achieves state-of-the-art denoising performance in terms of both peak signal-to-noise ratio and subjective visual quality.

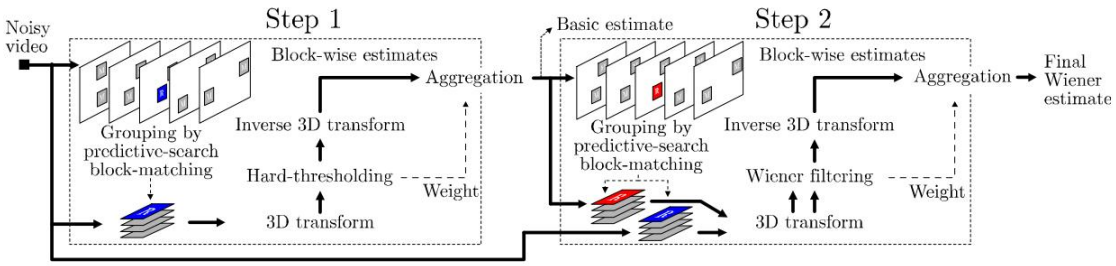


Figure 3.2: Flowchart of VBM3D denoising algorithm. The operation enclosed by dashed lines are repeated for each reference block.

The general procedure consists of the following two steps (see Figure 3.2). In the first step, a noisy video is processed in raster scan order and a block-wise manner. As for each reference block, a 3D array is grouped by stacking blocks from consecutive frames which are similar to the currently processing block. In grouping, a predictive-search Block-Matching is used. Then a 3D transform-domain shrinkage (hard-thresholding in the first step, and Wiener filtering in the second step) is applied to each of the grouped 3D array. Since the estimates of those obtained blocks are always overlapped, they are aggregated by a weighted average to obtain an intermediate estimate. In the second step, the intermediate estimate from the first step is used together with a noisy video for grouping and applying 3D collaborative empirical Wiener filtering.

The VBM3D algorithm has three important concepts: *grouping*, *collaborative filtering*, and *aggregation*. Prior to the VBM3D algorithm, let us discuss these three important concepts.

3.3.2 Grouping

The term *grouping* refers to the concept of collecting similar d-dimensional fragments of a given signal into a d+1-dimensional data structure. In the case of a video, the fragments can be any of the 2D blocks, and a group is a 3D array formed by stacking together similar blocks from consecutive frames (e.g. besides current frame, search among N forward and N backward frames). Similarity between blocks is computed using the l^2-norm of the difference between two blocks. In order to achieve efficient grouping, a *predictive-search Block-Matching* [1] is used to efficiently find similar blocks. The main idea of this method is to perform a full-search within a $N_S \times N_S$ window in current frame to obtain the N_B best-matching blocks. Then in following N_{FR} frames, it inductively searches for another N_B best-matching blocks within a smaller window size of $N_{PR} \times N_{PR}$ ($N_{PR} \ll N_S$). The window centers at the same position of the previous block. The benefit of grouping is to enable the use of high dimensional filtering, which utilizes the potential similarity between grouped blocks.

3.3.3 Collaborative filtering

Once a 3D array is obtained from grouping, *collaborative filtering* can be used to exploit both spatial correlation inside single block and the correlation between grouped blocks. Then it is followed by a shrinkage in the transform domain. The collaborative filtering is executed as following steps:

- Perform a linear 3-dimensional transform (e.g. 3D DCT) to the group.
- Shrink transformed coefficients by hard-thresholding or Wiener filtering to attenuate noise.
- Invert linear transform (e.g. inverse 3D DCT) to obtain estimates of grouped blocks.

The benefit of Collaborative filtering is to utilize both kinds of correlations to produce a sparse representation of the group, and the sparsity is desirable for effective shrinkage in noise attenuation.

3.3.4 Aggregation

In general, estimates of denoised 3D groups can be overlapped. In other words, there can be multiple estimates obtained from different filtered 3D groups but have exactly the same coordinates. This leads to an over-complete representation of original video sequence. To produce fine representation of the original video, *aggregation* is carried out to produce estimates of filtered 3D groups by a weighted averaging with adaptive weights.

3.3.5 Algorithm

In VBM3D filtering, we consider a noisy video as:

$$z(x) = y(x) + n(x), x \in X \subset \mathbb{Z}^3, \tag{3.2}$$

where y is the true video signal, $n(\cdot) \sim N(0, \sigma^2)$ is i.i.d. zero-mean Gaussian noise with variance σ^2, σ is assumed a prior known value, x is a 3D coordinate that belongs to the three-dimensional spatio-temporal domain $X \subset \mathbb{Z}^3$, and it can be expressed as following:

$$x = [x_1, x_2, t]. \tag{3.3}$$

The first and second coordinates are the 2D spatial coordinates in one video frame, and the third coordinate $t \subset \mathbb{Z}$, which indicates the frame number.

As for VBM3D algorithm, it can be divided into two steps [1], which are described below.

Step 1. Produce a basic estimate using grouping and collaborative hard-thresholding.

Each reference block Z_{x_R} with $x_R \in X$ of size $N_{ht} \times N_{ht}$ is taken from both horizontal and vertical directions with a step length of N_{step}. VBM3D groups a set of similar blocks by using predictive-search Block-Matching (PS-BM),

$$S^{ht}_{x_R} = PS\text{-}BM(Z_{x_R}), \tag{3.4}$$

where Z_{x_R} indicates a block whose upper-left corner is at x_R (similar notation is used for others), $S^{ht}_{x_R}$ are similar blocks for Z_{x_R}. All these similar blocks are grouped to form a set:

$$Z_{S^{ht}_{x_R}} = \{Z_{x_R} : x \in S^{ht}_{x_R}\}. \tag{3.5}$$

Then a collaborative hard-thresholding is carried out with threshold $\lambda_{3D}\sigma$ to produce an estimates of the set $Z_{S^{ht}_{x_R}}$:

$$\widehat{Y}_{S^{ht}_{x_R}} = T^{-1}_{3D}(HARD\text{-}THR(T_{3D}(Z_{S^{ht}_{x_R}}), \lambda_{3D}\sigma)), \tag{3.6}$$

where $\widehat{Y}_{S^{ht}_{x_R}}$ is a set with filtered blocks and can be expressed as:

$$\widehat{Y}_{S^{ht}_{x_R}} = \{\widehat{Y}^{x_R}_x : x \in S^{ht}_{x_R}\}. \tag{3.7}$$

After that the basic estimate $\widehat{y}^{basic}$ is calculated by aggregation of blockwise estimates $\widehat{Y}^{x_R}_x$ according to the formula,

$$\widehat{y}^{basic} = \frac{\sum_{x_R \in X} \sum_{x \in S^{ht}_{x_R}} w^{ht}_{x_R} \widehat{Y}^{ht,x_R}_x}{\sum_{x_R \in X} \sum_{x \in S^{ht}_{x_R}} w^{ht}_{x_R} \chi_x}, \tag{3.8}$$

where χ_x: $X \to \{0, 1\}$ is the characteristic function of the square support of a block located at $x \in X$, and $w^{ht}_{x_R}$ is the weight for the current block. This weight $w^{ht}_{x_R}$ is obtained by:

$$w^{ht}_{x_R} = \frac{1}{\sigma^2 N^{x_R}_{har}} W_{2D}, \tag{3.9}$$

where $N_{har}^{x_R}$ is the number of non-zero coefficients after hard-thresholding $T_{3D}(Z_{S_{x_R}^{ht}})$, and $N_{har}^{x_R} > 0$ because the DC value is always reserved, ensuring that division by zero never happens in aggregation, and W_{2D} is a $2D$ Kaiser window of size $N_{ht} \times N_{ht}$ which is used for reducing border effect.

Step 2. Obtain the final estimate by grouping within the basic estimate and collaborative Wiener filtering that uses the spectra of the corresponding groups from the basic estimates.

For each block $\widehat{Y}_{x_R}^{basic}$ with the size of $N_{wie} \times N_{wie}$, algorithm applies predictive-search Block-Matching:

$$S_{x_R}^{wie} = PS\text{-}BM(\widehat{Y}_{x_R}^{basic}), \tag{3.10}$$

and based on the set $S_{x_R}^{wie}$, two three-dimensional arrays are formed:

$$\widehat{Y}_{S_{x_R}^{wie}}^{basic} = \{\widehat{Y}_{x_R}^{basic} : x \in S_{x_R}^{wie}\}, \tag{3.11}$$

$$Z_{S_{x_R}^{wie}} = \{Z_x : x \in S_{x_R}^{wie}\}. \tag{3.12}$$

Then collaborative filtering is performed in second step by an empirical Wiener filtering, and it is defined as,

$$\widehat{Y}_{S_{x_R}^{wie}} = T_{3D}^{-1}\left(T_{3D}(Z_{S_{x_R}^{wie}})\frac{\left(T_{3D}(\widehat{Y}_{x_R}^{basic})\right)^2}{\left(T_{3D}(\widehat{Y}_{x_R}^{basic})\right)^2 + \sigma^2}\right). \tag{3.13}$$

The final estimate ($\widehat{Y}^{final}$) is produced by aggregation of those overlapped estimates. It is given by,

$$\widehat{y}^{final} = \frac{\sum_{x_R \in X}\sum_{x \in S_{x_R}^{wie}} w_{x_R}^{wie}\widehat{Y}_x^{wie,x_R}}{\sum_{x_R \in X}\sum_{x \in S_{x_R}^{wie}} w_{x_R}^{wie}\chi_x}, \tag{3.14}$$

with the weight of

$$w_{x_R}^{wie} = \sigma^{-2}\left\|\frac{\left(T_{3D}(\widehat{Y}_x^{basic})\right)^2}{\left(T_{3D}(\widehat{Y}_x^{basic})\right)^2 + \sigma^2}\right\|_2^{-2} W_{2D}, \tag{3.15}$$

where $\|\cdot\|_2$ denotes l^2-norm, and W_{2D} is a $2D$ Kaiser window of size $N_{wie} \times N_{wie}$.

3.3.6 Complexity Analysis

In this analysis, complexity is measured based on the number of basic arithmetic operations, however other factors, such as memory consumption and the number of memory accesses have not been considered.

The complexity of VBM3D (C_{VBM3D}) consists of the complexity of hard-thresholding stage (C_{VBM3D}^{ht}) and Wiener-filtering stage (C_{VBM3D}^{wiener}) [19] and the descriptions of the parameters in the following equations are shown in Table 3.1:

$$C_{VBM3D} = C_{VBM3D}^{ht} + C_{VBM3D}^{wiener}. \tag{3.16}$$

Hard-thresholding stage, for each processed block, at most M similar blocks are extracted within the search window of size $N_S \times N_S$ and stacked together as a group. Then a 3D transform and hard-thresholding are applied to the 3D group. Finally, the basic estimate is obtained by aggregating the inversed coefficients. Thus the complexity of hard-thresholding stage can be expressed as:

$$C_{VBM3D}^{ht} = T\frac{n}{N_{step}^2}\Big(\underbrace{\big(N_S^2 + zN_BN_{PR}^2\big)3N^2}_{\texttt{Grouping}} + \underbrace{2\big(2MC_{(N,N,N)} + C_{(M,M,N^2)}\big)}_{\texttt{3D forward and inverse transform}} + \underbrace{MN^2}_{\texttt{Aggregation}} \Big). \tag{3.17}$$

Wiener-filtering stage, the most processes are the same as those in hard-thresholding stage, but two groups instead of one need to be transformed. Element-wise multiplications are applied for obtaining coefficients shrinkage, which involves a set of weights in computation and requires 6 arithmetic operations per pixel:

$$C_{VBM3D}^{wiener} = T\frac{n}{N_{step}^2}\Big(\underbrace{\big(N_S^2 + zN_BN_{PR}^2\big)3N^2}_{\texttt{Grouping}} + \underbrace{4\big(2MC_{(N,N,N)} + C_{(M,M,N^2)}\big)}_{\texttt{3D forward and inverse transform}} + \underbrace{6MN^2}_{\texttt{Shrinkage}} + \underbrace{MN^2}_{\texttt{Aggregation}} \Big). \tag{3.18}$$

Table 3.1: Parameters involved in the VBM3D complexity analysis

Parameter	Description
T	Total number of frames in a video
n	Number of pixels in a frame
N	Length of the 2-D block
z	Length of temporal search window in grouping
N_S	Length of the spatial search window
N_{step}	Sliding step to process every next reference block
M	Number of blocks in a grouped 3-D array
$C_{(a,b,c)}$	Numeric operation required by a multiplication between two matrices of size $a \times b$ and $b \times c$

3.3.7 Practical Results

In this section, we present and discuss some experimental results obtained by VBM3D. The results of VBM3D filtering on three standard videos *foreman*[1] (352 × 288), *vassar*[2] (640 × 480) and *ballroom*[2] (640 × 480) corrupted by Gaussian noise with variance of 20^2 are shown in Table 3.2. Comparisons of subjective visual quality between original, noisy and denoised frames are illustrated in Figure 3.3.

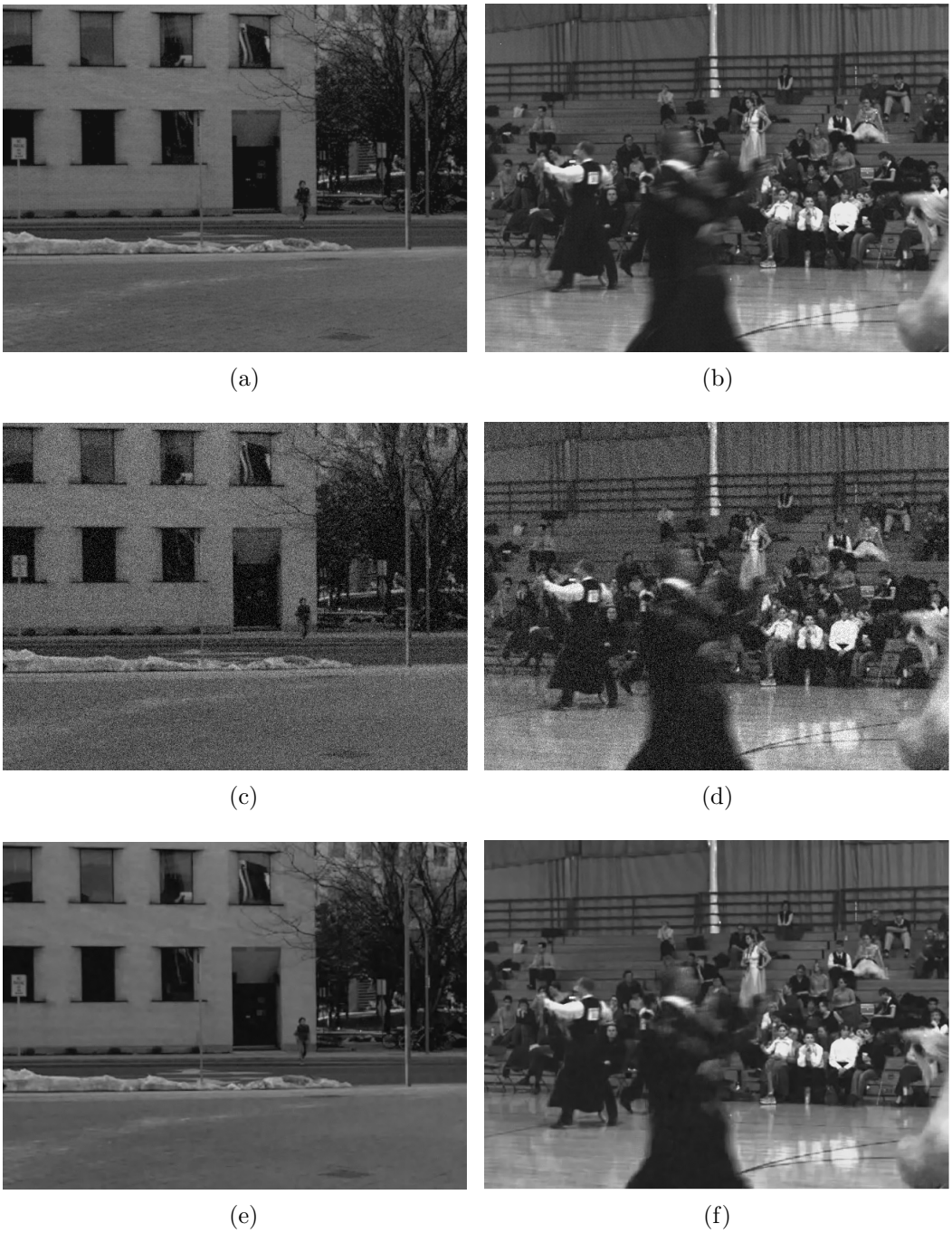
[1]Video *foreman* is from http://media.xiph.org/video/derf/.

[2]Video *vassar* and *ballroom* are from http://www.merl.com/pub/avetro/mvc-testseq/orig-yuv/.

Table 3.2: Performance of VBM3D among different test videos sequences corrupted by Gaussian noise with $\sigma = 20$ in computer with Intel Core 2 Duo 3GHz and 3.2GB of RAM.

$\sigma = 20$	foreman	vassar	ballroom
Resolution	352×288	640×480	640×480
Noisy (dB)	22.11	22.11	22.11
denoised (dB)	34.44	36.28	35.84
Speed (fps)	3.47	1.02	1.06

On one hand, the results reflect that VBM3D filter achieves state-of-the-art denoising performance in terms of both peak signal-to-noise ratio and subjective visual quality. On the other hand, due to the high complexity of the algorithm, the speed at which current implementation of VBM3D executes makes it hard to meet the real-time requirements. We define the real-time requirements as: filter has at least 25 fps for processing frames with resolution of 640×480 under computer platform with Intel Core 2 Duo 3 GHz and 3.2 GB of RAM. However, the speed of current implementation is only 1.02 fps for *vassar* (640×480) and 1.06 fps for *ballroom* (640×480). To solve this problem, we simplify the VBM3D algorithm and propose a fast integer implementation in the next section.



(a) (b)

(c) (d)

(e) (f)

Figure 3.3: Examples of VBM3D filtering: two test videos *vassar* and *ballroom* are corrupted by Gaussian noise with $\sigma = 20$, and (a),(c),(e), respectively are original, noisy and denoised frames for *vassar*, and (b),(d),(f), respectively are original, noisy and denoised frames for *ballroom*.

3.4 Real-time Implementation of the Video Block-Matching and 3D filtering

VBM3D achieves state-of-the-art denoising performance in terms of both peak signal-to-noise ratio and subjective visual quality. However the speed at which the current implementation performs is slow (around 1 fps for video (640×480) in computer with Intel Core 2 Duo 3GHZ and 3.2GB of RAM). This speed is far from the requirements of real-time applications, at least 25 fps for video (640×480) in computer with Intel Core 2 Duo 3GHZ and 3.2GB of RAM.

In order to accelerate the filter and preserve the good denoising performance, we propose to use proper complexity-scalable filter $C(F_i)$ according to this platform and find optimal filtering parameters F^*. The optimization task can be formulated by the following equations:

$$\begin{cases} F^* = \arg \min\limits_{F \subseteq \{F\}} \sum\limits_i D(F_i), \\ \sum\limits_i C(F_i) \leq C^{Max}, \end{cases} \tag{3.19}$$

where F_i are filtering parameters for the noisy frame i, respectively $D(F_i)$ is the distortion between denoised and the original frame after filtering current noisy frame with parameters F_i, $C(F_i)$ is the complexity for filtering frame i with parameters F_i, C^{Max} is complexity restriction.

Recall the concept of *complexity* from Section 2.1.4. Complexity can be defined as the number of basic arithmetic operations per pixel or per time interval. But in real applications, the number of operations does not show the full complexity because it does not include memory accesses and logical operations. In this thesis, complexity is defined as the number of frames which can be filtered in one second for given computer platform and frame resolution. Following this definition, we set $C^{Max} = 25$ fps for frame resolution 640×480 in computer platform with processor of Intel Core 2 Duo 3GHz and 3.2GB of RAM.

In this optimization task, two approaches are used,

1. Simplify VBM3D filter by using only the most influential parts for noise attenuation.

2. Propose a fast integer implementation of the simplified VBM3D.

In the first approach, we need to find which parts of VBM3D are most influential for noise attenuation. The VBM3D filter can be divided into two steps, and each step has several sub-steps as presented in Section 3.3. Some experiments are carried out to find the noise attenuation ability of each sub-step. We "turn off" one sub-step and "turn on" all the other parts of the VBM3D filter, then we record filter performance.

After experiments, we find that temporal correlation contributes more than spatial correlation in noise reduction. As a result, we choose only to use temporal search, temporal transform and hard-thresholding in the first step, but to remove Wiener filtering part due to its high complexity. At the same time, we change the values of two parameters in the filter settings. One is the number of temporal searching frames, reducing from 9 to 5. In other words, it only searches the current frame, the two previous and the two following frames. The other is N2 (maximum length of the 3-dimension transform), using 4 instead of 8. By doing this, the computational complexity of VBM3D is considerably decreased. The comparison of the *standard VBM3D*[1] and the *simplified VBM3D*[2] algorithm is shown in Table 3.3.

Table 3.3: Comparison of the standard VBM3D and the simplified VBM3D algorithm

Filters		Standard VBM3D	Simplified VBM3D
Step 1	Spatial search	+	–
	Spatial transform	+	–
	Temporal search	+	+
	Temporal transform	+	+
	Hard thresholding	+	+
Step 2	Spatial search	+	–
	Spatial transform	+	–
	Temporal search	+	–
	Temporal transform	+	–
	Wiener filtering	+	–
Temporal searching frames		9	5
N2 (maximum length of the haar transform)		8	4

Two video sequences *vassar* (640×480) and *ballroom* (640×480) are used in our experiments, and both of these two videos are corrupted by Gaussian noise with different variances. The comparison of performance between the standard VBM3D and the simplified VBM3D is shown in Table 3.4. As we can see from Table 3.4, for small sigma value, such as 5, even though the performance of the simplified VBM3D is not as good as the standard VBM3D due to simplification of the algorithm, the simplified VBM3D still has good denoising ability. This is because human eyes usually cannot tell the difference among images which have PSNR values above 37

[1] *Standard VBM3D* is the Matlab version with default settings from http://www.cs.tut.fi/ foi/GCF-BM3D/index.html.

[2] *Simplified VBM3D* means a simplified version by switching off some features of the standard VBM3D.

Table 3.4: Comparison of the performance between the standard VBM3D and the simplified VBM3D for denoising video sequences *vassar* and *ballroom* which are corrupted by Gaussian noise with different variances, in computer platform with Intel Core 2 Duo 3GHz and 3.2GB of RAM

σ/PSNR	Resolution: 640 × 480 Number of frames: 250		Standard VBM3D	Simplified VBM3D
5/34.13	vassar	Denoised (dB)	40.74	38.36
		Speed (fps)	1.09	**7.22**
	ballroom	Denoised (dB)	41.44	37.74
		Speed (fps)	1.11	**7.29**
10/28.12	vassar	Denoised (dB)	38.21	33.67
		Speed (fps)	1.14	**7.67**
	ballroom	Denoised (dB)	38.69	33.15
		Speed (fps)	1.09	**7.22**
15/24.63	vassar	Denoised (dB)	36.57	30.23
		Speed (fps)	1.10	**7.28**
	ballroom	Denoised (dB)	36.71	29.97
		Speed (fps)	1.13	**7.12**
20/22.18	vassar	Denoised (dB)	35.32	27.84
		Speed (fps)	1.13	**7.28**
	ballroom	Denoised (dB)	35.20	27.24
		Speed (fps)	1.15	**7.01**

dB. As the increase of sigma values, the simplified VBM3D performs worse than the standard VBM3D. But in general the simplified VBM3D improves the PSNR values of noisy videos by 4 to 6 dB. Moreover, it is important to note, that the speed of the simplified VBM3D is about 7 times faster than the speed for the standard VBM3D. However, it is still not fast enough for real-time applications since it needs to have at least 25 fps. Therefore, we continue to accelerate the simplified VBM3D by using the second approach.

In the second approach, an integer implementation of the simplified VBM3D is proposed. The algorithm comparison of the proposed implementation and the simplified VBM3D is shown in Table 3.5. The proposed implementation has several improvements compared to the simplified VBM3D, and they are described below.

1. Instead of float type, integer type is used for all variables.

2. Instead of buffer whole video, proposed implementation only buffers 4 frames.

3. Instead of full search, we propose to use a modified approach of diamond

Table 3.5: Algorithm comparison of the proposed implementation and the simplified VBM3D

	Proposed implementation	Simplified VBM3D
Data type	integer	float
Memory	buffer only 4 frames	buffer whole video
Block matching	modified diamond search	full search
Temporal search window	4	5

search [11], and the algorithm is described in detail in Table 3.6.

4. Decrease the number of temporal searching frames from 5 to 4, so all blocks grouped from searched frames are utilized in Haar transform, reducing the computational complexity.

Table 3.6: Description of modified Diamond search

Modified Diamond search algorithm
Step 1. Center a large diamond search pattern (LDSP) at a predefined search window, and search 5 check blocks in a predefined order: center, horizontal and vertical. The first check block, with sum squared difference less than threshold, is the final solution. If the sum squared differences of all check blocks are greater than threshold, and the minimum block distortion point, abbreviated as MBD, is found to be at the center, jump to Step 3; otherwise, go to Step 2.
Step 2. Create a LDSP centered at the position of MBD point from previous search. Search within 5 check blocks with a predefined order: center, horizontal and vertical. The first check block, with sum squared difference less than the threshold, is the final solution. If the sum squared differences of all check blocks are greater than threshold, and the minimum block distortion point, jump to Step 3; otherwise, repeat this step.
Step 3. Switch the search pattern from the large pattern to a small diamond search pattern (SDSP). Then create a SDSP at the position of MBD point from previous search. The minimum block distortion among check blocks is the final solution

Table 3.7 illustrates the performance comparisons of the proposed implementation and the simplified VBM3D. From the results, we find that the proposed implementation is about 4 to 5 time faster than the simplified VBM3D. The proposed filter has above 30 fps, which meets the requirements for real-time video denoising

Table 3.7: Comparison of the performance between the standard VBM3D, the simplified VBM3D and the proposed implementation for denoising video sequences *vassar* and *ballroom* which are corrupted by Gaussian noise with different variances, in computer platform with Intel Core 2 Duo 3GHz and 3.2GB of RAM

σ/PSNR	Resolution: 640 × 480 Number of frames: 250		Standard VBM3D	Simplified VBM3D	Proposed implementation
5/34.13	vassar	Denoised (dB)	40.74	38.36	38.41
		Speed (fps)	1.09	7.22	**34.11**
	ballroom	Denoised (dB)	41.44	37.74	37.73
		Speed (fps)	1.11	7.29	**34.40**
10/28.12	vassar	Denoised (dB)	38.21	33.67	33.92
		Speed (fps)	1.14	7.67	**34.47**
	ballroom	Denoised (dB)	38.69	33.15	32.78
		Speed (fps)	1.09	7.22	**30.49**
15/24.63	vassar	Denoised (dB)	36.57	30.23	30.94
		Speed (fps)	1.10	7.28	**34.25**
	ballroom	Denoised (dB)	36.71	29.97	29.82
		Speed (fps)	1.13	7.12	**34.94**
20/22.18	vassar	Denoised (dB)	35.32	27.84	28.37
		Speed (fps)	1.13	7.28	**33.94**
	ballroom	Denoised (dB)	35.20	27.24	27.80
		Speed (fps)	1.15	7.01	**34.54**

applications. Furthermore, for video *vassar* which has a static background, the proposed implementation outperforms the simplified VBM3D in terms of PSNR values, with PSNR improvement up to 0.7 dB. This is mainly because of the motion search method used in our algorithm. The modified diamond search algorithm gives a strong preference to the position of the reference block, which produces more precise prediction for static background. As a result, our proposed implementation is much faster than the simplified VBM3D, and it outperforms the simplified VBM3D for videos with a static background, just as in video conference applications.

3.5 Conclusion

In this section, we have a general review of video denoising algorithms and VBM3D. VBM3D has excellent filtering ability, but current implementation does not suit for real-time implementations. In order to accelerate VBM3D and preserve good filtering performance, we simplify VBM3D algorithm and implement it in real-time. From our experiments, we conclude that even though the proposed implementation has some PSNR degradation as compared with the standard VBM3D, it still has good denoising performance, with PSNR improvement of around 4 dB over noisy videos. Moreover, it is important to note that the proposed implementation is 30 times faster than the standard VBM3D, and it can be used in real-time video applications.

Chapter 4

Joint Rate-distortion Oriented Video denoising and Compression

4.1 Introduction

In Chapters 2 and 3, we discussed about video compression by using the H.264/AVC standard and video filtering by using VBM3D. We know that pre-filtering is desirable in video coding, since it can enhance both the visual quality and coding efficiency of the compression system [4]. However heuristic methods are typically employed in traditional video pre-filtering and compression systems. In other words, filtering parameters and quantization parameters for pre-filtering and compression are independently selected. But this kind of system does not guarantee the optimal parameters with respect to the rate-distortion framework.

To solve the problem and improve the compression performance of H.264/AVC, we propose two filtering and compression algorithms in this chapter, and they are

- A joint rate-distortion oriented pre-filtering and compression algorithm.
- A joint rate-distortion oriented in-loop filtering and compression algorithm.

We organize this chapter in the following order: Section 4.2 describes the traditional algorithm of separate pre-filtering and compression and its drawbacks. Section 4.3 proposes a joint rate-distortion oriented pre-filtering and compression algorithm. In this system, parameters for filtering by VBM3D and compression by the H.264/AVC encoder are selected together with respect to the rate-distortion framework. Furthermore, Section 4.4 describes the traditional scheme of in-loop filtering in the H.264/AVC standard and its limitations, and then proposes a joint rate-distortion oriented in-loop filtering and compression algorithm.

4.2 Pre-filtering in Typical Video Compression Scheme

Typical video compression systems [40, 41, 42] have two separate parts for preprocessing and compression, and parameters are chosen separately. In other words, heuristic methods are typically employed. The typical scheme of this system is shown in Figure 4.1.

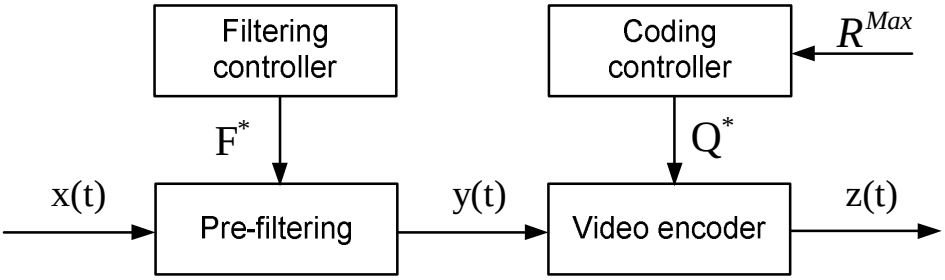


Figure 4.1: Typical pre-filtering and compression scheme

Video sequence $x(t)$ is filtered using a pre-processing filter with parameters F^* and the filtered sequence $y(t)$ is obtained. Then we input $y(t)$ to video encoder with quantization parameters Q^* and the compressed sequence $z(t)$ is received as the output. Since filtering parameters F^* and quantization parameters Q^* are selected separately without consideration of the rate-distortion framework, this system does not guarantee optimal parameters. In order to solve this problem, we need to find a better mode of pre-filtering and compression.

4.3 Joint Rate-distortion Oriented Pre-filtering and Compression

4.3.1 Definition of Optimization Task

In [4], an integrated approach of pre-filtering and compressing image sequences is introduced, where Gaussian filter and the MPEG-2 video compression standard are jointly employed to improve the compression performance by removing blocking artifacts with consideration of the operational rate-distortion framework.

We continue this research of joint parameters selection, and propose a compression system with pre-filtering by VBM3D (from Section 3.3) and compression by the H.264/AVC encoder. This system is shown in Figure 4.2. Video sequence $x(t)$ is filtered by VBM3D with parameter F^* and the filtered sequence $y(t)$ is obtained. Then $y(t)$ is introduced to the H.264/AVC encoder with quantization parameters Q^* and a compressed sequence $z(t)$ is received as the output. Filtering parame-

ters F^* and quantization parameters Q^* are chosen based on bit-budget and the rate-distortion framework by *Joint Rate controller*.

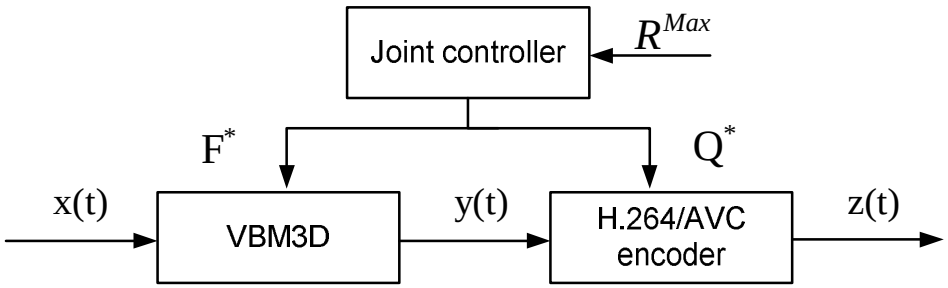


Figure 4.2: Joint pre-filtering and compression scheme

The task of the joint video denoising and compression system is to select the optimal filtering parameters F^* and quantization parameters Q^*, so that

$$\begin{cases} F^*, Q^* = arg \min\limits_{\substack{F \subseteq \{F\} \\ Q \subseteq \{Q\}}} \sum\limits_i D(F_i, Q_i), \\ \sum\limits_i R(F_i, Q_i) \leq R^{Max}, \end{cases} \tag{4.1}$$

where F_i and Q_i are filtering and quantization parameters for frame i, respectively $D(F_i, Q_i)$ and $R(F_i, Q_i)$ are the distortion and rate of frame i with filtering parameter F_i and quantization parameter Q_i, R^{Max} is the bit-budget, which means the maximum bandwidth in data transmission.

We provide a set of filtering parameters F and a set of quantization parameters Q (see Section 4.3.2). A full search is used here to find the optimal solution according to bit restriction and the rate-distortion framework.

In addition, we can treat the three parts - VBM3D, the H.264/AVC encoder and joint controller, together as one component, because from another point of view VBM3D reduces the redundancy contained in the video $x(t)$ to make it more compressible.

4.3.2 Practical Results

In our experiments, the proposed joint pre-filtering and compression algorithm is based on VBM3D (see settings in Table 4.1 and parameters' explanations in Table 4.2) and the H.264/AVC reference software JM V.17.1 [25] in baseline profile (see codec setting in Table 4.3).

Recall Equation 4.1, filtering parameters F^* here include two parts: the first part is fixed, which is shown in Table 4.1; the second part has different sigma values, varying from 0 to 5 with a step of 0.5. We choose maximum sigma as 5, because the source noise usually contained in a video is quite small. Sigma value is less than 5 according to experiments. Practical results were obtained for the test

video sequences *hall* and *foreman* with 352×288 resolution at a frame rate of 30 fps. The performance of the proposed approach is compared with that of the JM V.17.1 encoder in baseline profile (codec setting is the same as in Table 4.3).

Two modes are used in the experiments: *constant quantization mode* and *constant bitrates mode*.

- *Constant quantization mode*: while filtering a video, eleven σ values are used, varying from 0 to 5 with a step of 0.5, because the noise level in original video sequences is typically small. When a video is compressed, we set the quantization parameter $QPI \in \{21, 22 \dots 45\}$ for I frame, and respectively $QPP_i = QPI_i + 5$ for P frames [43]. In our experiments, for each quantization parameters Q^*, we use full search to find the best filtering parameters F^* .

- *Constant bitrates mode*: while filtering a video, we use the same strategy as in the first mode. But when a video is compressed, we enable the constant bitrates control in the H.264/AVC encoder and fix the bitrates with a proper value.

Table 4.1: VBM3D setting for pre-filtering

Parameters	Settings
denoiseFrames	5
transform-2D-HT-name	Identity transform
transform-3rd-dim-name	Haar
N1	8
Nstep	6
Nb	16
N2	4
Ns	5
tau-match	3000
lambda-thr3D	2.7
Wiener filtering	–

Figure 4.3 shows the case of *constant quantization mode for IPPP coding*: quantization parameter $QPI \in \{21, 22 \dots 45\}$ for I frame, and respectively $QPP_i = QPI_i + 5$ for P frames. For video sequence *hall* (352×288), the joint pre-filtering and compression system gains consistent PSNR values with a maximum of 0.5 dB, and uses less bitrates which is as low as 13.4%. Figure 4.4 shows that for video sequence *foreman* (352×288), the joint pre-filtering and compression system gains consistent PSNR values with a maximum of 0.1 dB, and uses less bitrates which is as low as 1.2%.

Table 4.2: Summary of parameters involved in VBM3D setting

Parameters	Description
denoiseFrames	Temporal window length
transform-2D-HT-name	2D transform used for hard-thresholding filtering
transform-3rd-dim-name	transform used in the 3rd dim
N1	N1 x N1 is the block size used for the hard-thresholding (HT) filtering
Nstep	sliding step to process every next reference block
Nb	number of blocks to be used in the predictive-search BM for the next frame
N2	maximum number of similar blocks (maximum size of the 3rd dimension of the 3D groups)
Ns	length of the side of the search neighbourhood for full-search block-matching (BM)
tau-match	threshold for the block distance (d-distance)
lambda-thr3D	threshold parameter for the hard-thresholding in transform domain

Table 4.3: Setting of JM codec

Setting	JM codec V.17.1
Profile	Baseline
Motion estimation	16×16 block in radius 16
search mode	Simplified UMHexagon search
Number of reference frames	1
Skip mode	Enable
deblocking filter	Enable
RD optimization	Low complexity mode
Rate control	Disable
slice size	50 macroblocks

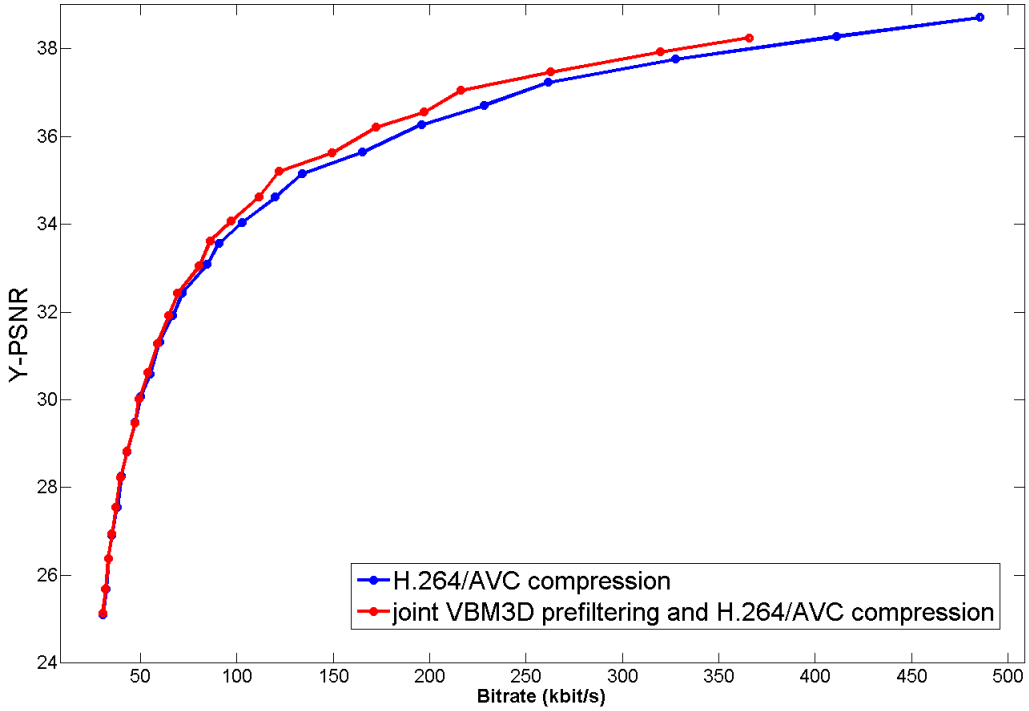


Figure 4.3: Rate-distortion comparison for video *hall* (352×288) in two compression modes: H.264/AVC compression; joint pre-filtering and H.264/AVC compression

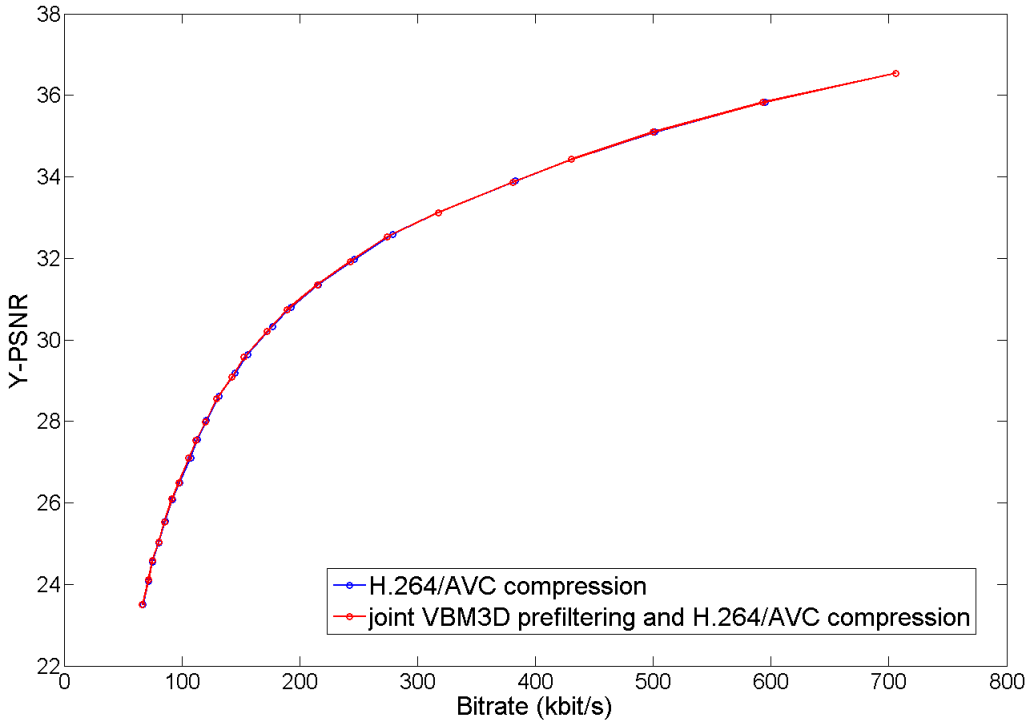


Figure 4.4: Rate-distortion comparison for video *foreman* (352×288) in two compression modes: H.264/AVC compression; joint pre-filtering and H.264/AVC compression

Figure 4.5 shows that in *constant bitrates mode* (bitrates = 215 kbit/s), for video sequence *hall* (352×288), the joint pre-filtering and compression consistently gains PSNR values which is up to 1.2 dB. Figure 4.6 and 4.7 are two frames taken from output video sequences under two compression modes: H.264/AVC compression with enabled constant bitrates control; joint pre-filtering and H.264/AVC compression with enabled constant bitrates control. The results show that the proposed joint pre-filtering and compression system can improve the visual quality of output videos by removing some noise at the door (compare (c) and (e)) and ringing artifacts around the foot (compare (d) and (f)).

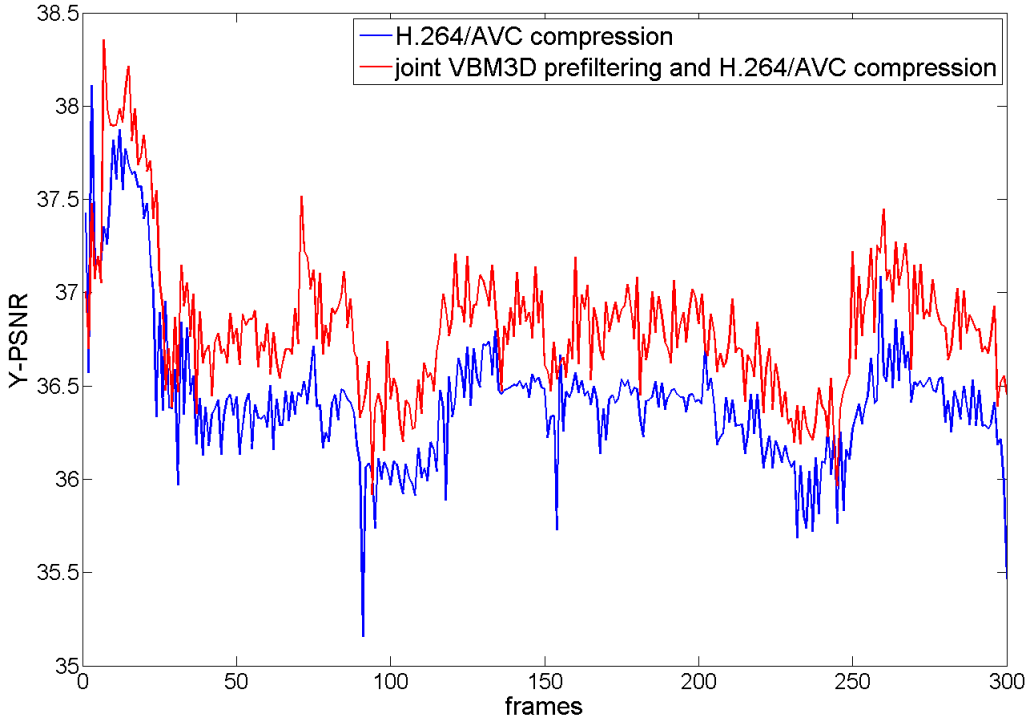


Figure 4.5: Enable constant bitrates control (bitrates = 215 kbit/s), frame by frame PSNR comparison for video *hall* (352×288) in two compression modes: H.264/AVC compression; joint pre-filtering and H.264/AVC compression.

(a) (b)

(c) (d)

(e) (f)

Figure 4.6: For video *hall*, (a) is the 23rd frame of the output video from the H.264/AVC compression system with enabled constant bitrates control, (b) is the 23rd frame of the output video from the joint VBM3D pre-filtering and H.264/AVC compression system with enabled constant bitrates control, (c) and (d) are fragments from (a), (e) and (f) are fragments from (b).

(a) (b)

(c) (d)

Figure 4.7: For video *hall*, (a) is the 91st frame of the output video from the H.264/AVC compression system with enabled constant bitrates control, (b) is the 91st frame of the output video from the joint VBM3D pre-filtering and H.264/AVC compression system with enabled constant bitrates control, (c) and (d) are fragments from (a) and (b) respectively.

4.3.3 Summary

Typical pre-filtering and compression systems do not guarantee optimal filtering and compression parameters. Thus we propose joint parameters selection for pre-filtering and compression system with pre-filtering by VBM3D and compression by the H.264/AVC encoder, and a full search is employed to find optimal filtering and compression parameters. Our results show that the joint pre-filtering and compression produces output video frames with less compression artifacts and increased PSNR up to 1.2 dB under *constant bitrates mode* and up to 0.5 dB under *constant quantization mode*. In addition, the joint pre-filtering and compression system uses less bitrates which is as low as 13.4% in comparison with only compression. In our future work, it is desirable to use a more efficient approach instead of full search to find optimal filtering and quantization parameters for this joint system.

4.4 In-loop Filtering in Typical Video Compression Scheme

In the H.264/AVC encoder, an in-loop deblocking filter is used for removing the blocking artifacts (see Figure 2.2), and the simplified scheme is shown in Figure 4.8. Recall the encoding process: previously coded and reconstructed frames $x^{'}(t+\Delta t)$ are motion estimated (ME), compensated (MC) and subtracted from $x(t)$ to form the residual. Then the integer transform (T) and quantization (Q) is applied to the residual signal to obtain the coefficients ΔX which are later entropy coded in the bitstream. In the reconstruction path, the coefficients ΔX are scaled and inverse transformed. Then the result is added to motion compensated frame and finally it is filtered by an in-loop deblocking filter (DF) to produce the reconstructed frame $x^{'}(t)$.

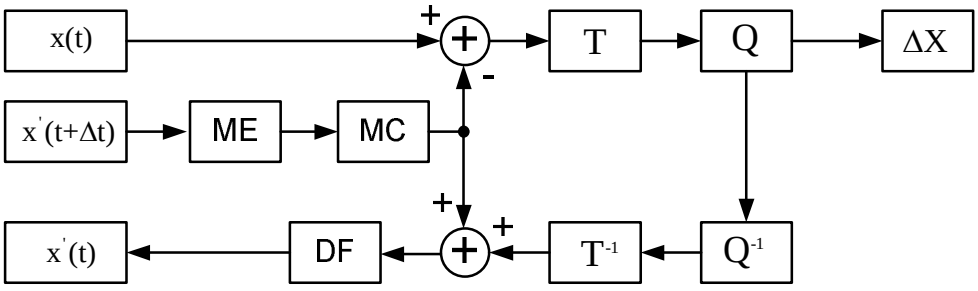


Figure 4.8: Simplified block diagram of the H.264/AVC encoder

As we have discussed in Section 2.7 and Section 2.8, deblocking filter functions well for removing blocking artifacts however it is not efficient for reducing other types of source noise. If we can remove other types of source noise in videos, it

is possible to improve the compression performance. Therefore a better in-loop filtering strategy is needed.

4.5 Joint Rate-distortion Oriented In-loop Filtering and Compression

4.5.1 Definition of Optimization Task

It has been suggested that adding an in-loop spatial-temporal filter in the H.264/AVC standard enhances the compression performance of the H.264/AVC standard [5].

We continue this research and propose a joint rate-distortion oriented in-loop filtering and compression algorithm. We add a VBM3D based real-time filter (proposed in Section 3.4) as an enhancing part after deblocking filter in both the encoder and decoder of H.264/AVC. This filter denoises the current frame with the assistance of previously processed frames and tunes the filtering parameters by using original frame as a target (see Figure 4.9).

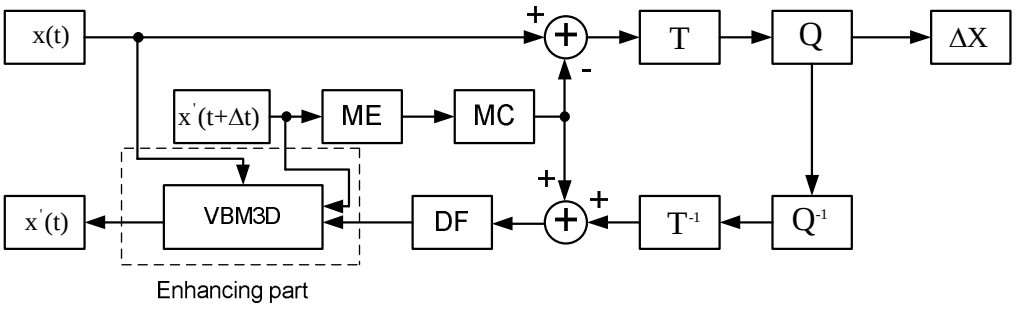


Figure 4.9: Using VBM3D as an enhancing part in H.264/AVC codec

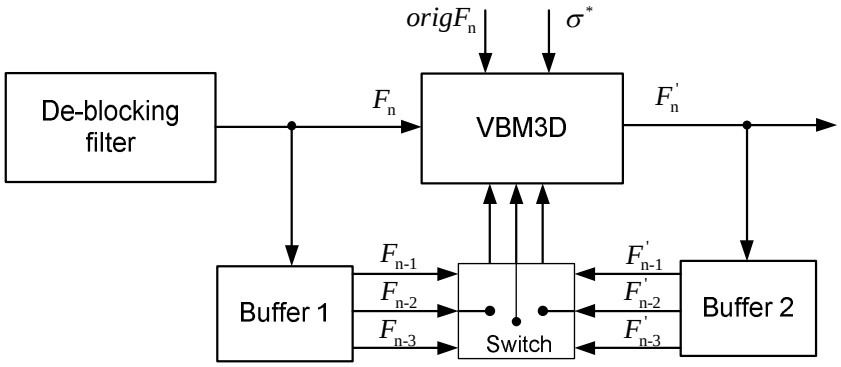


Figure 4.10: Optimization task

Implementation of the enhancing part can be further simplified as in Figure 4.10. The optimization task is to filter current frame with the best filtering parameter σ^*

by which this VBM3D based filter minimizes the difference between denoised frame $F_n^{'}$ and original frame $origF_n$:

$$\begin{cases} F_n^{'} = VBM3D(F_n, \sigma, F_{n-1}, F_{n-2}, F_{n-3}, F_{n-1}^{'}, F_{n-2}^{'}, F_{n-3}^{'}), \\ \sigma^* = arg \min\limits_{\sigma \subseteq \{\sigma\}} \{SSE(origF_n, F_n^{'})\}, \end{cases} \tag{4.2}$$

where F_n is the output of deblocking filter, F_{n-1}, F_{n-2} and F_{n-3} are previous noisy frames, $F_{n-1}^{'}$, $F_{n-2}^{'}$ and $F_{n-3}^{'}$ are previously filtered frames, σ^* is the best filtering parameter among a set of σ used in the filter - which is transmitted to the decoder later so that the filter inside the decoder can directly use σ^* to repeat the same filtering as in the encoder. $F_n^{'}$ is the filtered frame which is obtained from two filtering strategies: filter F_n together with the noisy frames or filter F_n together with the filtered frames, depending on the least sum of squared errors.

4.5.2 Practical Results

In our experiments, the joint in-loop filtering and compression algorithm is based on the proposed real-time filter which is mentioned in Section 3.4 (see filter setting in Table 4.4) and the H.264/AVC reference software JM V.18.0 in mainline profile (codec settings are from Table 4.5 or Table 4.6).

Recall Equation (4.2), for each input frame F_n, there is a set of σ, i.e., $\{\sigma\} = \{0, 0.1, 0.2, ..., 9.9, 10\}$. For each σ, the proposed VBM3D based filter calculates the filtered frame $F_n^{'}$ by two strategies: by using F_n together with three previous noisy frames F_{n-1}, F_{n-2} and F_{n-3}; by using F_n together with three previous denoised frames $F_{n-1}^{'}$, $F_{n-2}^{'}$ and $F_{n-3}^{'}$. The filter then searches the $F_n^{'}$ with the highest PSNR value among those filtered results, gives it as the output and transmits the best sigma value σ^* to video decoder.

While compression, both *inter* and *intra* modes are tested.

- In *inter* mode: we set the quantization parameter $QPI \in \{21, 22 \dots 45\}$ for I frame, and respective $QPP_i = QPI_i + 5$ for P frames [43]. Codec settings for JM V.18.0 are shown in Table 4.5.

- In *intra* mode: we set the quantization parameter $QPI \in \{21, 22 \dots 45\}$ for I frame. Codec settings for JM V.18.0 are shown in Table 4.6.

Practical results were obtained for the test video sequences *hall* and *foreman* with the resolution of 352×288 at a frame rate of 30 fps. The performance of the proposed approach is compared to that of the H.264/AVC reference software JM V.18.0 encoder in mainline profile.

Table 4.4: Setting of proposed filter

Parameters	Settings
denoiseFrames	4
transform-3rd-dim-name	Haar
N1	8
Nstep	6
Nb	1
N2	4
Ns	11
tau-match	3000
lambda-thr3D	2.7

Table 4.5: JM Codec setting under inter mode

Setting	JM codec V.18.0
Profile	Mainline
Inter mode	Enable
Motion estimation	8×8 block in radius 8
search mode	Enhanced Predictive Zonal Search (EPZS)
Number of reference frames	1
Skip mode	Enable
deblocking filter	Enable
RD optimization	Low complexity mode
Rate control	Disable
Slice Mode	Off

Table 4.6: JM Codec setting under intra mode

Setting	JM codec V.18.0
Profile	Mainline
Intra mode	Enable
deblocking filter	Enable
RD optimization	Low complexity mode
Rate control	Disable
Slice Mode	Off

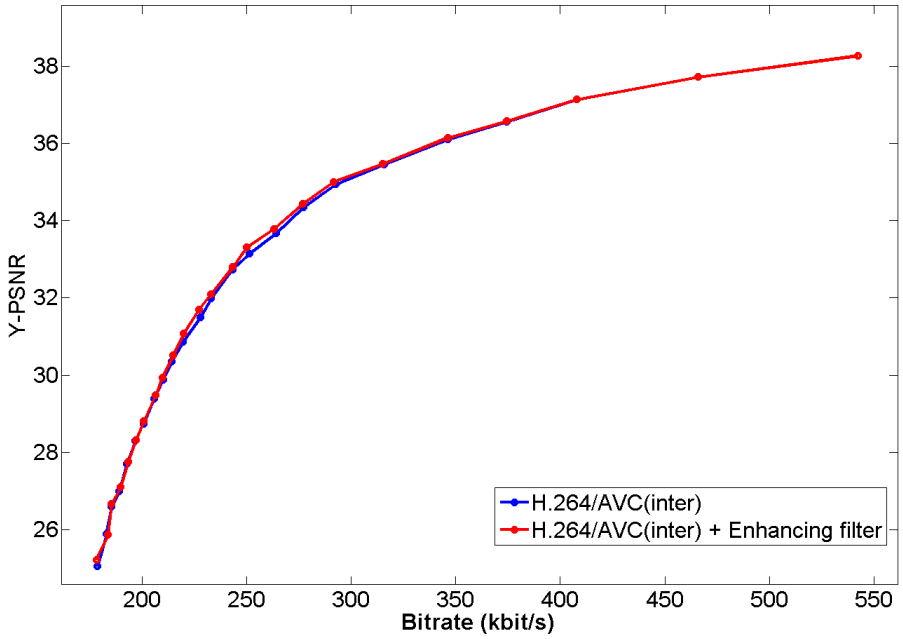


Figure 4.11: For video *hall*, comparison of rate-distortion performance in two compression modes: H.264/AVC under inter mode; H.264/AVC with enhanced in-loop filtering under inter mode

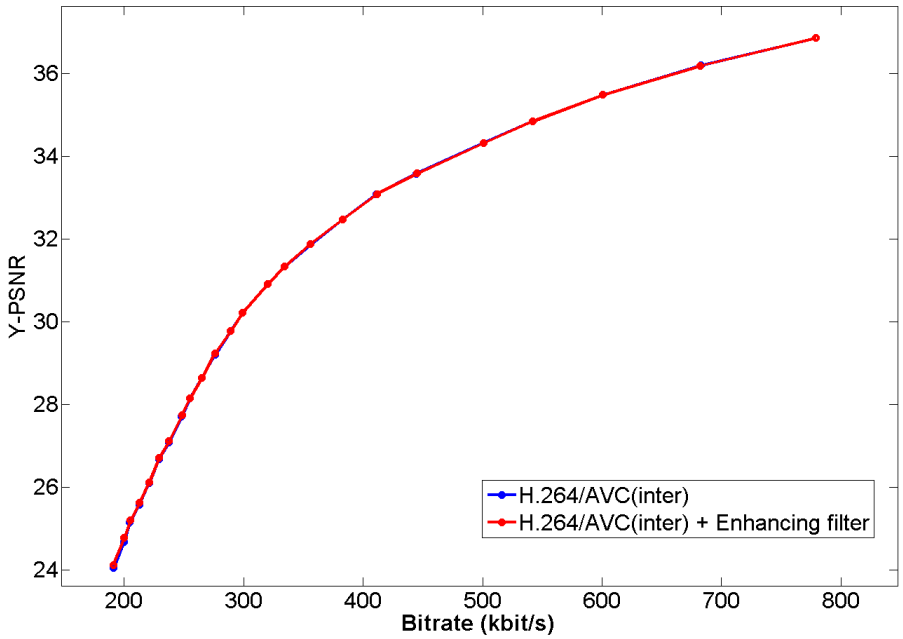


Figure 4.12: For video *foreman*, comparison of rate-distortion performance in two compression modes: H.264/AVC under inter mode; H.264/AVC with enhanced in-loop filtering under inter mode

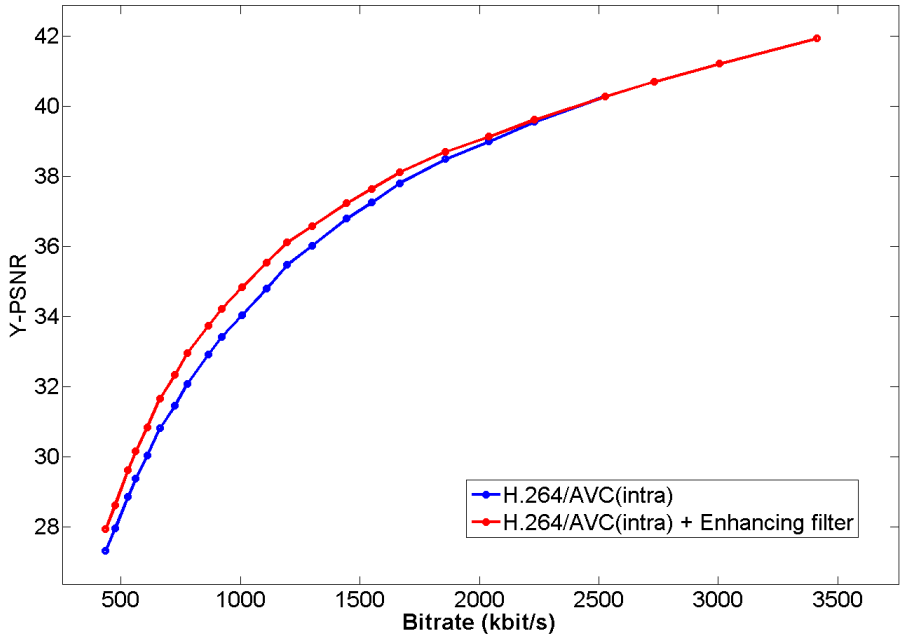


Figure 4.13: For video *hall*, comparison of rate-distortion performance in two compression modes: H.264/AVC under intra mode; H.264/AVC with enhanced in-loop filtering under intra mode

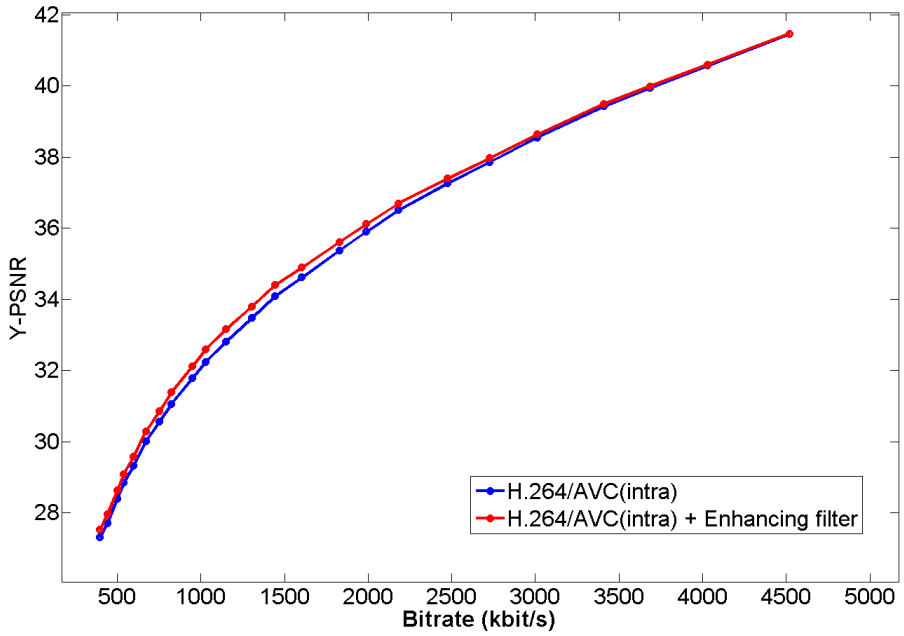


Figure 4.14: For video *foreman*, comparison of rate-distortion performance in two compression modes: H.264/AVC under intra mode; H.264/AVC with enhanced in-loop filtering under intra mode

Figure 4.11 shows that under *inter* mode, for video sequence *hall* (352×288), the joint in-loop filtering and compression system gains consistent PSNR values with a maximum value of 0.22 dB, and uses less bitrates which is as low as 2.1%. Figure 4.12 shows that under *inter* mode, for video sequence *foreman* (352×288), the joint in-loop filtering and compression system has consistent PSNR gains which is up to 0.1 dB.

Figure 4.13 shows that under *intra* mode, for video sequence *hall* (352×288), the output videos of the joint in-loop filtering and compression system have consistent higher PSNR values with a maximum value of 0.87 dB than only compression. Moreover, this joint approach can save the bitrates up to 10.5% in comparison with only compression. Figure 4.14 shows that under *intra* mode, for video sequence *foreman* (352×288), the joint in-loop filtering and compression system has consistent PSNR gains with a maximum value of 0.35 dB and bitrates savings up to 6.3%.

4.5.3 Summary

Typical in-loop filtering and compression system only has one deblocking filter which focuses on removing blocking artifacts. However, it is desirable to have an additional filter to remove other types of noise contained in the video. Thus, we present a joint in-loop filtering and compression system, in which we add a real-time filter into the H.264/AVC encoder and decoder to improve the compression performance. Results show that this joint approach consistently improves the compression performance of H.264/AVC under *intra* mode, but it gains little under *inter* mode. Under *intra* mode, compared to only compression, the proposed joint approach gains consistent PSNR values with a maximum value of 0.87 dB and uses less bitrates which is as low as 10.5%.

Chapter 5

Conclusion

In this study, we have presented a real-time video denoising filter, a joint pre-filtering and compression algorithm, and a joint in-loop filtering and compression algorithm.

VBM3D achieves state-of-the-art video denoising performance in terms of both peak signal-to-noise ratio and subjective visual quality. The proposed filter is based on VBM3D. Even though the simplification of the VBM3D algorithm leads to some PSNR degradation, the proposed filter has good denoising performance. Moreover, this filter is over 30 times faster than the original implementation of VBM3D, and makes it possible to be employed in real-time applications.

In traditional video pre-filtering and compression systems, pre-filtering and compression are two separate processes, and they do not guarantee optimal filtering and quantization parameters with respect to the rate-distortion framework. To solve this problem, we present a joint approach with pre-filtering by VBM3D and compression by H.264/AVC. Practical results show that this joint approach produces output video frames with less compression artifact and consistently increased PSNR values with a maximum value of 1.2 dB under *constant bitrates mode* and 0.5 dB under *constant quantization mode*. Besides, this joint approach uses less bitrates which is as low as 13.4% in comparison with only compression. Because the joint approach enhances the compression performance of the H.264/AVC standard without changing anything in the standard, the video coding standard can be replaced by any other kinds of video coding standards. Flexibility is the main advantage of this joint video pre-filtering and compression algorithm. In our future work, we plan to use a more efficient method instead of full search to find optimal filtering and quantization parameters.

A deblocking filter is used to reduce blocking artifacts in traditional video compression systems. However, other types of noise are introduced while compression decrease the compression performance of video coding standard. Therefore, we propose a joint in-loop filtering and compression algorithm, in which we add the proposed real-time filter as an enhancing part after deblocking filter in the H.264/AVC codec. Experiments illustrate that this joint approach consistently improves the

compression performance of H.264/AVC under both *intra* and *inter* mode. Under *intra* mode, compared to only compression, the proposed joint approach gains consistent PSNR values with a maximum value of 0.87 dB and uses less bitrates which is as low as 10.5%. However, the gain under *inter* mode is little, and this may be caused by the simplicity of the added filter. In other words, the filtering strength of current filter is not enough. Therefore, we can enhance the filtering strength by implementing a real-time filter with the full features of VBM3D and apply it in this joint approach.

In real-life applications, this joint rate-distortion oriented video denoising and compression algorithm can be used among the H.264/AVC standard based video applications, such as video conferencing or high-definition DVD. Furthermore, the idea of this joint video denoising and compression scheme is possible to be used in next video compression standard, High Efficiency Video Coding (HEVC) or H.265. Besides standard codecs, this joint video denoising and compression algorithm can be employed by non-standard video codec to improve their compression performance.

References

[1] K. Dabov, A. Foi and K. Egiazarian. Video denoising by sparse 3D transform-domain collaborative filtering. *European Signal Processing Conference (EUSIPCO), Poznan, Poland*, September 2007.

[2] Hantao Liu, N. Klomp and I. Heynderickx. A Perceptually Relevant Approach to Ringing Region Detection. *IEEE Transactions on Image Processing*, 19(6):1414-1426, June 2010

[3] T. Wiegand, G.J. Sullivan, G. Bjontegaard and A. Luthra. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology,* 13(7):560-576, July 2003

[4] P.V. Karunaratne, C.A. Segall and A.K. Katsaggelos. A rate-distortion optimal video pre-processing algorithm. *Proceedings of International Conference on Image Processing,* 1:481-484, 2001

[5] D.T Vo and T.Q. Nguyen. Optimal motion compensated spatio-temporal filter for quality enhancement of H.264/AVC compressed video sequences. *IEEE International Conference on Image Processing (ICIP)*, page 3173-3176, 2009.

[6] A. Hallapuro, M. Karczewicz and H. Malvar. Low Complexity Transform and Quantization Part I: Basic Implementation. *JVT document JVT-B038, Geneva*, February 2002.

[7] ISO/IEC 14496-10 and ITU-T Rec. H.264, Advanced Video Coding, 2003.

[8] I. E. G Richardson. Front Matter, in H.264 and MPEG-4 Video Compression: Video Coding for Next-Generation Multimedia. *John Wiley and Sons,* 2004.

[9] N. Ahmed, T. Natarajan and K.R. Rao. Discrete Cosine Transform. *IEEE Transactions on Computers,* C23(1):90-93, 1974.

[10] K. R. Rao and P. Yip. Discrete Cosine Transform: Algorithms, Advantages, Applications. *Academic Press, Boston,* 1990.

[11] Shan Zhu and Kai-Kuang Ma. A new diamond search algorithm for fast block matching motion estimation. *Proceedings of International Conference on Communications and Signal Processing,* 1:292-296, 1997.

[12] Ce Zhu, Xiao Lin and Lap-Pui Chau. Hexagon-based search pattern for fast block motion estimation. *IEEE Transactions on Circuits and Systems for Video Technology,* 12(5):349-355, May 2002.

[13] P. List, A. Joch, J. Lainema, G. Bjontegaard and M. Karczewicz. Adaptive deblocking filter. *IEEE Transactions on Circuits and Systems for Video Technology,* 13(7):614-619, July 2003.

[14] D. Gluss and E.W Weisstein. Lagrange Multiplier. From MathWorld.

[15] M. Yuen and H. R. Wu. A survey of hybrid MC/DPCM/DCT video coding distortions. *Signal Processing*, 70(3):247-278, 1998.

[16] K. Dabov, A. Foi, V. Katkovnik and K. Egiazarian. Image denoising by sparse 3D transform-domain collaborative filtering. *IEEE Transactions on Image Processing*,16(8):2080-2095, August 2007.

[17] K. Dabov, A. Foi, V. Katkovnik and K. Egiazarian. Image denoising with block-matching and 3D filtering. *Proceedings of SPIE Electronic Imaging, San Jose, California, USA,* January 2006.

[18] D. Rusanovskyy, K. Dabov and K. Egiazarian. Moving-Window Varying Size 3D Transform-Based Video Denoising. *Proceedings of International Workshop on Video Processing and Quality Metrics (VPQM), USA,* 2006.

[19] M. Maggioni, G. Boracchi, A. Foi and K. Egiazarian. Video Denoising, Deblocking and Enhancement through Separable 4-D Nonlocal Spatiotemporal Transforms. *IEEE Transactions on Image Processing, preprint,* January 2011.

[20] D. Marpe, T. Wiegand and G.J. Sullivan. The H.264/MPEG4 advanced video coding standard and its applications. *IEEE Transaction on Communications,* 44(8):134-143, August 2006.

[21] X. Shen and Y. Wu. Exploiting sparsity in dense optical flow. *IEEE International Conference on Image Processing (ICIP),* page 741-744, 2010.

[22] Advanced video coding for generic audiovisual services. ITU-T Recommendation H.264, March 2005.

[23] Iain E. G. Richardson. H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia. *John Wiley and Sons Publisher,* 2003.

[24] P.V. Karunaratne, C.A. Segall and A.K. Katsaggelos. A rate-distortion optimal video pre-processing algorithm. *Proceedings of International Conference on Image Processing*, 1:481-484, , 2001.

[25] H.264/AVC JM Reference Software, http://iphome.hhi.de/suehring/tml/

[26] M.K. Ozkan, M.I. Sezan and A.M. Tekalp. Adaptive motion-compensated filtering of noisy image sequences. *IEEE Transactions on Circuits and Systems for Video Technology,* 3(4):277-290, August 1993.

[27] F. Jin, P. Fieguth, L. Winger and E. Jernigan. Adaptive Wiener filtering of noisy images and image sequences. *Proceedings of International Conference on Image Processing,* September 2003.

[28] J. Woods and C. Radewan. Kalman filtering in two dimensions. *IEEE Transactions on Information Theory*, 23(4):473- 482, July 1977.

[29] A. Buades, B. Coll and J.M. Morel. A non-local algorithm for image denoising. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition,* 2:60-65, June 2005.

[30] D.L. Donoho. De-noising by soft-thresholding. IEEE Transactions on Information Theory, 41(3):613-627, May 1995.

[31] G. de Haan. IC for motion-compensated de-interlacing, noise reduction, and picture-rate conversion. IEEE Transactions on Consumer Electronics, 45(3):617-624, August 1999.

[32] R. Rajagopalan and M.T. Orchard. Synthesizing processed video by filtering temporal relationships. IEEE Transactions on Image Processing, 11(1):26-36, January 2002.

[33] A.J. Patti, A.M. Tekalp and M.I. Sezan. A new motion-compensated reduced-order model Kalman filter for space-varying restoration of progressive and interlaced video. *IEEE Transactions on Image Processing,* 7(4):543-554, April 1998.

[34] E.J. Balster, Y.F. Zheng and R.L. Ewing. Combined spatial and temporal domain wavelet shrinkage algorithm for video denoising. *IEEE Transactions on Circuits and Systems for Video Technology,* 16(2):220-230, February 2006.

[35] A. Buades, B. Coll and J.M. Morel. Denoising image sequences does not require motion estimation. *IEEE Conference on Advanced Video and Signal Based Surveillance,* page 70-74, September 2005.

[36] S. Zhu and Kai-Kuang Ma. A new diamond search algorithm for fast block-matching motion estimation. *IEEE Transactions on Image Processing,* 9(2):287-290, February 2000.

[37] I. Johnstone, D. Donoho and I.M. Johnstone. Ideal spatial adaptation by wavelet shrinkage. *Biometrika,* 81:425-455, 1993.

[38] B. K. P. Horn and B. G. Rhunck. Determining optical flow. *Artificial Intell.,* 17:185-203, April 1981.

[39] C. Liu and W.T. Freeman. A High-Quality Video Denoising Algorithm Based on Reliable Motion Estimation. *Proceedings of ECCV,* page 706-719, 2010.

[40] N. Young and A.N. Evans. Digital video pre-processing with multi-dimensional attribute morphology. *International Conference on Visual Information Engineering,* page 89-92, July 2003.

[41] J. Goel, D. Chan and P. Mandl. Pre-processing for MPEG compression using adaptive spatial filtering. *IEEE Transactions on Consumer Electronics,* 41(3):687-689, August 1995.

[42] N. Young and A.N. Evans. Psychovisually tuned attribute operators for pre-processing digital video. *IEEE Proceedings of Image and Signal Processing,* 150(5):277-86, October, 2003.

[43] H. Schwarz, D. Marpe and T. Wiegand. Overview of the Scalable Video Coding Extension of the H.264/AVC Standard. *IEEE Transactions on Circuits and Systems for Video Technology,* 17(9):1103-1120, September 2007.

[44] J.L. Liang and A. Ortega. Perceptually based video rate control using pre-filtering and predicted rate-distortion characteristics. , 1997. Proceedings of International Conference on Image Processing, 2:57-60, October 1997.

[45] MPEG-2, Test Model 5. Test Model Editing Committee, April 1993.