Improvement and Optimization of H.264 Video Codec

TANG, Kai Lam

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree of

Master of Philosophy

in

Electronic Engineering

Acknowledgement

I would like to express my deep and sincere gratitude to my supervisor, Professor Ngan King Ngi, for his invaluable suggestions and guidance. His expertise in visual signal processing guides me towards the completion of my project and research.

I wish to express my warm and sincere thanks to the technical staff and my fellow students in the Visual Signal Processing and Communications Laboratory. Their detailed and constructive comments, help and suggestions support me during my difficult moments.

I would like to warmly thank the department of Electronic Engineering of the Chinese University of Hong Kong for providing me with a comfortable study environment and advanced facilities.

Abstract

H.264/AVC is the latest video coding standard which outperforms previous video coding standards. However, it is computational intensive. Accordingly, optimization is required for reducing the complexity for real-time applications. In addition, even though the coding performance of H.264 is excellent, it can still be further improved.

Coding of intra slices requires a large number of bits and therefore, to improve the coding performance, intra block matching technique is applied with several proposed enhancement techniques including best match prediction, multiple best matches, novel padding method, skip mode, etc. Experimental results show that the coding performance can be improved significantly.

Variable block size motion estimation is one of the techniques contributes to the complexity of H.264. Accordingly, a fast variable block size motion estimation algorithm is proposed. It reuses SAD within the same macroblock and uses pattern-based motion estimation and refinement search to reduce the complexity while maintaining the coding performance.

The complexity of H.264 is so high that encoding and decoding video in real-time on Pocket PC is not easily achieved. Accordingly, various optimization techniques are proposed and applied to develop a Pocket PC software-based real-time H.264 codec. The developed codec can encode and decode video in more than 25 frames per second. Besides, since the system is developed in software, it is suitable for technology transfer.

摘要

H.264 是現在最先進的視頻編碼標準之一，其壓縮率及視訊品質都比過去的視頻編碼標準較爲優勝，可是它的算法複雜度極高，爲了能夠做到實時編解碼，優化是必須的。雖然 H.264 有這麼好的壓縮率及視訊品質，改進的空間仍然是存在的。

幀內條帶需要用很多的比特來進行編碼，爲了壓縮得更有效率，我們提出了多種的強化技術來提高一種叫幀內塊匹配技術的性能，提出的技術包括最佳匹配預測、數個最佳匹配、填充方法、跳過模式等。實驗結果顯示當用了這些技術以後，系統的性能得到大大的改善。

可變塊大小運動估計是其中一種令 H.264 的複雜度高的技術，因此，我們提出了一種快速可變塊運動估計算法，這算法主要是通過重複運用已計算的 SAD 與模式匹配運動估計來減低複雜度和保持編碼性能。

由於 H.264 的複雜度極高，所以要在掌上電腦上進行實時編解碼並不容易，爲了開發一個基於軟件的、實時的 H.264 掌上電腦編解碼器，我們提出了不少優化技術，開發出來的編解碼器能在每秒編解碼多於廿五張幀，此外，因爲整個系統的開發都是基於軟件的，所以很容易就可以進行科技轉換。

Contents

Publication List

1. Enhanced SAD Reuse Fast Motion Estimation, Kai Lam Tang and King N. Ngan, Proceedings of SPIE, Vol. 6508, Visual Communications and Image Processing 2007

2. Enhancement Techniques for Intra Block Matching, Kai Lam Tang and King N. Ngan, ICME'07 - Multimedia Coding and Processing Track

Chapter 1 Introduction

H.264/AVC [7, 18, 19, 30] is the latest video coding standard which outperforms previous video coding standards such as H.261, H.263, MPEG-1, MPEG-2 and MPEG-4. However, it is more computational intensive. Consequently, optimization is required for reducing the complexity for real time applications. In addition, even though the performance of H.264 is excellent, there is still room for further improving the standard to achieve better coding performance.

As mentioned, optimization and improvement of H.264 are necessary. Therefore, this thesis focuses on discussing the optimization techniques and improving the algorithms for H.264.

The following sections describe the fundamental principles of video coding and provide an introduction of the H.264 standard. At the end of this chapter, the organization of this thesis and the contributions will be discussed.

1.1 Video Coding

Video coding is the process of compressing and decompressing a digital video signal [18]. Video compression aims to reduce the data used to represent the video content without significant loss in quality. After the video is compressed, it can be stored with less storage resources or can be transmitted efficiently. A lot of data are required to represent an uncompressed video. Consequently, it is valuable to develop efficient compression algorithms. Normally, the spatial and temporal correlations of video are exploited to achieve data compression.

Video compression can be classified into two types. One is lossless compression and the other is lossy compression. With lossless compression, compressed data can be decompressed back to the original data, i.e. the decompressed data are the same as the original data. As implied from its name, lossy compression causes loss of data during compression. Therefore, the decompressed data are not the same as the original data. For video data, if the loss is not excessive, the visual quality of the decompressed video will not be degraded significantly or unable to be perceived by human eyes. .

When a video is being processed, normally each picture is partitioned into many macroblocks which comprise 16x16 pixels. For 4:2:0 Format, a macroblock comprises 16x16 luma components and two 8x8 chroma components as illustrated in Figure 1.1. The reason for partitioning each picture into 16x16 macroblocks is that when the partition size is too small, more overhead information is necessary to be transmitted. When the partition size is too large, prediction techniques cannot achieve accurate results and thus more resources are required to encode the prediction errors. Consequently, the size of a macroblock is 16x16 in traditional standards. (Recently, some researchers are trying to use 32x32 super-macroblock for High-Definition (HD) video processing.)

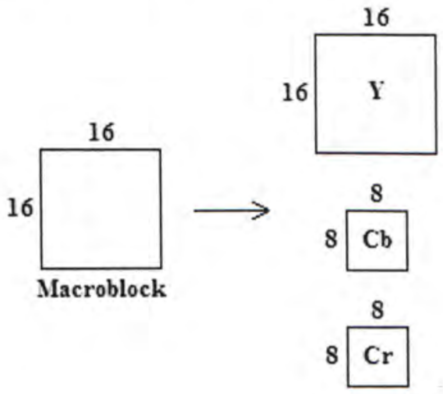
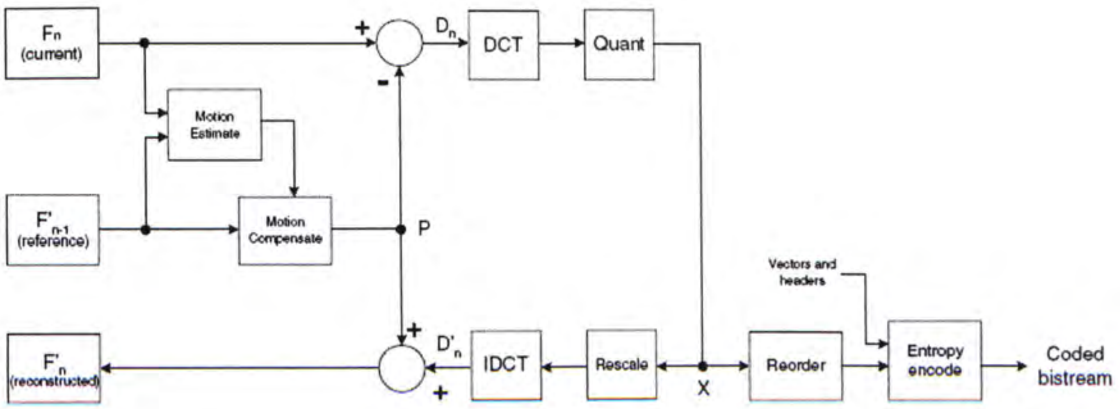
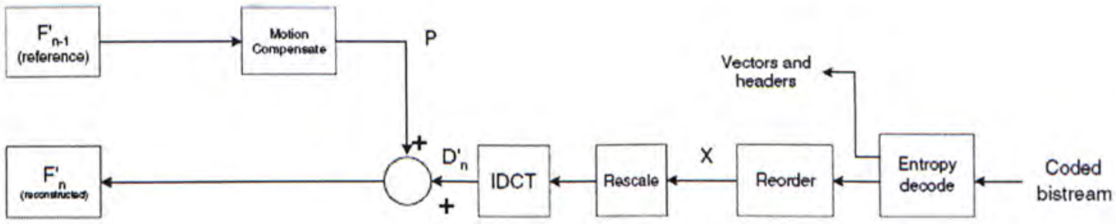


Figure 1.1 A macroblock comprises 16x16 luma (Y) components and two chroma (Cb and Cr) components for 4:2:0 Format.

After the picture is partitioned into macroblocks, each of them is processed with various techniques. Traditionally, these techniques mainly include intra/inter prediction, transform and inverse transform, quantization and de-quantization, entropy coding and decoding. With these techniques, videos can be compressed efficiently. Figure 1.2 illustrates a generic DPCM/DCT hybrid encoder and decoder [18].



(a)



(b)

Figure 1.2 (a) A DPCM/DCT video encoder and (b) A DPCM/DCT hybrid decoder [18].

In Figure 1.2 (a), each macroblock of picture F_n attempts to find a best match from the reference picture. After the best match is found, the motion vectors are encoded and the best match is selected as prediction P for the current macroblock. The prediction is subtracted from the current MB to generate the prediction errors D_n. The prediction errors are transformed and quantized. The quantized transform coefficients X are reordered with the specified scanning method and finally they are entropy encoded.

Only the encoded prediction errors and header information are transmitted to the decoder. Based on previously received data, the decoder generates the same prediction P as that generated in the encoder. In addition, the decoder performs entropy decoding and reorders the decoded quantized transform coefficients. Then they are rescaled, inverse transformed and adding to the prediction to construct a picture.

To ensure both encoder and decoder reconstruct the same picture for prediction, the prediction errors are also rescaled, inverse transformed and adding to the prediction in the encoder side. Note that the structure of the decoder and the reconstruction structure in the encoder are the same. Consequently, both encoder and decoder have the same reconstructed picture for prediction. These are the fundamental principles of video coding.

In the following sections, the techniques mentioned above including temporal prediction, transformation, quantization and entropy coding will be discussed in detail.

1.1.1 Temporal Prediction

As mentioned in Section 1.1, a best match is found from a reference picture for each macroblock. The reason for doing this is that normally there is high temporal correlation between neighboring pictures in a video, i.e. the neighboring pictures in a video are very similar. Figure 1.3 illustrates the temporal correlation of two video sequences, *mobile* and *news*. Figures 1.3 (a), (b), (d) and (e) show the frame 0 and frame 1 of *mobile* and *news*, respectively. Figures 1.3 (c) and (f) demonstrate the differences between frame 0 and frame 1 without motion compensation.

(a) *Mobile*, Frame 0

(b) *Mobile*, Frame 1

(c) *Mobile*, Frame 1 - Frame 0

(d) *News*, Frame 0

(e) *News*, Frame 1

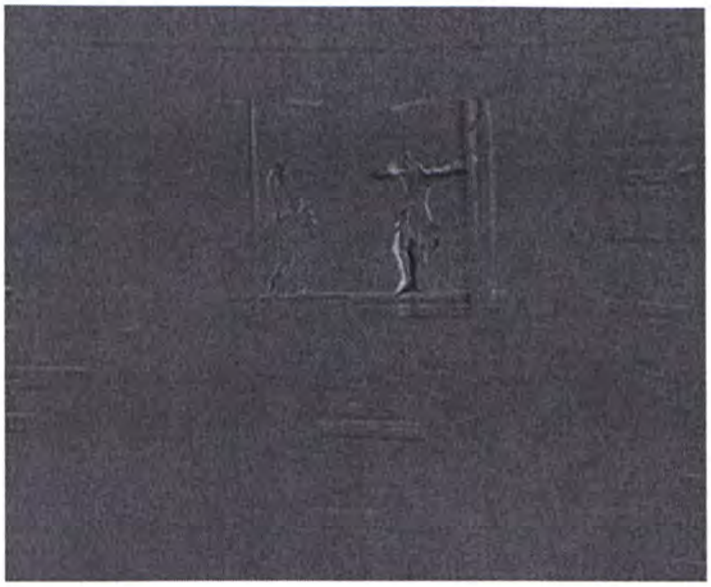

(f) *News*, Frame 1 - Frame 0

Figure 1.3 The temporal correlation of *mobile* and *new* video sequences.

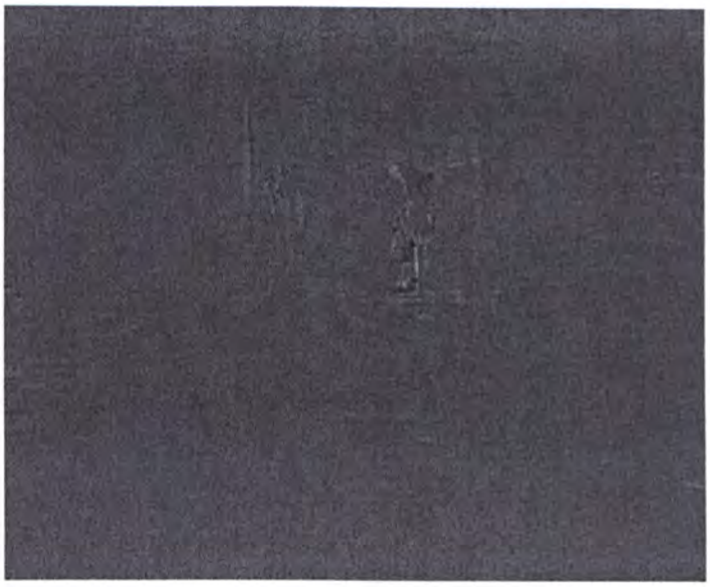
In Figure 1.3 (c) and (f), pixels with grey value represent zero prediction error, white and black represent 255 and -255, respectively. It is obvious that even though without applying motion compensation, for sequence with small motion, the prediction errors are small. However, when the motion of the sequence is large, then the prediction errors become larger and therefore more bits are necessary to encode the prediction errors. Consequently, a technique called motion estimation and compensation is developed to minimize the prediction errors.

Motion estimation is to estimate which position in the reference picture is the best match of the current block. To determine the best match, a cost function is required. Normally, the sum of absolute difference (SAD) or sum of squared difference (SSD) is employed. Due to the low complexity of SAD and its acceptable accuracy, it is generally preferred. The position with the minimum cost is selected as the best match and this best match is chosen as the prediction. With this motion compensated prediction, the prediction errors are reduced significantly as illustrated in Figure 1.4.

Figures 1.4 (a) and (b) show the prediction error of *mobile* and *news* with motion compensation. The motion estimation and compensation employed for Figure 1.4 was 8x8 block based with search range equaled to 32. From the figure, it can be observed that prediction errors are reduced significantly when compared with those in Figure 1.3 (c) and (f) and nearly equal to zero. This implies that fewer bits are required for encoding them. Consequently, higher compression can be achieved.

An accurate temporal prediction can contribute to high compression. If other techniques are employed to process the prediction errors, an even higher compression can be achieved. Those techniques will be discussed in the next sections.

(a)



(b)

Figure 1.4 The prediction errors with motion compensation of (a) *mobile* and (b) *news*.

1.1.2 Transform Coding

The correlation of image data or prediction errors is generally high. A decorrelation process is to reduce the correlation of the data. Transformation is such a process. After transformation, most of the energy is packed into few transform coefficients. These coefficients are quantized and can be encoded efficiently.

Karhunen-Loeve Transform (KLT) is optimal in the mean square error sense. However, it depends on the statistics of the data and therefore it cannot be implemented with fast algorithm. Due to its complexity, video coding standards do not adopt this transform for their transform coding process.

Discrete Cosine Transform (DCT) is a sub-optimal transform. Even though it cannot decorrelate data as efficiently as KLT, it does not depend on the statistics of the data and it has fast algorithm. Therefore, in previous video coding standards such as h.261, MPEG-1, DCT is adopted for their transform coding scheme.

The forward 2D DCT of an NxN data is given by Equation 1.1 [18],

$$Y = T\,X\,T^t \tag{1.1}$$

and the inverse 2D DCT is given by Equation 1.2.

$$X = T^t\,Y\,T \tag{1.2}$$

where X is a matrix of the input signals,

Y is a matrix of the DCT coefficients,

T is an NxN transform matrix, and

the elements of T are given by Equation 1.3.

$$T_{ij} = C_i \cos\frac{(2j+1)i\pi}{2N} \qquad \text{where } C_0 = \sqrt{\frac{1}{N}}, \text{ and } C_i = \sqrt{\frac{2}{N}} \quad (i > 0) \tag{1.3}$$

For 8x8 DCT, the transform matrix T is:

$$T = \begin{matrix}
0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 \\
0.4904 & 0.4157 & 0.2778 & 0.0975 & -0.0975 & -0.2778 & -0.4157 & -0.4904 \\
0.4619 & 0.1913 & -0.1913 & -0.4619 & -0.4619 & -0.1913 & 0.1913 & 0.4619 \\
0.4157 & -0.0975 & -0.4904 & -0.2778 & 0.2778 & 0.4904 & 0.0975 & -0.4157 \\
0.3536 & -0.3536 & -0.3536 & 0.3536 & 0.3536 & -0.3536 & -0.3536 & 0.3536 \\
0.2778 & -0.4904 & 0.0975 & 0.4157 & -0.4157 & -0.0975 & 0.4904 & -0.2778 \\
0.1913 & -0.4619 & 0.4619 & -0.1913 & -0.1913 & 0.4619 & -0.4619 & 0.1913 \\
0.0975 & -0.2778 & 0.4157 & -0.4904 & 0.4904 & -0.4157 & 0.2778 & -0.0975
\end{matrix}$$

The forward and inverse 2D DCT can be written in another form as shown in Equations 1.4 and 1.5 [18].

$$Y_{xy} = C_x C_y \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} X_{ij} \cos\frac{(2j+1)y\pi}{2N} \cos\frac{(2i+1)x\pi}{2N} \tag{1.4}$$

$$X_{ij} = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} C_x C_y Y_{xy} \cos\frac{(2j+1)y\pi}{2N} \cos\frac{(2i+1)x\pi}{2N} \tag{1.5}$$

The basis pictures of 8x8 DCT are shown in Figure 1.5.

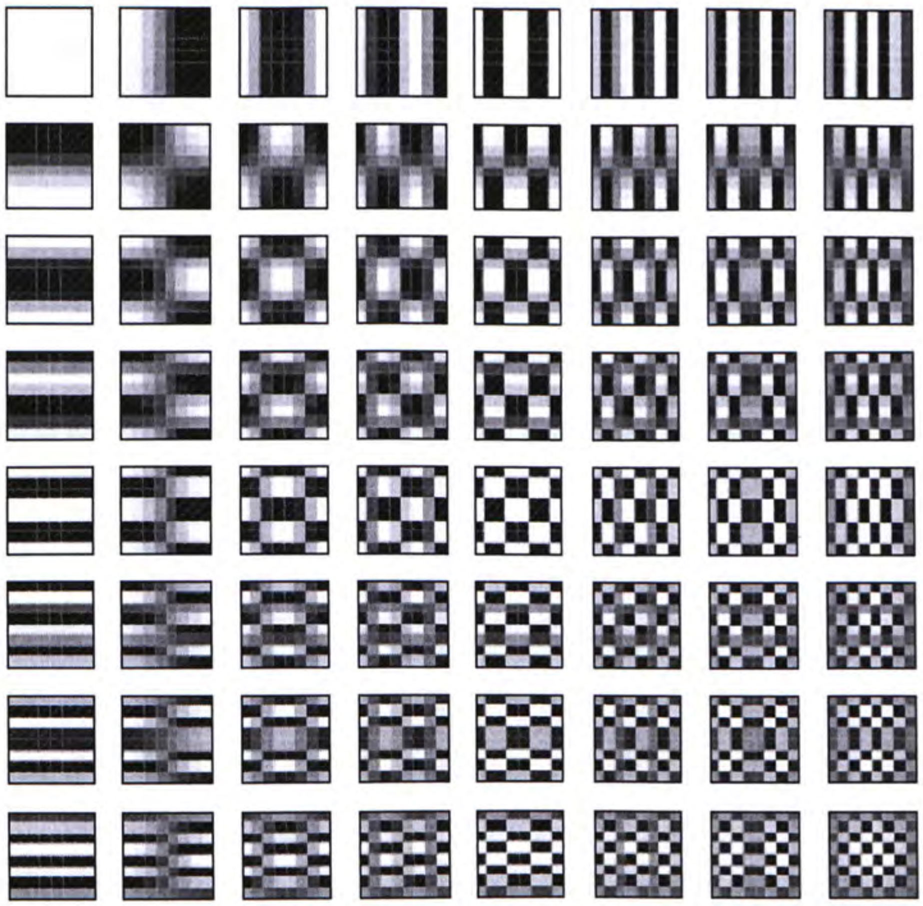


Figure 1.5 The 8x8 DCT basis pictures [18].

Figure 1.6 illustrates how DCT packs the energy of an 8x8 block into a small number of coefficients. X is the input signal. C is the DCT coefficients of the input signal. It can be observed that most of the energy is packed to the top-left corner in C. Therefore, after quantization and entropy encoding, the data can be encoded with a small number of bits and thus the compression can be increased.

```
       55   54   53   55   55   54   55   52
       54   55   52   52   53   54   54   49
       55   55   53   53   55   54   54   55
       55   55   54   55   55   54   54   52
 X =
       57   57   55   54   54   53   54   54
       55   52   53   54   55   55   55   49
       52   53   53   53   54   55   55   48
       55   55   55   55   53   53   54   54
```

(a)

```
      431    4   -2    5   -2    2   -3    1
        0    0    1    1    1   -1   -1    0
       -2   -1   -1    0   -1    1    0    0
       -1    0   -2    2   -1    2    0    1
 C =
        4    1    2   -3    2    0    0   -1
        0   -2   -3    1    0    1    0    0
        2   -1    1   -1    3    0    1    0
        2    0    2   -2    0   -2    0    0
```

(b)

Figure 1.6 (a) The input signal and (b) The DCT coefficients of the input signal.

1.1.3 Quantization

The fundamental principle of quantization is to quantize a signal with large range to smaller range. For example, the output value of a quantizer can be computed with Equation 1.6. To reconstruct the signal R, Equation 1.7 is applied. In the Equations, the QP is the quantization parameter which controls the step sizes of the quantizer. When QP is large, fewer bits are required to encode the output signals. Otherwise, more bits are required but the distortion introduced by quantization is less. Figure 1.7 shows an example of a three-bit uniform scalar quantizer. In the example, the quantizer has a QP of 32. If the value of the input signal is 66, it will be quantized to 2.

$$\text{Output} = \text{Round}\left(\frac{\text{Input}}{\text{QP}}\right) \tag{1.6}$$

$$R = \text{Output} \times \text{QP} \tag{1.7}$$

It should be noticed that the reconstructed signal is not the same as the input signal. Therefore, quantization is a lossy process. If the distortion introduced during the quantization process is not too large, the reconstructed signal is similar to the input signal. In image and video processing, QP can be used to control the compression and the quality of the reconstructed signal.

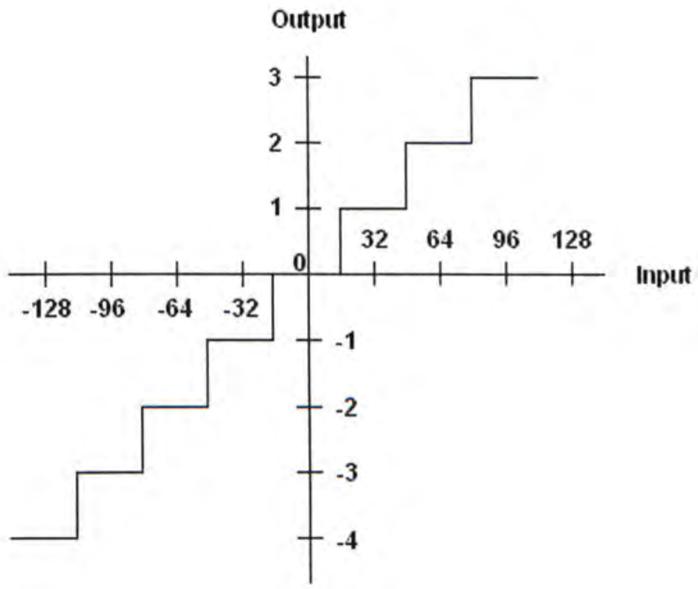


Figure 1.7 A three-bit uniform scalar quantizer.

In addition to scalar quantization, there is another type of quantization called vector quantization. A vector quantizer maps a set of input data to a single codeword. Both encoder and decoder store the same set of vectors in a codebook. Therefore, the vector can be dequantized correctly [18].

1.1.4 Entropy Coding

Entropy coding is a lossless process. One or more input symbols are converted to a single codeword. At the decoder side, the encoded codeword can be decoded back to the original symbols without any distortion. There are many entropy coding methods such as Runlength Coding, Huffman Coding, Arithmetic Coding, etc.

Runlength Coding converts run of zeros to a single codeword. Therefore, when the occurrence of zero in the input data set is high, the data can be compressed efficiently.

Huffman Coding encodes the input symbols based on the probabilities of their occurrences. A short length codeword is assigned to a symbol with higher probability of occurrence and vice versa. The codeword is unique and instantaneously decodable. The average bit per symbol R of Huffman Coding can be computed with Equation 1.8. Huffman Coding can achieve high compression. However, the codeword of each symbol contains an integral number of bits. In reality, the optimal number of bits for a symbol depends on the information content and is generally a fractional number. Consequently, Huffman Coding is only sub-optimal.

$$R = \sum_{i=0}^{N-1} P_i L_i \tag{1.8}$$

where P_i is the probability of occurrence of symbol i,

L_i is the length of the codeword of symbol i, and

N is the total number of symbols

Arithmetic Coding is an entropy coding technique that approaches the entropy limit. A message is represented by an interval of real numbers between 0 and 1. As the message is longer, the interval needed to represent it becomes smaller, and the number of bits needed to specify that interval grows. If the probability of occurrence of a symbol is very high, the interval will be reduced in a smaller amount and vice versa.

Unlike Huffman Coding, Arithmetic Coding converts a sequence of symbols to a single fractional number and therefore it can generally compress the symbols more efficient than Huffman Coding does [18].

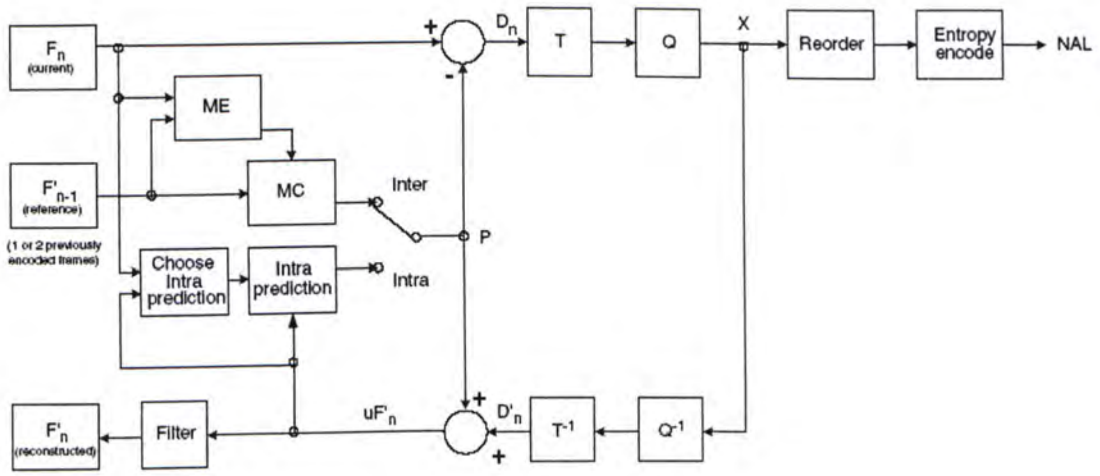
1.2 H.264/MPEG-4 Part 10

H.264/AVC is the latest video coding standard which is developed by the Moving Picture Experts Group and the Video Coding Experts Group (MPEG and VCEG). It outperforms previous video coding standards such MPEG-1, MPEG-2, MPEG-4, H.261, H.263, etc. It is entitled 'Advanced Video Coding' (AVC) and is published jointly as Part 10 of MPEG-4 and ITU-T Recommendation H.264 [19, 20].

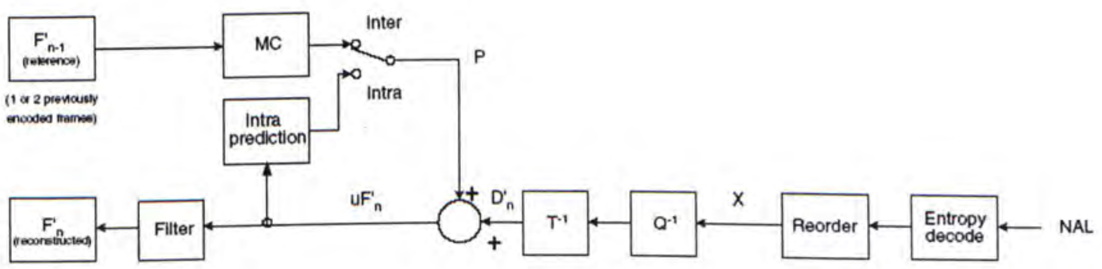
Unlike MPEG-4 which supports object-based video coding, H.264/AVC is frame-based. It encodes video with the highest compression and best quality amongst all existing video coding standards. Applications of H.264 include video telephony or video conferencing, storage, etc.

1.2.1 Overview

H.264 employs various advanced video processing techniques which include variable block size motion estimation and compensation, multiple reference pictures, various modes of intra prediction, deblocking filter, etc. These techniques contribute to the excellent performance of H.264. Even though H.264 outperforms previous video coding standards, H.264 is computational intensive. Consequently, many researchers develop fast algorithms, such as fast motion estimation, fast mode decision, early skip detection, etc, to reduce the complexity of H.264.



(a) The H.264 encoder



(b) The H.264 decoder

Figure 1.8 The H.264 codec [18].

Figure 1.8 (a) shows the H.264 encoder structure. Similar to previous video coding standards, H.264 applies hybrid coding scheme. For I picture, based on the cost function, the best intra prediction mode is selected and the prediction errors D_n are transformed, quantized, reordered and entropy encoded. The quantized Integer Cosine Transform (ICT) coefficients are rescaled and inverse transformed to generate D_n'. The prediction P is added to D_n' to form the unfiltered reconstructed block. Finally, the unfiltered reconstructed block is filtered by a deblocking filter and the output is the reconstructed frame used for temporal prediction. It should be noticed that intra prediction exploits unfiltered reconstructed blocks to perform the prediction.

In H.264, seven different block types are available for motion compensation. A macroblock can be partitioned into 16x16, 8x16, 16x8, or 8x8 blocks. If 8x8 partition size is selected, the 8x8 sub-macroblock can further be partitioned into 8x8, 4x8, 8x4 or 4x4 blocks as illustrated in Figure 1.9. When compared with the fixed block size strategy in previous standards, this variable block size strategy can reduce the temporal prediction errors more significantly. For P or B pictures, the best mode is selected amongst all the enabled intra and inter modes based on the cost function defined in the encoder. After the best prediction is found, the whole process is the same as that of I picture.

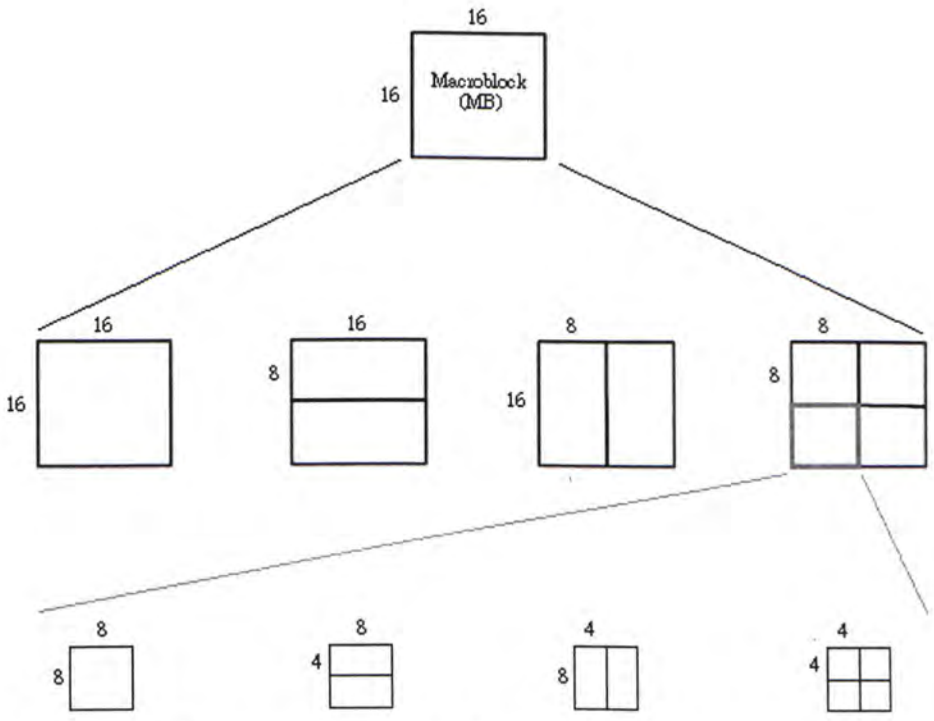


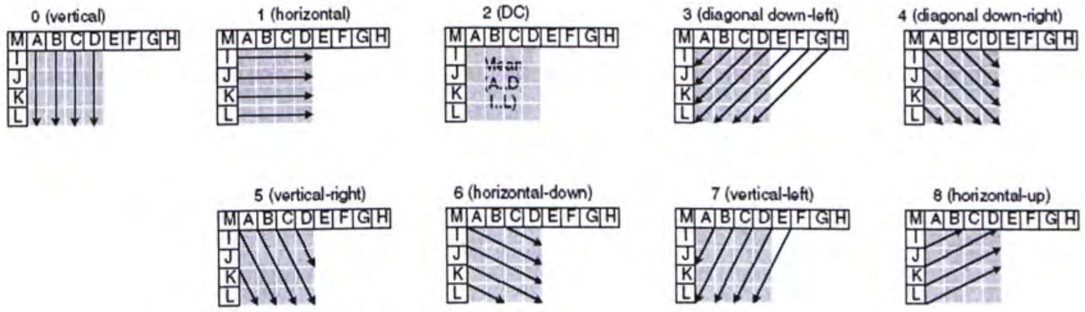
Figure 1.9 The macroblock and sub-macroblock partitions in H.264.

Figure 1.8 (b) shows the H.264 decoder structure. It can be observed that the decoder structure is similar to the reconstruction part of the encoder (bottom parts) except that the blocks "Choose Intra prediction" and "ME" are not present.

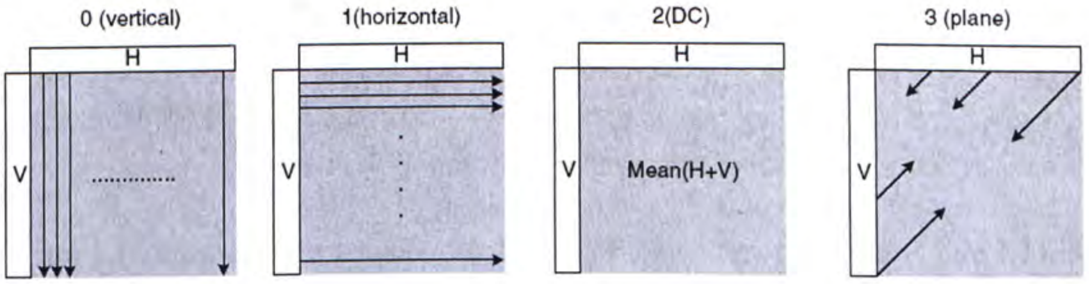
The Baseline Profile of H.264 supports I and P slices only. It supports 17 intra prediction modes which include nine 4x4, four 16x16 luma intra prediction modes and four 8x8 chroma prediction modes. For inter prediction, it supports all the seven block types. In addition, 1/4 and 1/8 pixel motion compensations are supported for luma and chroma components, respectively. The transform and quantization are 4x4 based. The quantized transform coefficients are encoded with Context-based Adaptive Variable Length Coding (CAVLC) and all other syntax elements are encoded with Exponential-Golomb Variable Codes. The ON/OFF of the deblocking filter is optional. In the following sections, the main components of H.264 will be discussed in detail. The discussions focus on the Baseline Profile of H.264.

1.2.2 Intra Prediction

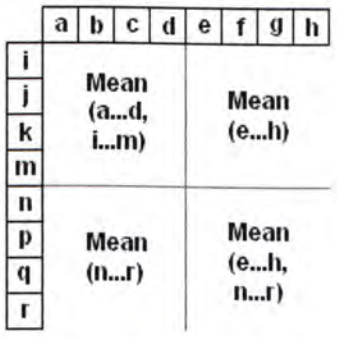
As mentioned before, in H.264 Baseline Profile, there are totally 17 intra prediction modes as illustrated in Figure 1.10. Figure 1.10 (a) shows the nine 4x4 luma intra prediction modes. It should be noted that the prediction samples, i.e. A...M can be the reconstructed samples of current macroblock or previously reconstructed macroblock within the same slice. Figure 1.10 (b) shows the four 16x16 luma intra prediction modes. The chroma intra prediction modes are similar to those of 16x16 luma prediction modes except the DC mode is quite different. Figure 1.10 (c) is an example of chroma DC prediction. In the example, the neighboring prediction samples (a...r) are assumed to be available.



(a) The 4x4 luma prediction modes [18].



(b) The 16x16 luma prediction modes [18].



(c) An example of chroma DC prediction.

Figure 1.10 The intra prediction modes in H.264 Baseline Profile.

1.2.3 Inter Prediction

In H.264, only the motion vector differences which can be derived with Equation 1.9 are encoded. In natural video, motion of neighboring blocks is highly correlated. Therefore, the motion of the current block can be predicted from its neighboring blocks to reduce the bits used to encode the motion information.

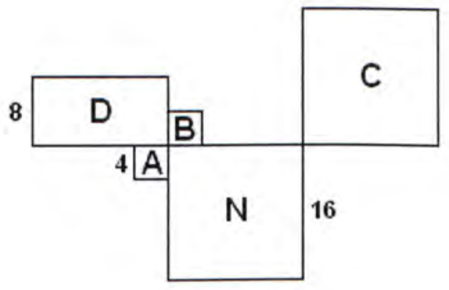
$$MVd = MV - Pred\_MV \qquad (1.9)$$

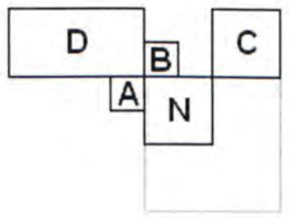
where MVd is the motion vector difference,

MV is the motion vector of the best match, and

Pred_MV is the prediction motion vector which is defined in the H.264 standard.

Figure 1.11 illustrates the neighboring blocks of the current block. In Figure 1.11, N is the current block, A, B, C, and D are the left, upper, upper-right and upper-left blocks of the current block. Pred_MV is mainly the median of the motion vectors of blocks A, B and C. When blocks C is not available, the motion vector of block D is used instead.





(a) (b)

Figure 1.11 The current block N and its neighboring blocks. The block size of N is (a) 16x16 and (b) 8x8.

The deriving processes of Pred_MV for 8x16 and 16x8 blocks are exceptional cases. Figure 1.12 illustrates the prediction for these two cases. The Pred_MV of the upper 8x16 blocks is set to the motion vector of the upper block while the Pred_MV of the lower one is set to the motion vector of the left block only. The Pred_MVs of the left and right 16x8 blocks are set to the motion vectors of the left and upper-right block, respectively.

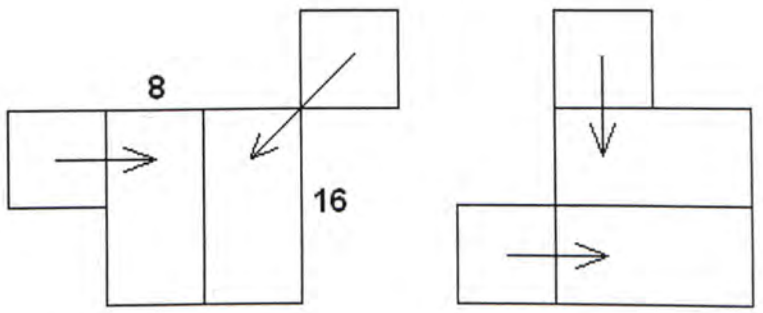


Figure 1.12 The prediction directions of 16x8 and 8x16 block sizes.

All the above examples assume the motion vectors of the neighboring blocks are available. Otherwise, the deriving process of the prediction motion vector is required to be modified.

During motion estimation, the best matches are found amongst all seven block types. The smallest block size of H.264 is 4x4. With such small block size, accurate matching can be achieved. However, this requires a large amount of bits to encode the motion information. Consequently, small block size is not always preferred although prediction errors are small. For the interpolation of the sub-pixel positions, a six tap filter with tap value (1, -5, 20, 20, -5, 1) is employed for luma component. The interpolation process for luma component will be discussed in Section 5.1.2 in detail. Figure 1.13 illustrates the interpolation of the chroma components. N indicates the sample to be interpolated. A, B, C and D are integer chroma samples. The value of N can be computed with Equation 1.10 [30].

$$N = [(8-x)(8-y)A + x(8-y)B + (8-x)yC + xyD + 32] >> 6 \tag{1.10}$$

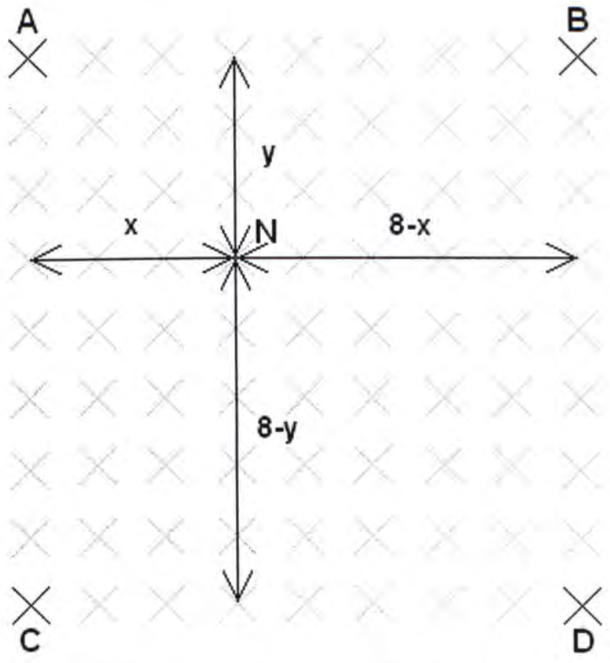


Figure 1.13 The interpolation of chroma 1/8 samples.

1.2.4 Transform and Quantization

H.264 employs Integer Cosine Transform (ICT) and Hadamard Transform instead of Discrete Cosine Transform (DCT). The transformations in H.264 involve integer operations only while DCT requires floating point operations. In addition, ICT can achieve results which are similar to those of DCT. Consequently, ICT can outperform DCT in terms of complexity without significant loss in coding efficiency.

In H.264 Baseline Profile, the 2x2 chroma DC coefficients for 4:2:0 or the 4x4 luma DC coefficients (when 16x16 intra prediction is employed) are further decorrelated with Hadamard Transform. All other prediction errors employ 4x4 ICT. Equation 1.11 and 1.12 are the forward and inverse 4x4 ICT used in H.264 [18].

$$Y = C_f X C_f^{\mathrm{T}} \otimes E_f = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} & & & \\ & \mathrm{X} & & \\ & & & \\ & & & \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix}$$

(1.11)

$$\mathbf{X} = \mathbf{C}_i^{\mathrm{T}} (\mathbf{Y} \otimes \mathbf{E}_i) \mathbf{C}_i = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \left(\begin{bmatrix} & & & \\ & \mathbf{Y} & & \\ & & & \\ & & & \end{bmatrix} \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \right) \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix}$$

(1.12)

where $a = \frac{1}{2}$, $b = \sqrt{\frac{2}{5}}$,

X is the prediction errors matrix, and

Y is the transform coefficients matrix.

In Equations 1.11 and 1.12, there are scaling matrices E_f and E_i. Since they contain floating point number, they cannot be implemented by right shift or left shift. To reduce the complexity, the scaling process is combined into the quantization process.

The transform coefficients are quantized with Equation 1.13 [18].

$$Z_{ij} = round(W_{ij} \bullet \frac{PF}{Q_{step}}) \qquad (1.13)$$

where $W_{ij} = C_f X C_f^T$, and

Z_{ij} is the quantized and scaled coefficients.

Position	PF
(0, 0), (2, 0), (0, 2) or (2, 2)	a^2
(1, 1), (1, 3), (3, 1) or (3, 3)	$\frac{b^2}{4}$
Others	$\frac{ab}{2}$

QP	Q_{step}	QP	Q_{step}	QP	Q_{step}	QP	Q_{step}
0	0.625	7	1.375	18	5	⋮	
1	0.6875	8	1.625	⋮		42	80
2	0.8125	9	1.75	24	10	⋮	
3	0.875	10	2	⋮		48	160
4	1	11	2.25	30	20	⋮	
5	1.125	12	2.5	⋮		51	224
6	1.25	⋮		36	40		

Table 1.1 The quantization step sizes in H.264 codec.

The dequantized coefficients can be obtained with Equation 1.14:

$$W_{ij}' = 64 Z_{ij} Q_{step} PFi \qquad (1.14)$$

Position	PFi
(0, 0), (2, 0), (0, 2) or (2, 2)	a^2
(1, 1), (1, 3), (3, 1) or (3, 3)	b^2
Others	ab

1.2.5 Entropy Coding

Context-based Adaptive Variable Length Coding (CAVLC) and Exponential-Golomb Variable Coding are entropy coding methods adopted in H.264 Baseline Profile. Only prediction errors are encoded with CAVLC, other syntax elements such as motion vector differences, macroblock type, coded block patterns, etc, are encoded with Exponential-Golomb Variable Coding.

The syntax elements used in CAVLC are shown in Table 1.2 [30]. There are totally six tables for parsing the syntax element coeff_token. Two of them are for chroma DC level (4:2:0 and 4:2:2) and the others are for luma level, chroma AC and chroma DC level (4:4:4). A parameter nC determines which table should be used. Table 1.3 illustrates the calculation of nC. nA and nB are the number of non-zero coefficients of the left and upper block of the current block, respectively. If nC is small (nA and nB are small), there is a high probability that the current block contains a small number of non-zero coefficients. Therefore, shorter codes are assigned to small TotalCoeff and vice versa.

The level (non trailing_ones) of a non-zero coefficient consists of two parts, the level_prefix and the level_suffix. The length of the code for level_suffix is determined by a variable called levelSuffixSize. If level_prefix is smaller than 14, levelSuffixSize is set to suffixLength. suffixLength changes mainly according to the magnitude of previously encoded/decoded non-zero coefficient. When level_prefix is equaled to 14 and suffixLength is 0, levelSuffixSize is set to 4. Otherwise, levelSuffixSize is set equal to level_prefix - 3. The reason for doing this is that this can change the length of the code for non-zero coefficient adaptively. Table 1.4 illustrates the codeword of the level_prefix.

Syntax Element	Meaning
coeff_token	TotalCoeff: Total number of non-zero quantized transform coefficient levels TrailingOnes: The number of trailing one quantized transform coefficient levels, maximum number is 3
trailing_ones_sign_flag	1 bit syntax element, when it is equal to: 0: level of the coefficient is +1 1: level of the coefficient is -1
level_prefix	The first part of code of a non-zero coefficient (excluding trailing ones)
level_suffix	The second part of code of a non-zero coefficient
total_zeros	Total number of zeros after the first non-zero coefficients in reverse zig-zag order
run_before	Number of zeros after a non-zero coefficient in reverse zig-zag order (excluding the last non-zero coefficient)

Table 1.2 The syntax elements of CAVLC.

Blocks A and B are available	nC = (nA + nB + 1)>>1
Only block A is available	nC = nA
Only block B is available	nC = nB
Blocks A and B are unavailable	nC = 0

Table 1.3 The calculation of nC.

Level_prefix	bit string
0	1
1	01
2	001
3	0001
4	0000 1
...	...

Table 1.4 The codeword table for level_prefix [30].

The CALVC process is complicated. Interested readers are recommended to read [18, 21 and 30] for details.

When CAVLC is applied, Exponential Golomb codes [22] are used to encode the non-prediction errors syntax elements. Table 1.5 shows the relationship between the bit strings and the range of codeNum. For example, the bit strings for codeNums equaled to 1 and 2 are 0 1 0 and 0 1 1, respectively.

There are four mapping types for Exponential Golomb codes in H.264. They are unsigned (ue), truncated (te), signed (se) and mapped (me). The codeNum of syntax element with ue type is set to its value. te is similar to ue except when the range of the syntax element is equal to 1, only 1 bit is encoded/decoded (bit 0 for codeNum 1 and bit 1 for codeNum 0). The relationship between the codeNum and the se type syntax element value is shown in Table 1.6. me is for coded block pattern. The codeNum for me type syntax element is derived according to the table on pages 202-204 of [30].

Bit string form	Range of codeNum
1	0
0 1 x_0	1~2
0 0 1 x_1 x_0	3~6
0 0 0 1 x_2 x_1 x_0	7~14
0 0 0 0 1 x_3 x_2 x_1 x_0	15~30
0 0 0 0 0 1 x_4 x_3 x_2 x_1 x_0	31~62
…	…

Table 1.5 The relationship between bit strings and codeNum ranges [30].

codeNum	Syntax element value
0	0
1	1
2	-1
3	2
4	-2
k	$(-1)^{k+1}$round(k/2)

Table 1.6 The relationship between codeNum and syntax element value for se type

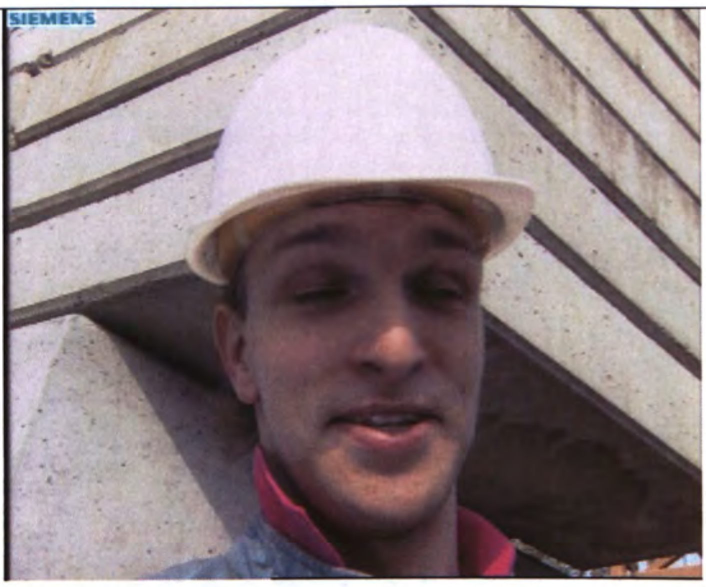
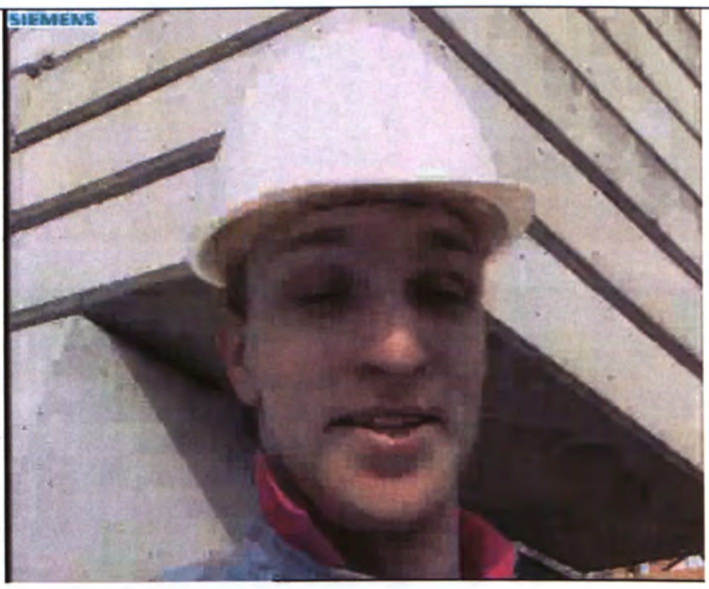
[30].

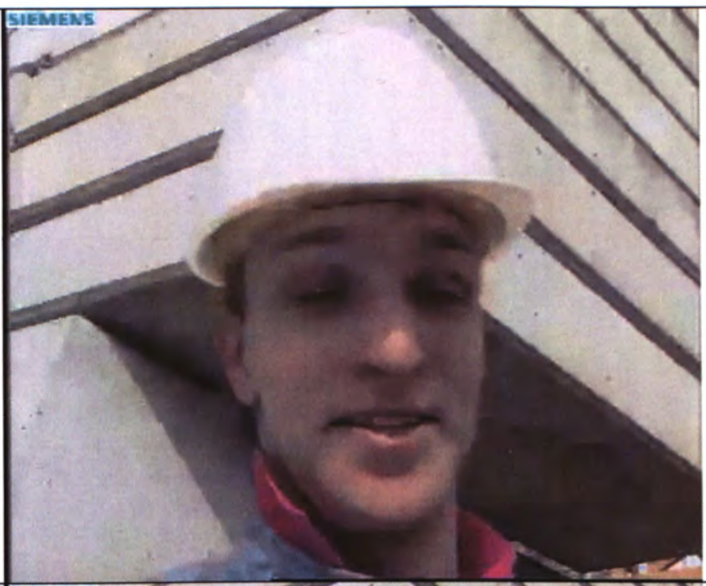
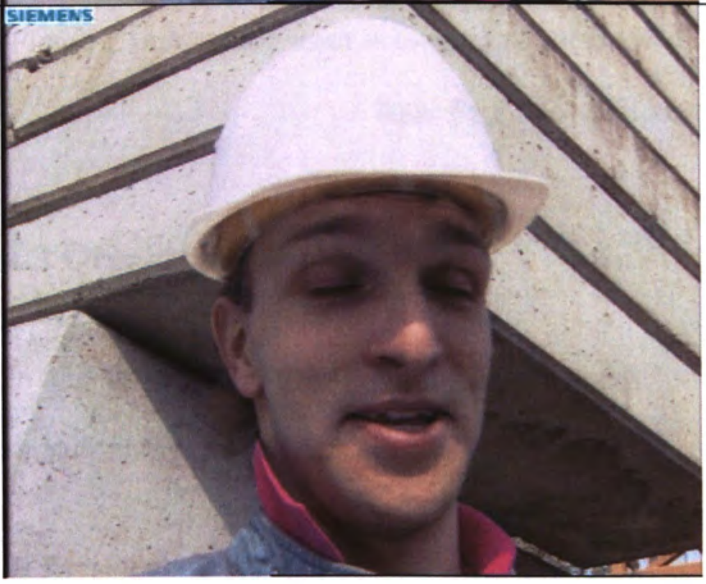
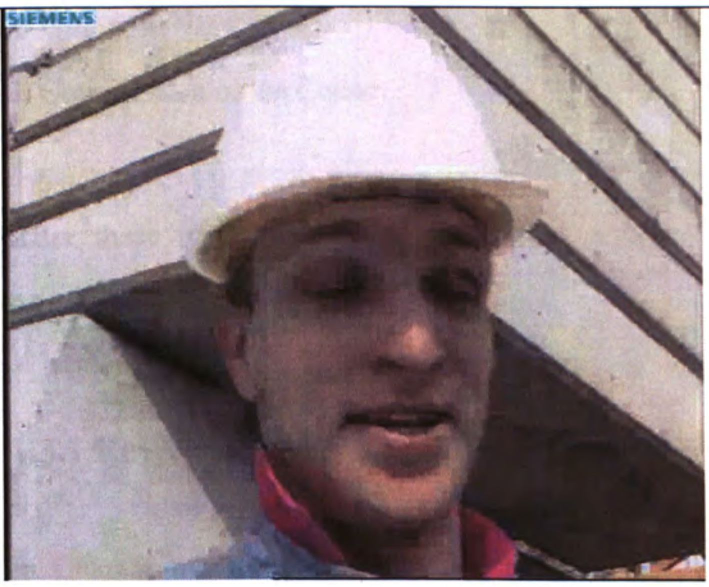
The design of these mappings aims to produce short bit strings for frequently occurring values and longer bit strings for the others [18]. For example, motion vector difference value of 0 occurs frequently and thus it is encoded with 1 bit only while the other less frequently occurring values are encoded with more than 1 bit.

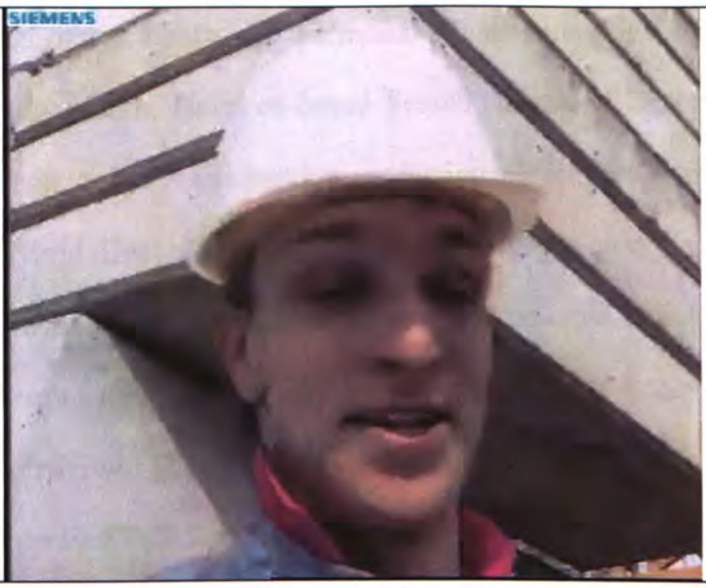
1.2.6 Deblocking Filter

Since the video coding scheme employed in H.264 is block based, blocking artefacts occur especially when the compression is high. In H.264, an optional in-loop filter can be applied to filter the reconstructed picture. This can smooth the blocking artefacts and cause the reconstructed picture to be more pleasing to view visually. In addition, the temporal prediction can be more accurate. Consequently, the in-loop filter can improve both coding efficiency and subjective quality of the reconstructed picture with a little increment in complexity. Figure 1.14 shows an experiment to investigate about the effect of applying the deblocking filter for QP equaled to 36 on Common Intermediate Format (CIF) video. The compression ratio for both cases are nearly equal while the (peak-signal-to-noise ratio) PSNR of the reconstructed frames with deblocking filter is 0.3965dB higher than those without deblocking filter. It is easy to notice that the subjective quality of the reconstructed frames with deblocking filter is much higher than those without deblocking filter. The total encoding time for the case with deblocking filter and without deblocking filter were 1.766s and 1.718s, respectively (increased by about 2.8 %).

The deblocking filter adopted in H.264 applies to luma and chroma pixel locations which are multiples of four only because the transformation and quantization are 4x4 based for the Baseline Profile. The filter is adaptive. It determines whether the edge is a real edge before filtering it. This can retain the features and reduce the blocking artefacts of the reconstructed picture.

	(a) Original Frame 0.
	(b) Intra-coded Frame 0 without deblocking filter. PSNR: 32.749 dB

	(c) Intra-coded Frame 0 with deblocking filter. PSNR: 33.126 dB
	(d) Original Frame 1.
	(e) Hybrid-coded Frame 1 without deblocking filter. PSNR: 31.939 dB



(f) Hybrid-coded Frame 1 with deblocking filter.

PSNR: 32.355 dB

Figure 1.14 An experiment to investigate about the effect of applying the deblocking filter for QP equaled to 36.

1.3 Organization of the Thesis

In the following chapters, this thesis is organized into mainly three parts. The organization is shown as follows:

1. Review of Motion Estimation Techniques
2. Proposed Algorithms
3. Optimization of the Codec

After these parts are discussed, there is a conclusion. In addition, the future development of the algorithms is discussed.

1.3.1 Review of Motion Estimation Techniques

In Chapter 2, different motion estimation (ME) algorithms are reviewed, which include integer pixel, sub-pixel or even variable block size motion estimation algorithms. Firstly, the traditional motion estimation algorithms, which include Three Step Search, Hexagon-based Search, Diamond Search, etc, are briefly described. Then, some novel motion estimation algorithms which include Fast Full Search, Hybrid Unsymmetrical-cross Multi-Hexagon-grid Search (UMHS), Center biased Fractional Pel Search and Enhanced Predictive Zonal Search (EPZS) are reviewed. Due to their low complexity and high accuracy, they are adopted in the reference software of H.264 as fast motion estimation algorithms. UMHS and EPZS are faster than Fast Full Search by about 85% without significant loss in coding performance.

1.3.2 The Proposed Algorithms

The proposed algorithms are introduced in Chapters 3, 4 and 5. In Chapter 3, an intra prediction method which is similar to motion estimation and compensation is reviewed. Several techniques, which include padding, multiple best matches, different modes, etc., are proposed to enhance the performance of the intra prediction method. Besides, fast algorithms are developed to reduce the complexity of the proposed techniques without degrading the coding performance [31].

In Chapter 4, a fast variable block size motion estimation algorithm is proposed. This algorithm exploits the characteristics of variable block size motion estimation. It reuses the SADs within a macroblock and uses pattern-based motion estimation and refinement search to reduce the complexity with a little degradation of coding performance in terms of PSNR and bitrate.

1.3.3 Optimization of the Codec

Chapter 5 discusses the development of a Pocket PC based real-time H.264 codec. In that chapter, two algorithms are introduced. One is a fast interpolation algorithm which aims to reduce the complexity of the interpolation process without degrading the performance. The other is a fast sub-pixel motion estimation algorithm that can reduce the complexity significantly with only a little degradation of coding performance.

The optimization of the H.264 Baseline Profile is also discussed. The optimization focuses on programming optimization techniques. With these techniques and the mentioned fast algorithms, the developed Pocket PC based H.264 Baseline Profile codec can encode and decode videos in real time with only a little degradation of coding performance when compared with that of the reference software. Due to limited resources of Pocket PC, it is difficult for the highly complex H.264 codec to run in real time. Consequently, this is a tremendous achievement. The developed codec can be applied for different applications such as video telephony, video conferencing, etc.

At the end of Chapters 3, 4 and 5, there are experimental results to investigate the performance of the proposed algorithms and the developed Pocket PC based real-time H.264 codec. The results focus on analyzing the complexity and the coding performance.

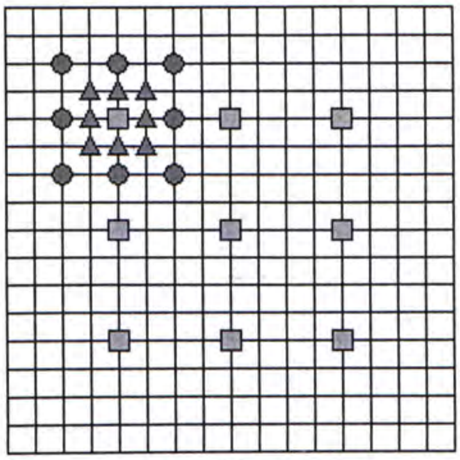
1.4 Contributions

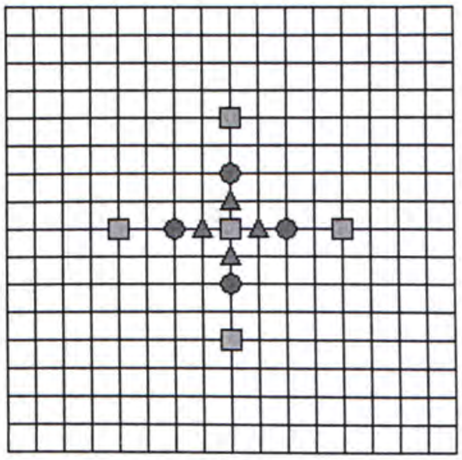
This thesis describes algorithms for improving and optimizing the latest video coding standard H.264/AVC. Even though tremendous coding performance is achieved, H.264 is computational intensive. Therefore, algorithms aim to reduce the complexity are proposed. Besides, there is room to further improve the coding performance of H.264. Accordingly, algorithms which are designed for improving the coding performance are proposed.

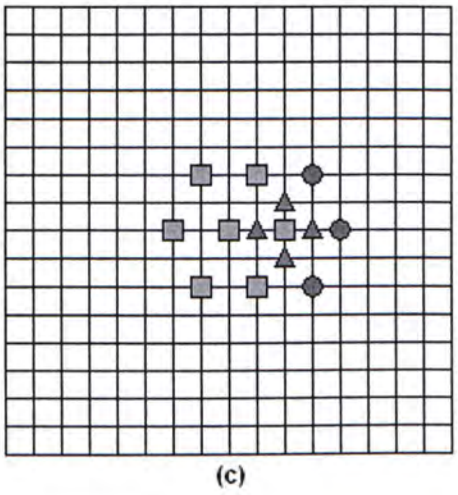
Several enhancement techniques are proposed in Chapter 3 to improve the coding performance of intra block matching. In Chapter 4, a fast variable block size motion estimation algorithm, which reduces the complexity of variable block size motion estimation significantly, is proposed. This algorithm is the fastest when compared with the fast motion estimation algorithms adopted in the H.264 reference software. The development of a Pocket PC software-based real-time H.264 codec is described in Chapter 5. During the development, all the codes are re-written instead of using those of the reference software. This allows us to design the structure of the codec and redundant features and operations can be avoided easily. Different optimization techniques are proposed and applied. Eventually, the developed codec can encode and decode video in real-time (more than 25 fps) on Pocket PC. This is a tremendous achievement because the complexity of H.264 is so high that it is not easy to develop a software-based real-time H.264 codec on Pocket PC platform.

Chapter 2 Review of Motion Estimation Techniques

In this chapter, several algorithms are reviewed. They focus on integer and sub-pixel motion estimation. Full Search is an exhaustive search technique which evaluates all the search points within the search window. Accordingly, its complexity is very high and fast motion estimation algorithms are required. Typical fast motion estimation algorithms include three step search (TSS), 2-D logarithmic search (2-D LOGS) [13], Hexagon-based Search (HEXBS) [14], Diamond search (DS) [15], etc. Since they evaluate fewer search points than Full Search does, they are much faster. However, they are susceptive to be trapped in local minimum and this introduces large prediction errors. Their search patterns are shown in Figure 2.1. In the figures, the squares indicate the first step; circles indicate the second step and triangles are the last one. The search range is ± 8.


(a)


(b)



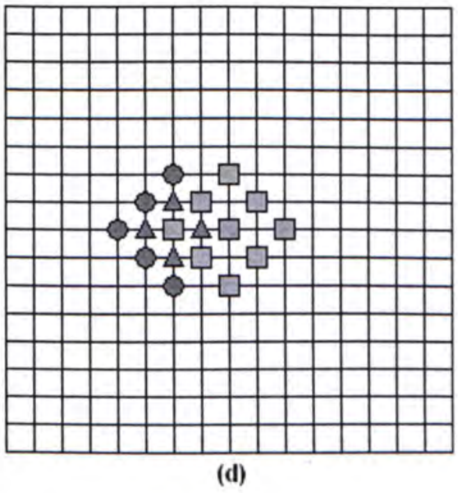


Figure 2.1 Different search patterns of fast motion estimation algorithms, (a) TSS, (b) 2-D LOGS (c) HEXBS and (d) DS.

The above algorithms are not specially designed for H.264. In this chapter, algorithms which are designed or adopted in H.264 are introduced. These algorithms include Fast Full Search (FFS), Hybrid Unsymmetrical-cross Multi-Hexagon-grid Search (UMHS), Center biased Fractional Pel Search (CBFPS) and Enhanced Predictive Zonal Search (EPZS). Fast Full Search is specially designed for variable block size motion estimation. UMHS are designed based on the characteristics of natural video. CBFPS are sub-pixel motion estimation algorithm. EPZS employs predictor selection.

2.1 Fast Full Search

Fast Full Search (FFS) can provide coding performance which is similar to that of Full Search. It is specially designed for variable block size motion estimation. There are totally seven different block sizes in H.264. Motion estimation is performed for each of them. Therefore, the complexity of motion estimation process of encoder will be high if Full Search is applied. Consequently, Fast Full Search is designed to reduce the complexity while achieving similar coding performance.

As mentioned in Chapter 1, motion estimation requires a cost function to find the best match. In the reference software of H.264, the cost function for motion estimation consists of two parts. One is the cost contributed by encoding the motion vector difference. The other is the sum of absolute difference (SAD) which is for measuring the similarity between the current block and the reference block. SAD computation is very time consuming. Accordingly, if the number of SAD computations can be reduced, the complexity of motion estimation can be reduced significantly. The objective of Fast Full Search is to reduce repetitive SAD computations during variable block size motion estimation.

With the same search center for all block types within a macroblock, SAD can be reused. At the beginning, the macroblock is partitioned into sixteen 4x4 blocks and the SADs (SAD_{4x4}) of each of them within the search window are pre-computed and stored. Consequently, the stored SAD_{4x4}s can be used to construct the true SADs for different block sizes during variable block size motion estimation as illustrated in Figure 2.2. Thus, the complexity can be reduced significantly.

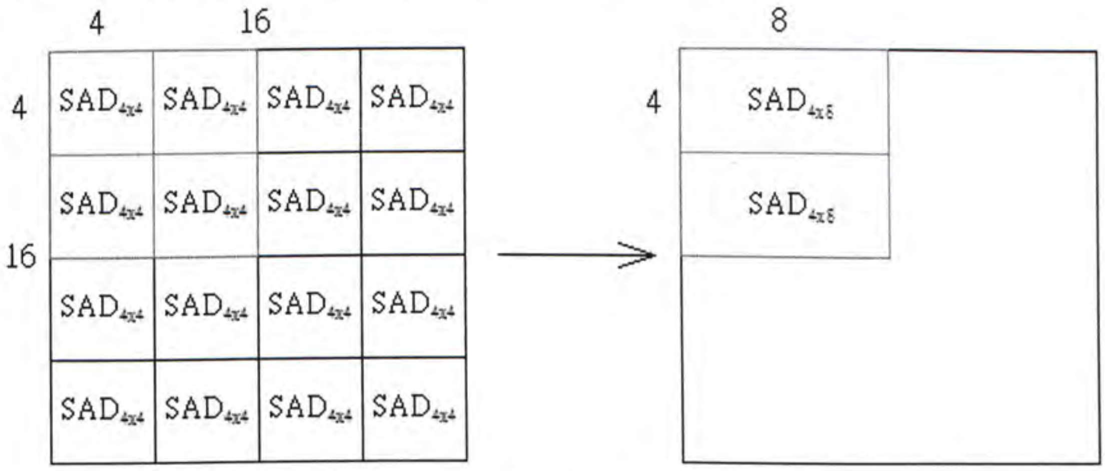


Figure 2.2 Construction of the SAD of 4x8 blocks from the stored SAD_{4x4}s.

In the reference software, the search center for Full Search is the prediction motion vector, which is defined in the standard, of each block. Consequently, different block within a macroblock can have different search center. Since Fast Full Search can only have a fixed search center, the prediction motion vector for the 16x16 block type is selected. The search center affects the cost of the motion vector differences. Therefore, the coding performance of Fast Full Search and that of Full Search is not the same. However, they are very similar. Therefore, Fast Full Search can benefit the encoder in terms of complexity.

2.2 Hybrid Unsymmetrical-cross Multi-Hexagon-grid Search

The Hybrid Unsymmetrical-cross Multi-Hexagon-grid Search (UMHS) is a fast integer pixel motion estimation algorithm proposed in [2] and it is adopted in the H.264 reference software as one of the fast motion estimation algorithms. It is faster than Fast Full Search and generally can achieve coding performance which is similar to that of Fast Full Search.

UMHS includes several steps. The first step is to find an initial search point prediction. Different modes have different prediction candidates which include the median prediction defined in the H.264 standard, (0, 0) vector prediction and motion vectors of the upper layers. The upper layers of current mode are the modes with larger block sizes. This strategy aims to utilize the motion relationship between different modes. The prediction candidate with the minimum cost is selected as the initial search point prediction.

After the initial search point prediction is found, mainly four steps are performed with different kinds of search pattern as illustrated in Figure 2.3. First, an unsymmetrical-cross search is performed. Based on the argument that vertical motion is less likely than horizontal motion for natural video, the search pattern is designed to search more points in horizontal direction and search fewer points in vertical direction as illustrated in step 2 of Figure 2.3. The search point with the minimum cost is chosen as search center for the next step.

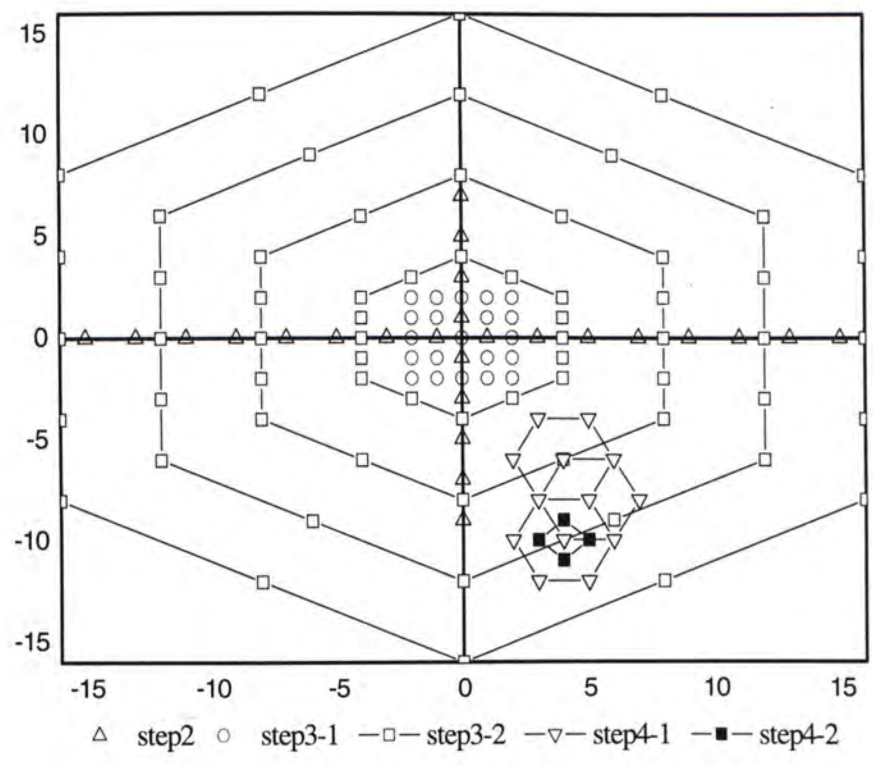


Figure 2.3 Search process of HUCMHGS algorithm, search range is 16 [2].

A full search with search range equaled to 2 is carried out around the search center as illustrated in step 3-1 of Figure 2.3. Then an Uneven Multi-Hexagon-grid Search strategy is applied as illustrated in step 3-2 of Figure 2.3 and Figure 2.4. As shown in the figures, the pattern contains fewer search points in vertical direction than those in horizontal direction and therefore it is called uneven. This design is based on the same argument mentioned before, i.e. vertical motion is less likely than horizontal motion for natural video. Finally, with the best search point in the previous step as search center, a search process which is similar to that of Hexagon-based Search is performed to find the best integer pixel motion vector as illustrated in step 4 of Figure 2.3 [2].

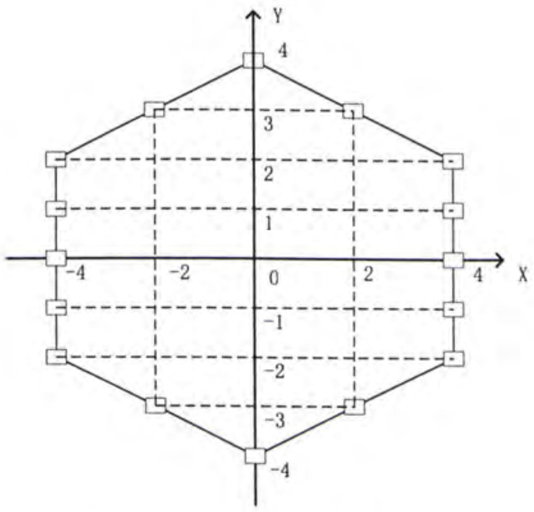


Figure 2.4 The basic search pattern of Uneven Multi-Hexagon-grid Search [2].

2.3 Center biased Fractional Pel Search

The Center biased Fractional Pel Search (CBFPS) is a fast sub-pixel motion estimation algorithm proposed in [2]. It utilizes the assumption that the unimodal error surface assumption is true in most cases for fractional pixel search [12].

$$\text{frac\_pred\_mv} = (\text{pred\_mv} - \text{mv})\%\beta \qquad (2.1)$$

where frac_pred_mv is the fractional prediction motion vector,
pred_mv is the prediction motion vector,
mv is the best integer pixel motion vector of the current block which is in fractional pixel unit,
% is the modulus operation, and
$\beta = 4$ for quarter pixel case.

Firstly, a prediction motion vector, frac_pred_mv, is derived with Equation 2.1. Then the costs of (0, 0) and frac_pred_mv are compared and the one with the minimum cost is chosen as the search center for the next step. Finally, the small diamond search pattern is employed to find the best match as illustrated in Figure 2.5. The search center is updated to the best search point (the search point with the minimum cost) at each stage if that best search point is not located at the search center. Otherwise, the search process is completed and the best search point is the best match [2].

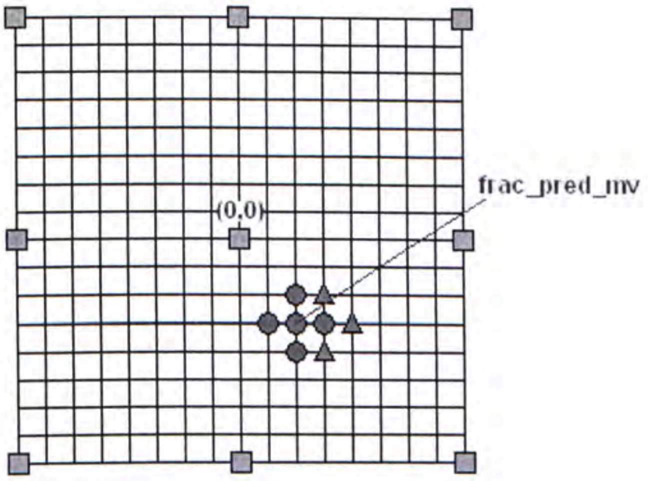


Figure 2.5 Implementation of CBFPS algorithm, squares are the integer pixel positions.

2.4 Enhanced Predictive Zonal Search

The Enhanced Predictive Zonal Search (EPZS) [3, 16, 17] mainly comprises three features which include the initial predictor selection, the adaptive early termination and the final prediction refinement. It is much faster than Full Search and even Fast Full Search without significant loss in coding performance. Same as UMHS, it is also adopted by the reference software of H.264 as one of the fast motion estimation algorithms.

The predictor selection can be considered as the key feature of EPZS. The predictors can be selected by exploiting temporal and spatial correlation. The predictors of EPZS include the median predictor which is the prediction motion vector defined in the H.264 standard, other spatial predictors, temporal predictors and a set of search range dependent predictors. In [16], the spatial predictors include the median predictor, the left, upper, upper-left, upper-right block motion vectors as illustrated in Figure 2.6 and the (0, 0) predictor. In addition to spatial predictors, temporal predictors are used in EPZS which include the co-located motion vector, 8 motion vectors of the adjacent blocks which are around the co-located block, etc. A set of search range dependent predictors is also defined to better cover the large search ranges for high motion videos. The two possible search range dependent predictor sets in [17] are illustrated in Figure 2.7. The centers in Figure 2.7 can be either at (0, 0) or the median predictor [16].

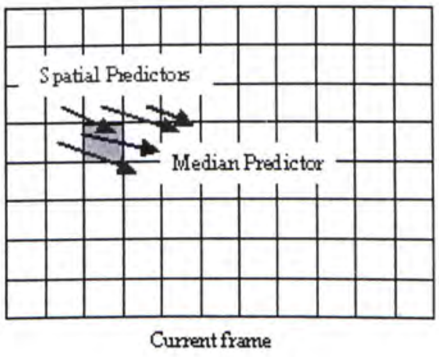


Figure 2.6 Spatial predictors of EPZS [16].

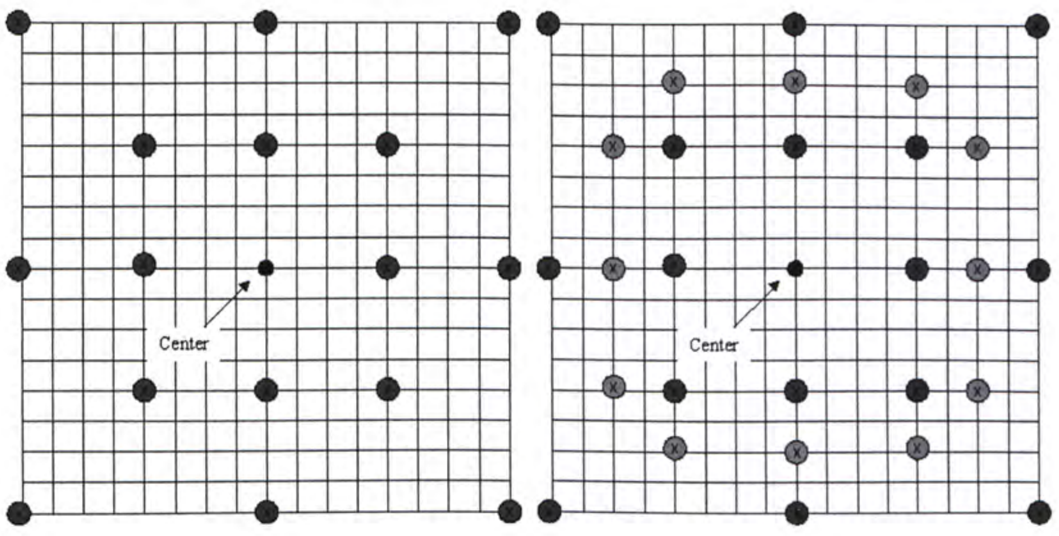


Figure 2.7 Two possible search range dependent predictor sets of EPZS with search range equaled to 8 [17].

The adaptive early termination of EPZS enables an extra reduction in complexity of the motion estimation process. If the cost of the median predictor is smaller than a threshold T_1, the motion estimation process is terminated without examining any other predictors. In this case the median predictor is defined as the best integer pixel motion vector. Otherwise, all other predictors are examined. If the minimum cost of the predictors is smaller than another threshold T_2, the search is terminated. Otherwise, motion vector refinement is performed with the best predictor as search center.

The search patterns employed by EPZS include small diamond pattern, square pattern (EPZS^2) and extended EPZS pattern (extEPZS) as shown in Figure 2.8. It can be observed that the small diamond pattern provides least complexity while the extEPZS provides best coding performance amongst the patterns. If the best search point is at the search center, it is defined as the best integer pixel motion vector.

Otherwise, the search center is set to the best search point in the current step. The process is repetitively performed until the best search point in the current step is at the search center.

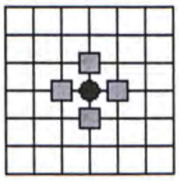
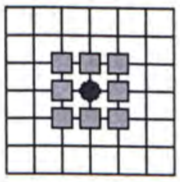
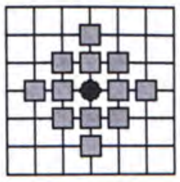
		
(a)	(b)	(c)

Figure 2.8 (a) Small diamond pattern, (b) Square pattern (EPZS^2) and (c) Extended EPZS pattern (extEPZS) [17].

Besides the above mentioned features, EPZS has other features. Therefore, interested readers are recommended to read [3, 16, 17] for more details about EPZS.

Chapter 3 Enhancement Techniques for Intra Block Matching

In this chapter, a novel intra block matching algorithm [23, 24] is briefly reviewed. It is similar to motion estimation and compensation technique. For motion estimation and compensation, the pixels of the current macroblock are predicted from the pixels of other pictures. This algorithm predicts pixels of the current block from the reconstructed pixels of the current slice. In this chapter, this algorithm is called intra block matching.

Several enhancement techniques for further improving the algorithm are proposed. The proposed techniques include best match prediction, multiple best matches, novel padding method, skip mode, etc. Experimental results show that the coding performance can be improved significantly and more than 1 dB and 0.6 dB gains in PSNR can be achieved for intra and hybrid coding, respectively.

In the following sections, the reason and the whole process of applying intra block matching are first discussed. Then the proposed enhancement algorithms, modes and fast algorithms are described in detail. After the description, experimental results are shown at the end of this chapter to illustrate the performance of the proposed techniques. Finally, there is a discussion on the experimental results.

3.1 Introduction

3.1.1 Fundamental Principles

The intra prediction techniques adopted in H.264 predict the intensities of the pixels of the current block from its neighboring pixels. This aims to exploit the spatial correlation in a slice. However, when the texture of the reconstructed pixels is not smooth, using neighboring pixels to predict the pixels of the current block may not be accurate. Consequently, encoding of intra slices requires a large number of bits. To improve the coding efficiency, intra block matching technique is applied with several proposed enhancement techniques.

As mentioned before, the main idea of intra block matching is to predict the pixels of the current block from the pixels in the current slice within a search region. Therefore, similar to motion vector, an offset vector is required to indicate the position of the best match. In this chapter, that vector is called intra displacement vector (Intra_DV). The defined search region in our experiment is illustrated in Figure 3.1. In the figure, all the pixels that are reconstructed or padded, and within macroblocks "P" and "C" are used for prediction.

The intra block matching process is carried out as follows:

1. Find a best match for the current block within the search region. (The cost function defined in our experiment is shown in Equation 3.1)
2. After the best matches of all the intra block matching modes are found, the total $\text{costs}_{\text{mode}}$ (cost for mode decision) of all the available modes of the encoder are compared with each other.
3. When the best one is intra block matching mode, the corresponding best matches are used for prediction.
4. Prediction errors are transformed, quantized, encoded and reconstructed as usual.

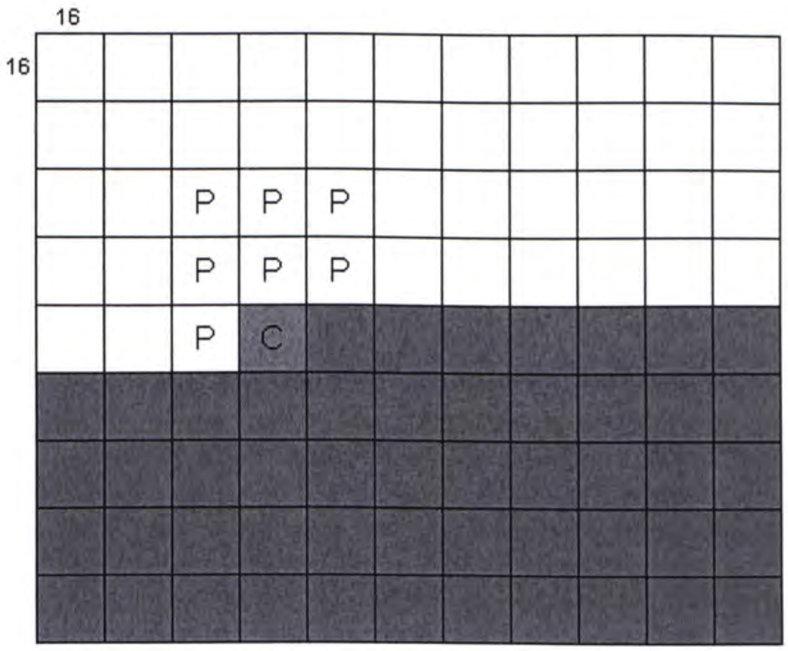


Figure 3.1 White blocks are reconstructed blocks, block "C" and "P" are the current macroblock and the pixels used for prediction, respectively, and the shaded macroblocks are not encoded and thus are not reconstructed.

$$J = SAD + \lambda_{displacement}\, bits_{Intra\_DVd} \tag{3.1}$$

where J is the cost of a candidate search point,

SAD is the sum of absolute differences between two blocks,

$\lambda_{displacement}$ is the Lagrange Multiplier, and

$bits_{Intra\_DVds}$ is the number of bits used to encode the Intra_DV differences (Intra_DVds).

3.1.2 Variable Block Size Intra Block Matching

Similar to motion compensation of H.264, intra block matching can also employ variable block size technique. The reason of partitioning a macroblock to perform intra block matching is that when 16x16 intra block matching is performed, the prediction errors are large but less overhead information is encoded for the

Intra_DVds. When 4x4 intra block matching is performed, the prediction errors are reduced but more overhead information is encoded. Therefore, variable block size can provide a good balance between prediction errors and the amount of overhead information for different situations.

In [24], seven different block sizes are employed. They include all the block sizes employed in the motion compensation process of H.264. Even though accurate prediction can be found, much processing power is required. In contrast, fewer number of block sizes is employed in [23]. The complexity is definitely less but the prediction may not be very accurate. In our design, the current macroblock can be partitioned into mainly four different block sizes as illustrated in Figure 3.2. Each macroblock can be partitioned into one 16x16, four 8x8, sixteen 4x4 blocks or into four Combine blocks. Each Combine block is a sub-macroblock and can further be partitioned into one 8x8 block or four 4x4 blocks. We found that adding more block sizes cannot achieve much gain in coding performance but increase the complexity in a great amount. Accordingly, only four block sizes are adopted to achieve a good balance among prediction errors, complexity and the amount of overhead information.

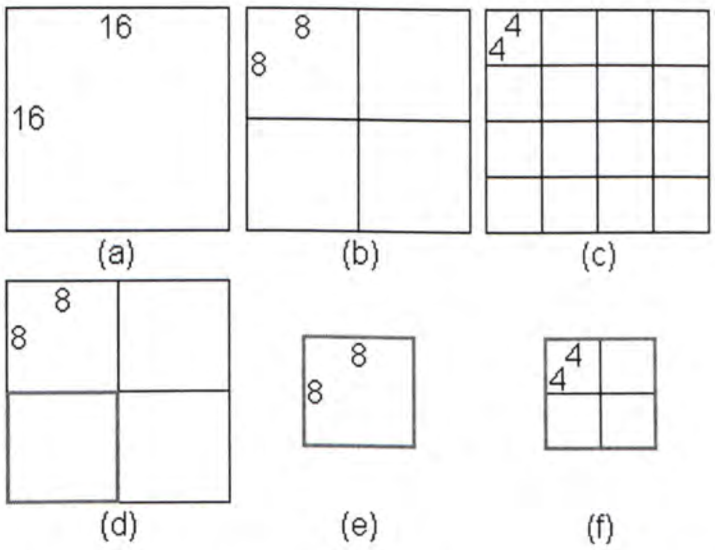


Figure 3.2 The four variable block sizes, (a) 16x16, (b) 8x8, (c) 4x4, and (d) Combine. For the Combine block size, each sub-macroblock can be partitioned into (e) one 8x8 block or (f) four 4x4 blocks to perform intra block matching.

3.2 Proposed Techniques

3.2.1 Padding

The current block is padded from the neighboring pixels before intra block matching to maximize the number of candidates. Let the position of the pixel in the top-left corner of the block to be padded and the current block are (xp, yp) and (x, y), respectively, imgY represents current (reconstructed) slice and xO, yO are the intra padded block dimensions. They can be smaller than the current block size. In our experiment, for 4x4 and 8x8, xO = 4 and yO = 4. For 16x16, xO = 8 and yO = 8. This is due to the spatial distances between the bottom right corner pixels and the neighboring pixels of the current block are longer when the block sizes are large. The padding is classified into three types, the DC, horizontal and vertical padding. To avoid encoding the overhead information, the costs of these three padding types are computed and compared. The one with the minimum cost is selected as the best padding type. The same cost evaluation and comparison are carried out in both encoder and decoder side. Therefore, no extra overhead information is required. The padding process is performed until the whole current block is padded by changing (xp, yp) iteratively. In iteration, the following process is performed.

When neither the left nor the upper block is available (When a block is located inside the current slide and has already been encoded and reconstructed, it is available. Otherwise, it is unavailable.), DC intra padding method is defined as the best and imgY (r, s) with r = xp...xp + xO - 1 and s = yp...yp + yO – 1 is derived with Equation 3.12.

When at least one of the left and upper blocks is available, three padding costs are computed for finding the best padding type amongst the three padding types. The padding costs of DC (PadCost0), horizontal (PadCost1) and vertical (PadCost2) padding types change according to the availabilities of the left and upper blocks.

For PadCost1 and PadCost2, when the involved neighboring block (left block for horizontal padding and upper block for vertical padding) is unavailable, they are set to infinite. Otherwise, they are derived with Equations 3.2 and 3.3, respectively. When horizontal or vertical padding is the best amongst the three padding types, imgY(r, s) is set to imgY(x-1, s) or imgY(r, y-1), respectively.

For DC padding, as mentioned before, PadCost0 changes according to the availabilities of the neighboring blocks. The computation of PadCost0 and the DC padding value, PadDC, is performed based on the following three different situations.

1. Both the left and upper blocks are available.
2. Only the left block is available.
3. Only the upper block is available.

For case 1, Equations 3.4, 3.5 and 3.6 are used to derive PadCost0 and Equation 3.7 is used to derive PadDC. For case 2, Equations 3.5 and 3.8 are used to derive PadCost0 and Equation 3.9 is used to derive PadDC. For case 3, Equations 3.4 and 3.10 are used to derive PadCost0 and Equation 3.11 is used to derive PadDC. When DC padding type is the best, imgY(r, s) is set to PadDC.

imgY(r,s) (the padded pixels) are used for the searching process of intra block matching only. After the best match is found, the padded pixels are replaced by the best prediction pixels.

$$\text{PadCost1} = \sum_{i=yp}^{yp+yO-1} \sum_{a=1}^{2} |\,(\text{imgY}(x-3,i) - \text{imgY}(x-a,i)\,| \tag{3.2}$$

$$\text{PadCost2} = \sum_{i=xp}^{xp+xO-1} \sum_{a=1}^{2} |\,(\text{imgY}(i,y-3) - \text{imgY}(i,y-a)\,| \tag{3.3}$$

$$\text{Pad0} = \text{round}\left(\frac{\sum_{i=xp}^{xp+xO-1} \text{imgY}(i,y-3)}{xO}\right) \tag{3.4}$$

$$\text{Pad1} = \text{round}\left(\frac{\sum_{i=yp}^{yp+yO-1} \text{imgY}(x-3,i)}{yO}\right) \tag{3.5}$$

$$\text{PadCost0} = \text{round}\left(\frac{\sum_{i=xp}^{xp+xO-1} \sum_{a=1}^{2} |\,\text{imgY}(i,y-a) - \text{Pad0}\,| + \sum_{j=yp}^{yp+yO-1} \sum_{b=1}^{2} |\,\text{imgY}(x-b,j) - \text{Pad1}\,|}{2}\right) \tag{3.6}$$

$$\text{PadDC} = \text{round}\left(\frac{\sum_{i=xp}^{xp+xO-1} \text{imgY}(i, y-1) + \sum_{j=yp}^{yp+yO-1} \text{imgY}(x-1, j)}{xO + yO}\right) \tag{3.7}$$

$$\text{PadCost0} = \sum_{i=yp}^{yp+yO-1} \sum_{a=1}^{2} |\,\text{imgY}(x-a, i) - \text{Pad1}\,| \tag{3.8}$$

$$\text{PadDC} = \text{round}\left(\frac{\sum_{i=yp}^{yp+yO-1} \text{imgY}(x-1, i)}{yO}\right) \tag{3.9}$$

$$\text{PadCost0} = \sum_{i=xp}^{xp+xO-1} \sum_{a=1}^{2} |\,\text{imgY}(i, y-a) - \text{Pad0}\,| \tag{3.10}$$

$$\text{PadDC} = \text{round}\left(\frac{\sum_{i=xp}^{xp+xO-1} \text{imgY}(i, y-1)}{xO}\right) \tag{3.11}$$

$$\text{imgY}(r, s) = (1 << (\text{BitDepth} - 1)) \tag{3.12}$$

where round () indicates rounding the result to the nearest integer, and

BitDepth is the number of bits used to represent each luma component.

Figure 3.3 is an example to illustrate the DC intra padding process of an 8x8 block for xO = 4 and yO = 4. N is the block to be padded. The color pixels within N are the pixels to be padded. The pixels with blue colour are the neighboring pixels. Through the above process, the current block is padded. In addition to the reconstructed pixels of the current slice, the pixels of this padded block are used for intra block matching.

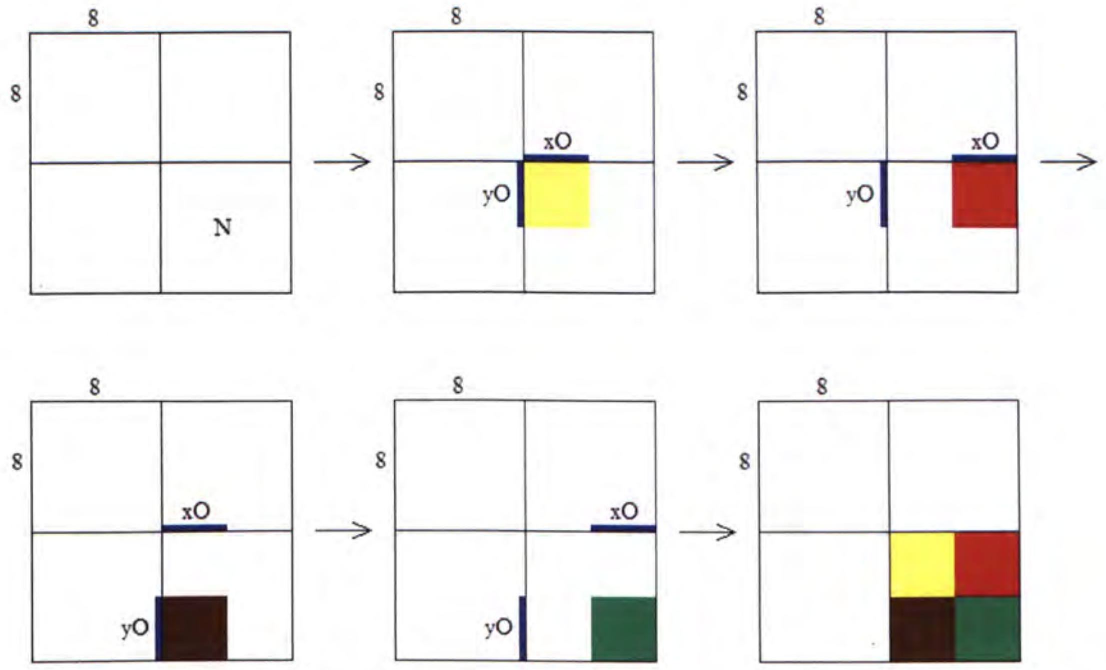


Figure 3.3 An example of DC intra padding process of an 8x8 block with xO = 4 and yO = 4.

The above padding process is for luma component. The intra padding process for chroma components is the same as that for luma except that the intra padding block sizes are scaled according to input file format. For example, when the input file format is 4:2:0, the intra padded block dimensions for chroma components are xO/2 and yO/2.

3.2.2 Modes

In our design, there are totally nine modes for intra block matching. They are summarized in Table 3.1. The terminology used in Table 3.1 is shown in Table 3.2.

Table 3.3 shows the priority of each mode. In the table, the mode with higher priority has a smaller priority number. A macroblock type with higher priority means that fewer bits are used to encode it. Experimentally, COMBINE and DIRECT_PRED have high selection probability and thus their priority is set to be the highest. For the remaining modes, high priority is assigned to INT or SUB 16x16, 8x8 because their overhead information is less when compared with that of 4x4. The priority of INT or SUB 4x4 is set to be higher than that of INT_SUB_4x4 because INT_SUB_4x4 requires one more flag to indicate using INT or SUB for each sub-macroblock. It can be noticed that for SUB, the integer Intra_DVds are required to be multiplied by four. Consequently, the overhead information of SUB is larger than that of INT and the priority of INT is set to be higher than that of SUB.

Mode	Name	Description
0	INT_4x4	The macroblock is partitioned into sixteen 4x4 blocks. This mode applies SUB_MB_DVD_ZERO.
1	INT_8x8	The macroblock is partitioned into four 8x8 sub-macroblocks.
2	INT_16x16	The macroblock is partitioned into one 16x16 block.
3	SUB_4x4	The macroblock is partitioned into sixteen 4x4 blocks. This mode applies SUB_MB_DVD_ZERO.
4	SUB _8x8	The macroblock is partitioned into four 8x8 sub-macroblocks.
5	SUB _16x16	The macroblock is partitioned into one 16x16 block.
6	DIRECT_PRED	The macroblock is partitioned into sixteen 4x4 blocks and all Intra_DVds are zero.
7	COMBINE	The macroblock is partitioned into four 8x8 sub-macroblocks and each of them is further partitioned into four 4x4 blocks or one 8x8 block. This mode applies INT_SUB.
8	INT_SUB_4x4	The macroblock is partitioned into sixteen 4x4 blocks. This mode applies SUB_MB_DVD_ZERO.

Table 3.1 The modes employed for intra block matching.

INT	Only integer pixel intra block matching is performed.
SUB	Only sub-pixel intra block matching is performed.
INT_SUB	Each sub-macroblock can use INT or SUB adaptively with encoding an extra 1 bit flag for each of them.
SUB_MB_DVD_ZERO	By encoding extra four 1 bit flags to indicate which of the sub-MBs with all Intra_DVds equal to zero. Therefore, the Intra_DVds for that/those sub-macroblock(s) are not encoded. When at least one of the Intra_DVds is non-zero, all the four Intra_DVds of that sub-macroblock are encoded.

Table 3.2 The terminology used in Table 3.1.

Name of the mode	Priority Number	Bit String (I slice)
INT_4x4	6	0 0 0 1 0 0 0
INT_8x8	3	0 0 1 0 1
INT_16x16	2	0 0 1 0 0
SUB_4x4	7	0 0 0 1 0 0 1
SUB _8x8	5	0 0 1 1 1
SUB _16x16	4	0 0 1 1 0
DIRECT_PRED	1	0 1 1
COMBINE	0	0 1 0
INT_SUB_4x4	8	0 0 0 1 0 1 0

Table 3.3 The priority of each mode.

The bit strings of the proposed modes for P or B slice can be derived by the Exp-Golomb method defined in the H.264 standard. The modes are code as ue(v). The input value is the priority number of the mode added by 6 or 24 for P or B slice, respectively.

3.2.3 Performance Enhancement Tools

3.2.3.1 Multiple Best Matches

For motion estimation, generally one integer pixel best match is first found. Then it is used as the center for sub-pixel search. Instead of just finding one integer pixel best match, we propose to find several best matches to improve the coding efficiency of intra block matching. In our experiment, three best matches are found. After finding them, another cost function is used to find the true best match amongst them. Equation 3.13 is the cost function used in our experiment.

$$J\_best = SATD + \lambda_{displacement}\, bits_{Intra\_DVds} \tag{3.13}$$

where J_best is the new cost of the best match,

SATD is the sum of absolute transform differences after transforming the prediction errors with Hadamard Transform,

$\lambda_{displacement}$ is the Lagrange Multiplier, and

$bits_{Intra\_DVds}$ is the number of bits used to encode the Intra_DVds.

The reason of choosing Hadamard Transform is that it is similar to ICT. Therefore, it is more accurate than SAD. In addition, the complexity of Hadamard Transform is less than that of ICT.

As mentioned before, in our experiment, only three best matches are first found. The reason of choosing three is that if too many best matches are found, computational complexity will be increased significantly. In addition, experimentally, three best matches can achieve good result.

This enhancement tool can improve the coding efficiency without increasing the complexity in a great amount. This tool can enhance the performance because the original simple cost function, i.e. Equation 3.1, is not accurate enough. Two or more candidate search points sometimes have the same cost or wrong decision may even be made due to the inaccuracy of Equation 3.1. Therefore, if several best matches are first found and then a more precise cost function is used, better result can be achieved. Certainly, this enhancement tool can also be applied to the motion estimation process. Since we want to test the improvement of all the proposed algorithms and techniques for intra block matching only, this enhancement tool was not applied to motion estimation process.

3.2.3.2 Adaptive Integer and Sub-pixel Intra Block Matching

The motion vector precision of H.264 for luma component is quarter pixel. The accuracy of sub-pixel search is better than that of integer pixel search. However, due to H.264 only allows fixed motion vector precision, the motion vector differences are multiplied by four for the integer pixel positions before they are encoded. Therefore, much overhead information is necessary for encoding them.

In our design, as shown in Table 3.1, each macroblock is allowed to select integer and sub-pixel intra block matching adaptively. The decoder can decode the mode first. Then it knows whether sub-pixel is required. As a result, The Intra_DVds can be encoded efficiently. In addition, when the cost of encoding the vector differences decreases for INT, there is a larger margin for reducing the prediction errors during the block matching process.

To perform sub-pixel intra block matching, the integer pixel best match is first found. Then that best match is selected as search center for the sub-pixel search. Totally 7x7 (including the center point) search points are evaluated with cost function (3.13) and the one with the minimum cost is the best match. The interpolation process for luma and chroma components are the same as that defined in the H.264 standard.

The conversion of the luma Intra_DVs to the chroma Intra_DVs is the same as that of the luma motion vectors to the chroma motion vectors defined in the H.264 standard. Therefore, the precision of chroma Intra_DVs is 1/8.

3.2.4 Pseudo Intra Block Matching

In the H.264 standard, the best match position is first predicted from its neighboring blocks. Then the motion vector difference is computed by subtracting the prediction motion vector from the true motion vector. This is based on the assumption that neighboring blocks belong to the same object and thus their motions are similar. This prediction technique can reduce the bits used to encode the motion information.

For intra block matching, the Intra_DVs are not motion vectors. They are just offsets to indicate the position of the best match. Consequently, the characteristics exploited in motion compensation can not be applied to intra block matching efficiently.

A method to predict a best match position from the previous I slice is proposed to reduce the overhead information. When the co-located macroblock in the nearest previous I slice is coded with the intra block matching technique and its size is larger than or equal to that of the current block, the predicted best Intra_DV (PBIDV) is set to the Intra_DV of that co-located block and the Intra_DVd of the current block is derived with Equation 3.14. Otherwise, when the block size of the current block is larger, the mean of the Intra_DVs of all the involved smaller co-located blocks is computed as the PBIDV. When any of the pixels which are located at the PBIDV is not in the search region, the default predicted position is applied to avoid inaccurate prediction because there is no chance the PBIDV is the best Intra_DV. Certainly, the region outside the search region can also be padded and searched. However, the probability of selecting them as the best match is not very high and thus padding and searching them do not have much gain in coding efficiency. Consequently, the default predicted position is used instead to reduce the complexity. When the co-located macroblock in the nearest previous I slice is not coded with the intra block matching technique, the default predicted position is selected.

Experimentally, three default predicted positions are found. For 4x4 block, Pred_x = 0 and Pred_y = 0, for 8x8 block, Pred_x = -4 and Pred_y = -4 and for 16x16 block, Pred_x = -4 and Pred_y = -12. The values of Pred_x and Pred_y are expressed relative to the top-left corner pixel of the current block. These default predicted positions can achieve acceptable results for different video sequences and different quantization parameters.

$$\text{Intra\_DVd} = \text{Intra\_DV} - \text{PBIDV} \tag{3.14}$$

This technique is called pseudo because the Intra_DVs from the previous I slice are used. Nevertheless, intra prediction demands using decoded samples of the current slice to perform prediction and thus this can still be called intra prediction. To prevent errors in the transmitted Intra_DVds, a one bit flag is designed to be transmitted in the I slices header to indicate whether previous Intra_DVs are used to predict PBIDV. In our experiments, the I slices in the coming nearest I picture use the default predicted positions after every one second. Then the counter resets.

3.3 Proposed Fast Algorithms

In this section, totally four fast algorithms are proposed for variable block size intra block matching. Except fast intra block matching decision, all the proposed fast algorithms reduce the complexity of the whole process without degrading the coding performance.

3.3.1 Fast Intra Block Matching Decision

This fast intra block matching decision method is based on the algorithm proposed in [25]. The performance of this algorithm is video sequence dependent. The mean of the rate distortion costs (RDcosts) of the selected intra block matching modes is updated with Equation 3.15 and 3.16 when one of the modes in Table 3.1 is selected as the best mode. Num and MeanRD are initialized to zero at the beginning of the encoding process.

$$\mathrm{Num}_{k+1} = \mathrm{Num}_k + 1 \tag{3.15}$$

$$\mathrm{MeanRD}_{k+1} = \mathrm{MeanRD}_k + \frac{\mathrm{New\_RDcost} - \mathrm{MeanRD}_k}{\mathrm{Num}_{k+1}} \tag{3.16}$$

where Num is the total number of selected intra block matching macroblocks,

MeanRD is the mean of the RDcosts of the selected intra block matching macroblocks,

New_RDcost is the RDcost of the latest intra block matching coded macroblock, and

k and k+1 indicate the current and updated value, respectively.

Since recent RDcosts are more important for making correct decision, Num is updated with Equation 3.17 after several pictures are encoded. In our experiment, Num is updated after every five pictures are encoded. Offset and Pic_Num in Equation 3.17 were set to 4 and 5, respectively.

$$\mathrm{Num}_{k+1} = \mathrm{Truncate}(\frac{\mathrm{Num}_k + \mathrm{offset}}{\mathrm{Pic\_Num}}) \qquad (3.17)$$

where truncate ( ) means the result are truncated toward zero,

Pic_Num is the number of pictures after which the value of Num is required to be updated, and

offset is a constant for rounding purpose.

After finding the best mode amongst all of the original modes of H.264, the current best RDcost is compared with MeanRD. If MeanRD is smaller, all of the modes in Table 3.1 are evaluated. Otherwise, their checking processes are skipped.

This algorithm is applied only when current slice is a P or B slice. In P or B slice, the probability of selecting intra mode as the best mode is low. Therefore, applying this fast algorithm does not affect the coding performance in a great amount while the complexity can be greatly reduced.

The differences between our algorithm and that proposed in [25] are shown as follows:

1. [25] does not consider that recent RDcosts are more important.
2. Equation 3.16 can be implemented with one division only while [25] requires one multiplication and one division.
3. [25] applies their mode decision algorithm to the intra modes of H.264 while the proposed one applies to intra block matching.

3.3.2 Skipping some Intra Block Matching Processes

This method reuses some evaluated data to skip some processes. This method is applied to COMBINE and INT_SUB_4x4 modes. Some of the intra block matching processes for COMBINE and INT_SUB_4x4 modes can be skipped if INT_4x4, SUB_4x4, INT_8x8 and SUB_8x8 modes are evaluated first. For example, the results of the first sub-macroblock of COMBINE and INT_SUB_4x4 modes can be generated by directly copying the results of the first sub-macroblock of the mentioned four modes. In addition, for instance, if the best sub mode of the first sub-macroblock for COMBINE mode is int_4x4 (The sub-macroblock is partitioned into four 4x4 blocks and all of them use integer intra block matching, int or sub refers to integer or sub-pixel intra block matching.), the block matching process of int_4x4 for the second sub-macroblock can be skipped and the result of the second sub-macroblock of INT_4x4 is the required int_4x4 result for COMBINE mode because the conditions are the same.

It can be noticed that this algorithm reuses the computed data to avoid repetitive computation when certain conditions are fulfilled. Therefore, this algorithm does not degrade the coding performance of the whole matching process. It only reduces the complexity.

3.3.3 Early Termination

This algorithm is designed for COMBINE and INT_SUB_4x4 mode. The main idea of this algorithm is to compare the partial sum of the sub-macroblock RDcosts with the current best RDcost. When the partial sum is larger, the rest of the process is not necessary to be performed.

For COMBINE and INT_SUB_4x4 modes, RD costs are computed for each sub mode (sub-macroblock can be classified into int_4x4, sub_4x4, int_8x8 or sub_8x8.) to find the best. The partial sum of the best sub-macroblock RDcosts of all evaluated sub-macroblocks, RD_costs_sum, is derived with Equation 3.18. Then it is compared with the current best RDcost of the current macroblock. If the RD_costs_sum is larger, the intra block matching process for COMBINE or INT_SUB_4x4 modes will not be carried out. Otherwise, intra block matching is performed continuously for that mode.

$$\text{RD\_costs\_sum} = \lambda \times \text{bits}_{\text{mode}} + \sum_{i=0}^{n} [\lambda \times (\text{bit\_8x8}_i) + \text{Distortion\_8x8}_i] \qquad (3.18)$$

where i is the index of a sub-macroblock,

n is the index of the current sub-macroblock,

λ is the Lagrange Multiplier,

Distortion_8x8_i is the distortion of the sub-macroblock i (luma component),

$bits_{mode}$ is the number of bits used to encode the mode and header information of the current macroblock, and

bit_8x8_i is derived with Equation 3.19.

$$bit\_8x8_i = bit\_coeff_i + bit\_intra\_DVds_i + bit\_other_i \qquad (3.19)$$

where bit_coeff_i is the number of bits used to encode the prediction errors of sub-macroblock i (luma component),

$bit\_intra\_DVds_i$ is the number of bits used to encode the intra_DVds of sub-macroblock i (if necessary),

bit_other_i includes the number of bits used to encode a skip_flag which is for indicating whether all intra_DVds of sub-macroblock i are zero and the number of bits used to indicate the sub mode of sub-MB i. [Note: For block size equals to 4x4 and all intra_DVds equal to 0, no extra flag is required to indicate whether the sub-MB is int_4x4 or sub_4x4 because all intra_DVds are zero.]

In Equations 3.18 and 3.19, $Distortion\_8x8_i$ and bit_coeff_i only comprise those costs of luma component. In H.264, the chroma prediction errors are first transformed with ICT and the DC coefficients are further de-correlated with Hadamard Transform. Therefore, the final distortion of chroma components and the bits used for encoding the chroma prediction errors cannot be found until the whole 8x8 prediction errors matrix is derived (for the case of 4:2:0). This implies that the partial RDcosts contributed by chroma components cannot be found. However, the cost contributed by luma component is normally much higher than that contributed by chroma components and thus the effect of this algorithm is still significant.

It can be noticed that this algorithm does not degrade the coding performance of the whole matching process. It only reduces the complexity. In addition, this algorithm can also be applied to the original intra or inter modes of H.264 to reduce the complexity. Since we focus on the intra block matching process, in our experiments, this algorithm was only applied to intra block matching process.

3.3.4 SAD Reuse Techniques

[4], [26], and Fast Full Search are designed for variable block size motion estimation. In Chapter 2, Fast Full Search has already been reviewed while [26] will be introduced in Chapter 4. SADs are reused by partitioning a macroblock into sixteen 4x4 blocks. Then the SADs of each of them are computed and stored. The SAD of larger block sizes can be derived by summing the required SADs of the 4x4 blocks.

This SAD Reuse Technique can also be applied to intra block matching to reduce the complexity. The current macroblock is partitioned into sixteen 4x4 blocks. Since the padding method is different for different block sizes, when a candidate search point consists of neither pixels outside the search region nor the padded pixels, the 4x4 SAD of the candidate search point is computed, stored and reused. By applying this technique, the computational complexity of intra block matching can be reduced significantly. Same as the algorithms introduced in Section 3.3.2 and 3.3.3, this method does not degrade the coding performance.

3.4 Experimental Results

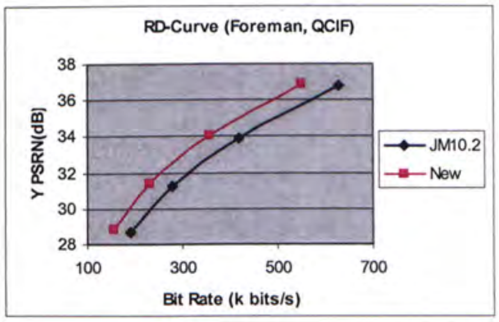
The results were generated with JM10.2 after integrating the intra block matching algorithm with the proposed techniques, modes and fast algorithms. In our experiments, the search region is shown in Figure 3.1, i.e. all the pixels that are reconstructed or padded, and within macroblocks "P" and "C". RDO was turned on. One reference frame and Hadamard Transform were used. No 8x8 transform was involved, CAVLC and Exp-Golomb were applied for entropy coding and QPs were 28, 32, 36 and 40. Table 3.4 shows the improved coding performance in terms of ΔPSNR (dB) or the equivalent ΔBitrate (%) for different video sequences with QCIF or CIF 4:2:0 format when compared with those of the original H.264 encoder (The results do not compare with those of [23] and [24] because the conditions were totally different. Interested readers can use their results for reference). The extra encoding time (EET) is computed with Equation 3.20. ΔPSNR_1 and ΔBitrate_1 show the results when GOP (Group of Pictures) structure was IIIIII... with average EET = 242.378% and ΔPSNR_2 and ΔBitrate_2 show the results when GOP structure was IBBPBBI... with average EET = 26.877%. The computation method of the ΔPSNRs and the equivalent ΔBitrates is proposed in [11]. Figure 3.4 and 3.5 show the Rate Distortion Curves for different video sequences with IIIIII... and IBBPBBI...GOP structures, respectively.

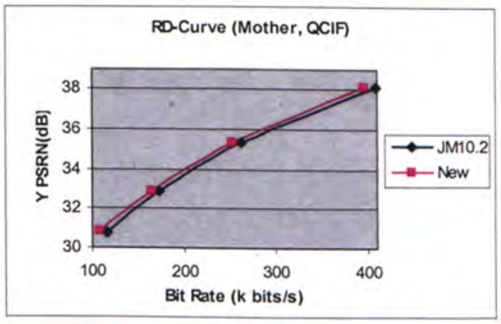
The results show that the coding performance of the encoder can be improved significantly. However, EET is very high for intra coding and is still quite high for hybrid coding. The reason is that the proposed modes are evaluated and compared one by one and thus the complexity is high. If efficient mode decision algorithms, which are similar to inter mode decision algorithms, are developed, the problem can be solved.

$$\text{EET}(\%)=\frac{\text{New\_Encoding\_Time}-\text{Original\_Encoding\_Time}}{\text{Original\_Encoding\_Time}}\times 100 \tag{3.20}$$

Sequence	ΔPSNR_1	ΔBitrate_1	ΔPSNR_2	ΔBitrate_2
Akiyo, QCIF	0.61	-7.66	0.38	-5.12
Carphone, QCIF	0.94	-12.14	0.42	-7.07
Claire, QCIF	0.76	-9.36	0.39	-5.35
Foreman, QCIF	1.26	-17.53	0.61	-10.51
Mother, QCIF	0.28	-4.80	0.14	-2.70
News, QCIF	0.60	-6.95	0.29	-3.86
Table, QCIF	0.78	-12.08	0.31	-6.02
Akiyo, CIF	0.69	-9.36	0.37	-5.59
Bus, CIF	0.34	-4.93	0.12	-2.32
Carphone, CIF	1.00	-15.18	0.40	-8.08
Foreman, CIF	0.75	-12.68	0.32	-6.42
Hall, CIF	0.41	-5.61	0.19	-3.57
Highway, CIF	0.80	-16.50	0.35	-9.79
Mobile, CIF	0.43	-4.67	0.11	-2.11
Mother, CIF	0.30	-5.33	0.14	-2.92
News, CIF	0.75	-9.41	0.34	-4.98
Paris, CIF	0.58	-7.13	0.25	-3.83
Table, CIF	0.52	-10.67	0.21	-5.65
Average	0.656	-9.556	0.297	-5.327

Table 3.4 The ΔPSNRs and ΔBitrates of different video sequences.





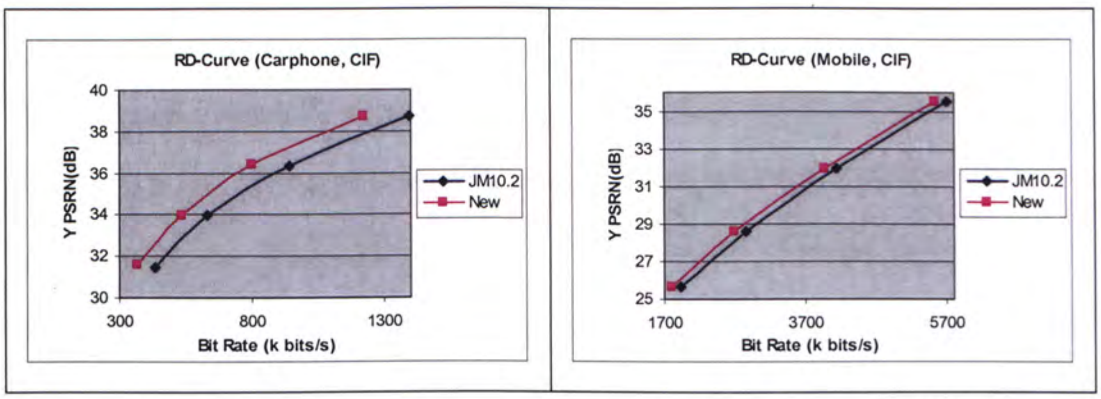


Figure 3.4 RD-curve comparisons of different video sequences with IIIIII... GOP structure.

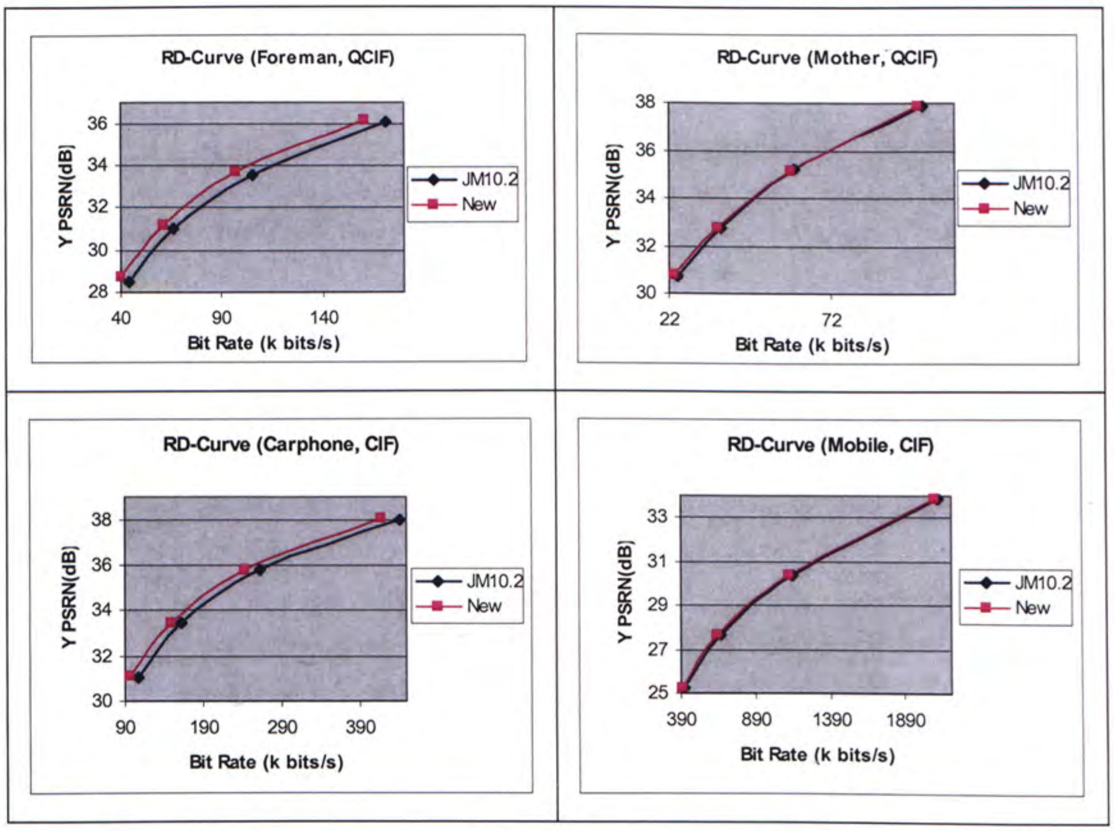
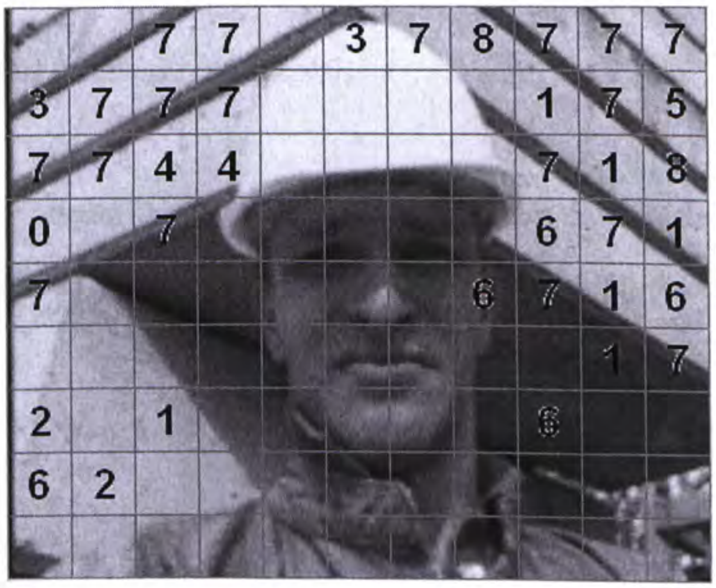


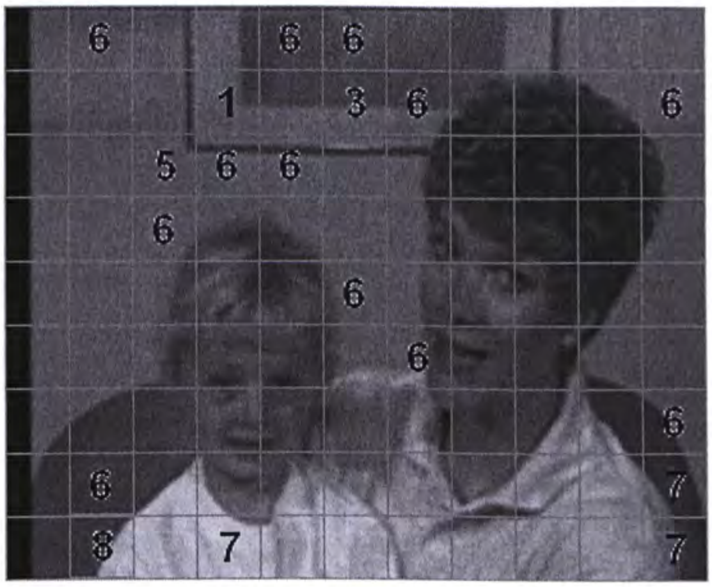
Figure 3.5 RD-curve comparisons of different video sequences with IBBPBBI... GOP structure.

Figure 3.6 are two examples to illustrate the selected intra block matching best modes of the sequences *Foreman* and *Mother* with QCIF 4:2:0 format. In the figure, the numbers indicate the selected best modes which are defined in Table 3.1. It is obvious that most of the macroblocks representing the background building of *Foreman* select intra block matching modes. The reason is that the directions of the grey lines in the picture are not similar to those of the intra prediction technique adopted in H.264. The directions of 4x4 intra prediction in H.264 include vertical, horizontal, 45^o, 26.6^o and DC. The grey lines in the picture are not in these directions and thus intra block matching modes are selected. For the human in *Foreman*, it is difficult to find a similar pattern from the neighbouring blocks. Accordingly, the intra block matching modes are not selected. For *Mother*, similar patterns are either difficult to be found or very smooth. Accordingly, intra block matching modes are not selected frequently. Even though some macroblocks are encoded with intra block matching techniques, due to the mentioned features of this sequence, the improvement is still not significant.

From these two examples, it can be observed that when intra block matching modes are selected frequently, the coding performance can be improved substantially and vice versa.



(a) *Foreman*



(b) *Mother*

Figure 3.6 The selected intra block matching best modes of sequences (a) *Foreman* and (b) *Mother*.

As mentioned before, the computation method of the ΔPSNRs and the equivalent ΔBitrates is proposed in [11]. This method mainly includes three basic elements.

1. Fitting a curve through four data points.
2. An expression is found for the integral of the curve.
3. The average difference is computed as the difference between the integrals divided by the integration interval.

The curve is fitted with a third order polynomial which is expressed by Equation 3.21. With this method, two curves can be fitted. One is the result of the reference software and the other is the result of the improved encoder. The bitrate can be found as a function of PSNR in a similar way. This function is defined by Equation 3.22.

$$\text{PSNR} = a + b*\text{bit} + c*\text{bit}^2 + d*\text{bit}^3 \qquad (3.21)$$

$$\text{bit} = a + b*\text{PSNR} + c*\text{PSNR}^2 + d*\text{PSNR}^3 \qquad (3.22)$$

Where PSNR is the PSNR at a particular bitrate which is the bit in the equations,

a, b, c and d are determined such that the curve passes through all the four data points.

In this way, the followings can be found.

1. Average PSNR difference in dB over the whole range of bitrates
2. Average bitrate difference in % over the whole range of PSNR

Therefore, the results can be compared [11].

In this Chapter, many enhancement algorithms, modes and fast algorithms for intra block matching are proposed. With them, the coding performance of H.264 can be improved significantly. The pseudo intra block matching technique predicts the location of the best match accurately. Therefore, the overhead information for encoding the Intra_DVds is greatly reduced. This technique contributes to the improvement most.

The maximum gain in PSNR can be more than 1 dB for intra coding and more than 0.6 dB when hybrid coding is applied when compared with the coding performance of the original H.264 encoder.

Chapter 4 Enhanced SAD Reuse Fast Motion Estimation

In this chapter, a fast variable block size motion estimation algorithm is introduced. It focuses on reducing the complexity by utilizing the characteristics of variable block size motion estimation. With the proposed algorithm, the encoding speed can be improved significantly while the coding performance can be maintained. The complexity and coding performance comparisons of the proposed algorithm and different fast motion estimation algorithms which are adopted in the reference software of H.264 are shown at the end of this chapter.

4.1 Introduction

Motion estimation and compensation is a video coding technique that exploits temporal redundancy between pictures in a video to compress the video efficiently. Many video coding standards such as MPEG-1, 2, 4 and H.261, H.263 use this technique with fixed block size. H.264 also applies it to encode videos with some modification. Instead of fixed block size, H.264 uses variable block size to improve the coding efficiency. However, variable block size motion estimation is computational intensive and thus fast motion estimation algorithms are required to reduce its complexity. In H.264, seven different block types (16x16, 8x16, 16x8, 8x8, 4x8, 8x4 and 4x4) are adopted. Each block type requires performing motion estimation and therefore the total time required for variable block size motion estimation is very long.

For finding a best match during the motion estimation process, a cost function is necessary. In the reference software of H.264, the sum of absolute difference (SAD) is a part of the cost function and it is the most complex part. Consequently, reducing SAD computation can significantly reduce the total time required for motion estimation.

In the literature, some fast motion estimation algorithms such as those proposed in [1-3] and [5-9] concentrate on reducing the number of search points without reusing SAD. Fast Full Search (FFS), the fast motion estimation algorithm adopted in the reference software of H.264 for reducing the complexity of Full Search while maintaining the coding efficiency, uses a fixed search center for the motion estimation of the seven block types within the same macroblock. The SADs of each of the sixteen 4x4 blocks are pre-computed and stored. Therefore, these stored SADs can be reused during the motion estimation processes of all the seven block types. This can significantly reduce the complexity of the variable block size motion estimation process and thus FFS is less complex than Full Search. Even though FFS considers SAD reuse, it is not fast enough because it still performs Full Search once for each macroblock. The algorithm proposed in [4] also considers SAD reuse without pattern-based motion estimation and this may not be very accurate. We propose to reuse SAD with pattern-based motion estimation and refinement search to reduce the motion estimation time while maintaining good coding performance in terms of PSNR and bitrate.

4.2 Proposed Fast Motion Estimation Algorithm

4.2.1 Best Initial Motion Vector

At the beginning, the availabilities of the neighboring blocks A and B of the current block are derived. Blocks A and B are the left and upper block of the current block, respectively, as shown in Figure 4.1. When the neighboring block A or B exceeds the boundary of the current slice or is intra-coded, that neighboring block is unavailable. Otherwise, it is available.

16x16 block type is evaluated first, followed by 8x16, 16x8, 8x8, 4x8, 8x4 and 4x4. The following process is performed only when block type is 16x16. Other block types do not require to perform it. At most three initial candidate search points can be derived. When blocks A and/or B are available, their motion vectors, which are used to derive initial candidate search points, are available and are used to derive the first and/or the second initial candidate search points. The prediction motion vector defined in the H.264 standard, i.e. the median of the motion vectors of the left block, upper block, upper-left or upper-right block, is used to derive the third initial candidate search point.

Finally, the costs of each of these initial candidate search points are computed. The one with the minimum cost is the best initial candidate search point and the corresponding motion vector is the best initial motion vector (When the motion vector used for deriving an initial candidate search point is not available, the cost of the corresponding initial candidate search point is regarded as larger than those of the others.). The cost function is defined by Equation 4.1.

$$\text{Cost} = \text{SAD} + \lambda(\text{QP}) \times \text{bit}_{\text{MVd}} \tag{4.1}$$

where SAD is the sum of absolute difference between the current block and a block in the reference frame,

$\lambda(\text{QP})$ is the Lagrange multiplier whose value depends on QP, the quantization parameter, and

bit_{MVd} is the number of bits used to encode the motion vector difference (motion vector - prediction motion vector).

The best initial motion vector is used to derive the center of a fixed initial search pattern. The initial search pattern changes according to the search range and this will be discussed in Section 4.2.2. The best initial motion vector derived in the above process is used for all other block types, i.e. 8x16, 16x8, 8x8 4x8, 8x4 and 4x4, within the same macroblock.

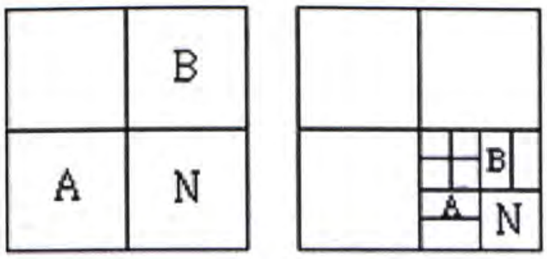


Figure 4.1 Neighboring blocks A and B of the current block N.

4.2.2 Initial Search Pattern

The best initial motion vector is used to derive the center of a fixed initial search pattern. The fixed initial search pattern consists of two parts called the center part and the outside part. The candidate search points in the outside part are for handling large motion situations while those in the center part are for handling small motion situations. The center part is fixed and does not depend on search ranges. The outside part changes according to the search range, as illustrated in Figure 4.2. Figure 4.2 (a) illustrates the center part, "-1" indicates the best initial candidate search point derived in Section 4.2.1. "-1", "0" and "1" are the indices of the candidate search points. Candidate search points with index equals to 1 are at the boundary and the others are not at the boundary of the center part. Figure 4.2 (b) illustrates the center part and outside part.

Equations 4.2 and 4.3 are used to derive the number of layers of the outside part and the distance between each layer of the outside part and the center part. S is the number of layers and L (n) is the distance between layer n and the center part. Table 4.1 shows some examples to illustrate the relationship among the search range, S and L (n).

$$S = \text{Truncate}(\log_2(\frac{\text{Search\_Range}}{4})) \tag{4.2}$$

$$L(n) = \text{Truncate}(\frac{\text{Search\_Range}}{2^n}), \quad 1 \le n \le S \tag{4.3}$$

If $S > 0$, both the center part and the outside part are required and Equation 4.3 is evaluated to determine the positions of the candidate search points in the outside part. If $S = 0$, only the center part is required. n is an integer which is greater than or equals to 1 and smaller than or equals to S. Truncate ( ) truncates the result toward zero, for example, Truncate (1.8) = 1.

Search range	<8	8	16	32	64
S	0	1	2	3	4
L(1)		4	8	16	32
L(2)			4	8	16
L(3)				4	8
L(4)					4

Table 4.1 Examples for illustrating the relationship among the search range, S and L(n).

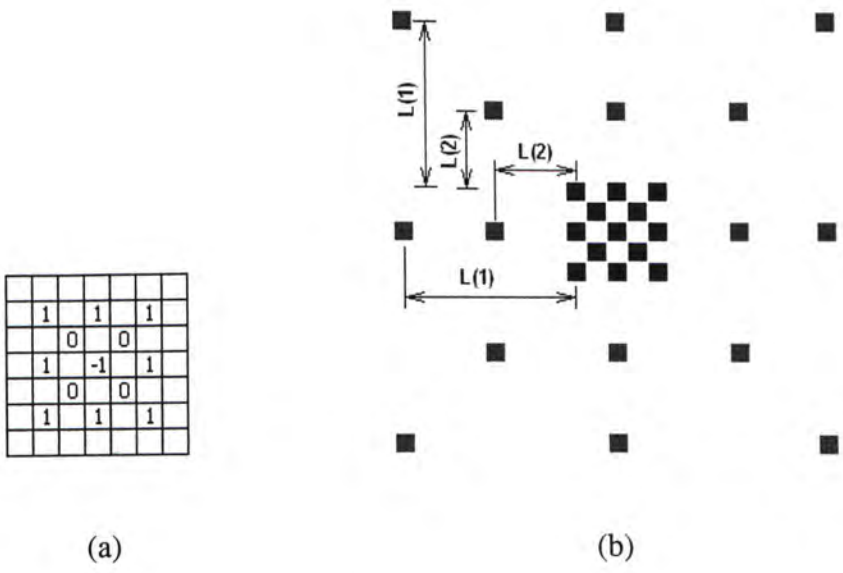


Figure 4.2 Initial search pattern, (a) only the center part, (b) the center part and the outside part for S = 2.

The candidate search points in the initial search pattern required to be evaluated for all the block types. First, the current macroblock is partitioned into sixteen 4x4 blocks and the SADs, called SAD_{4x4}s, of each of them are pre-computed with the candidate search points in the initial search pattern. Then each SAD_{4x4} is stored and can be reused for computing the true SADs of all block types within the same macroblock. Since all the block types use the same best initial motion vector, SAD_{4x4}s can be reused. This can reduce the complexity of the variable block size motion estimation process significantly.

4.2.3 Initial Search Process and Search Pattern Improvement Process

The costs of all candidate search points in the initial search pattern are computed with Equation 4.1 (SAD can be computed by summing the corresponding SAD_{4x4}s). The one with the minimum cost is defined as the best initial search pattern candidate search point (BISPCSP).

When block type is 16x16, go to Section 4.2.3.1.

For the other block types, the cost of another candidate search point, called improved candidate search point (ICSP), which is derived from the prediction motion vector defined in the H.264 standard is evaluated.

When block type is 8x16 or 16x8 and the cost of ICSP is smaller than that of BISPCSP, ICSP is defined as motion estimation center (MEC) and motion estimation process defined in Section 4.2.4.1 is carried out. Otherwise, proceed to Section 4.2.3.1.

For 8x8 block type, when the cost of ICSP is smaller than that of BISPCSP, ICSP is defined as the center of an improved search pattern (ISP) and ISP of that sub-macroblock is available. It is fixed for all search ranges as illustrated in Figure

4.3. All of its candidate search points are evaluated when it is available. Since a sub-macroblock is 8x8, it can be partitioned into four 4x4 blocks and their SAD_{4x4}s are computed and stored. Therefore, the SAD_{4x4}s can be reused to evaluate the candidate search points in ISP for all sub-macroblock types of that sub-macroblock. If the best candidate search point is in ISP, go to Section 4.2.3.2. Otherwise, go to Section 4.2.3.1.

For 4x8, 8x4 or 4x4 block type, when the ISP of the corresponding sub-macroblock is available, the costs of its candidate search points (SAD_{4x4}s can be reused) are compared with those of BISPCSP and ICSP. If the best candidate search point is BISPCSP, go to Section 4.2.3.1. If ICSP is the best, proceed to Section 4.2.4.1 with ICSP as MEC. Otherwise, the best candidate search point must be one of the candidate search points in ISP. Accordingly, proceed to Section 4.2.3.2

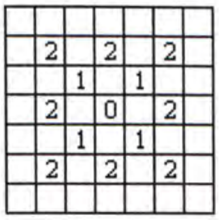


Figure 4.3 Improved search pattern (ISP), "0" is the center of ISP, "0" or "1" is non-boundary candidate search point and "2" is boundary candidate search point of ISP.

4.2.3.1 BISPCSP Motion Estimation or Refinement Process Decision

When BISPCSP is in the outside part or is at the boundary of the center part of the initial search pattern, proceed to Section 4.2.4.1 with BISPCSP as MEC. Otherwise, proceed to Section 4.2.4.2 with BISPCSP as refinement center.

4.2.3.2 ISP Motion Estimation or Refinement Process Decision

When the best candidate search point is not at the boundary of ISP, proceed to Section 4.2.4.2 with it as the refinement center. Otherwise, proceed to Section 4.2.4.1 with it as the MEC.

4.2.4 Motion Estimation Process and Refinement Process

The patterns used in the motion estimation process and the refinement process are illustrated in Figure 4.4. The patterns used for these two processes are changed according to different conditions. The decision process of using which of them is discussed in Section 4.2.4.1 and 4.2.4.2.

4.2.4.1 Motion Estimation Process

The center of the search window is defined as ICSP, the center of the initial search pattern, or the center of ISP, depending on where the MEC comes from. When the MEC is ICSP, the center of the search window is ICSP. When the MEC is in the initial search pattern or is in ISP, the center of the search window is set to the center of the initial search pattern or ISP, respectively.

When the MEC is in the outside part of the initial search pattern or is ICSP, vertical hexagon search pattern is selected. Otherwise, star search pattern is selected. When the MEC is at the boundary of the center part of the initial search pattern or at the boundary of ISP, the block is assumed to have small motion and therefore star pattern is selected to minimize the number of search points. If horizontal hexagonal search pattern is used instead of vertical hexagonal pattern, through experiments, the overall performance is degraded a little. This can be explained with the argument that [2] vertical motion is less likely than horizontal motion for natural video. Consequently, vertical hexagonal search pattern is used.

After the search pattern is selected, the motion estimation process is performed as follows:

Step 1) Six or four (depending on vertical hexagon or star search pattern is selected) candidate search points centered at the MEC are evaluated with the selected search pattern. If the best one is at the center, proceed to step 3; otherwise, proceed to step 2.

Step 2) With the best candidate search point in the previous search step as the center, three more candidate search points are evaluated with the same search pattern. If the best one is at the center, proceed to step 3; otherwise, repeat this step.

Step 3) Perform the refinement process in Section 4.2.4.2 with the best candidate search point in the previous search step as refinement center.

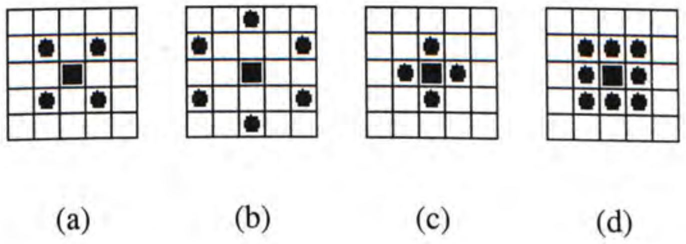


Figure 4.4 (a) Star search pattern and (b) Vertical hexagon search pattern, square is the center and circles are the candidate search points which are evaluated in

each search step. (c) Small diamond refinement pattern and (d) Hexagon refinement pattern, square is the refinement center and circles are the candidate search points which will be evaluated.

4.2.4.2 Refinement Process

If the star search pattern is chosen as the motion estimation search pattern or the motion estimation process is not required, small diamond refinement pattern is selected as the refinement pattern. Otherwise, hexagon refinement pattern is chosen.

Eight or four candidate search points centered at the refinement center are evaluated with the selected refinement pattern. The best one is defined as the best match.

The above processes are summarized in Figure 4.5.

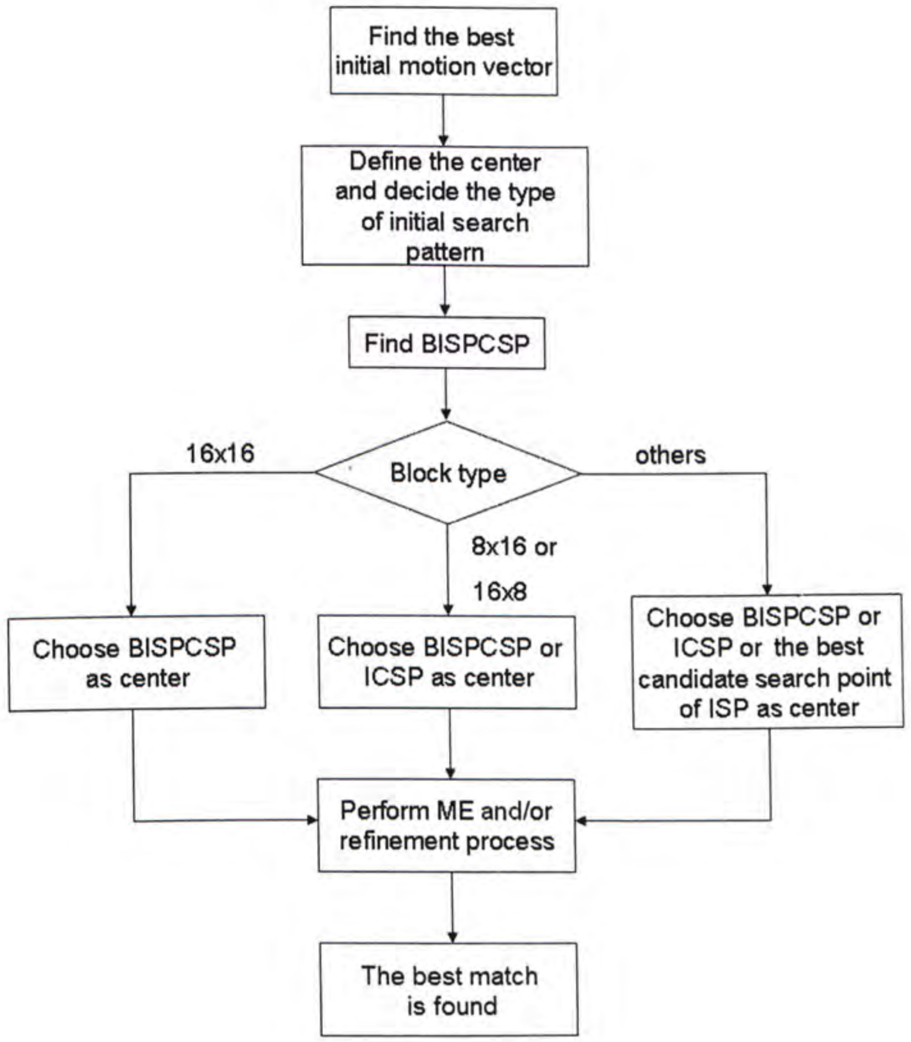

Find the best initial motion vector
Define the center and decide the type of initial search pattern
Find BISPCSP
Block type
16x16
8x16 or 16x8
others
Choose BISPCSP as center
Choose BISPCSP or ICSP as center
Choose BISPCSP or ICSP or the best candidate search point of ISP as center
Perform ME and/or refinement process
The best match is found

Figure 4.5 Flow chart of the proposed SAD reuse motion estimation process.

4.2.5 Motion Estimation Skip Process for B Pictures

If the cost of a particular block type of list 1 is smaller than that of list 0 (list 0 and list 1 are the lists storing the reference pictures for motion estimation and compensation), the mean of the costs of that block type of list 1 will be updated. After the cost of one block type of list 0 is evaluated, it is compared with the weighted mean of the cost of the same block type of list 1. Weighted mean is the mean multiplied by a weighting factor, in our experiments, which was set to 0.75. The weighting factor is adjusted according to whether high speed or good coding performance is demanded. If the cost of certain block type of list 0 is smaller than the weighted mean of the same block type of list 1, the motion estimation process of list 1 for that particular block type will be skipped and the best MV of list 1 for that block type will be set to the prediction MV which is defined in the H.264 standard. To achieve better coding performance, this comparison process will be performed only if the number of samples used to compute the corresponding mean is larger than or equals to certain threshold, in our experiments, which was set to ten. Since SAD_{4x4} requires to be computed and stored when block type is 16x16, this motion estimation skip process is not performed when block type is 16x16.

4.3 Experimental Results

The experiments were carried out using the H.264 reference software JM 10.2 [10] on PC with 3.2G Hz CPU and 1G RAM. Two tests were carried out. The first test used only one I frame and all of the remaining frames were P frames and 150 frames were tested for each video sequence. The second test used only one I frame, there were two B frames between two consecutive I or P frames and 148 frames were tested for each video sequence. Slice QPs of I, P and B frames were 24, 28, 32 and 36. Only integer motion estimation was applied and the number of reference frames was one. RDO was turned off and all seven block types were evaluated with search range equaled to 16. The ME Time is computed by JM 10.2. All comparisons are with respect to FFS. The proposed algorithm is compared with four fast motion estimation algorithms adopted in JM 10.2, namely Hybrid Unsymmetrical-cross Multi-Hexagon-grid Search (UMHS), simplified UMHS (SUMHS), Enhanced Predictive Zonal Search (EPZS) and FFS. The results showing the coding performances of the motion estimation algorithms in terms of PSNR variation or bitrate variation are in Table 4.2 and Table 4.3, respectively. The PSNR variation and the bitrate variation in the tables are computed with the method proposed in [11].

The measurement of the complexity used in the comparisons is derived Equation 4.4:

$$\text{ME Time}\,(\%) = 100 \times \left(\frac{\text{ME Time}_{\text{Fast\_ME}}}{\text{ME Time}_{\text{FFS}}}\right) \tag{4.4}$$

where Fast_ME is UMHS, SUMHS, EPZS or New which is the proposed algorithm, and

ME Time is the motion estimation time computed by JM 10.2.

From Table 4.2, 4.3 and 4.4, it can be observed that the motion estimation time of the proposed algorithm is the shortest and the coding performance in terms of bitrate variation or PSNR variation of the proposed motion estimation algorithm is similar to those of the other motion estimation algorithms.

Figure 4.6 and Figure 4.7 show the RD curve comparisons of the motion estimation algorithms, it can be observed the RD-curves of the proposed algorithm are similar to those of FFS.

	Bitrate Variation (%) w.r.t. FFS				PSNR Variation (dB) w.r.t. FFS			
Sequence	UMHS	SUMHS	EPZS	New	UMHS	SUMHS	EPZS	New

Carphone, C	3.84	8.11	2.57	4.61	-0.14	-0.29	-0.09	-0.17
Coastguard, C	0.19	0.01	-0.24	0.20	0.00	0.00	0.01	-0.01
Container, C	0.77	0.92	0.29	0.35	-0.03	-0.04	-0.01	-0.01
Foreman, C	2.52	6.59	1.85	3.62	-0.09	-0.24	-0.07	-0.13
Carphone, Q	0.96	3.40	1.29	1.93	-0.04	-0.14	-0.05	-0.08
Child, Q	0.60	1.43	0.19	0.76	-0.04	-0.09	-0.01	-0.05
Claire, Q	0.98	3.82	0.91	0.67	-0.04	-0.17	-0.04	-0.03
Coastguard, Q	0.12	0.01	0.03	0.08	0.00	0.00	0.00	0.00
Foreman, Q	3.38	7.63	1.47	3.82	-0.15	-0.33	-0.07	-0.17
Suzie, Q	0.22	2.19	0.22	0.85	-0.01	-0.08	-0.01	-0.03
Table, Q	1.81	3.19	1.09	2.78	-0.09	-0.15	-0.05	-0.13
Trevor, Q	0.53	1.44	0.85	0.82	-0.02	-0.06	-0.04	-0.04
Average	**1.33**	**3.23**	**0.88**	**1.71**	**-0.055**	**-0.133**	**-0.037**	**-0.072**

Table 4.2 Bitrate and PSNR comparisons, C is CIF, Q is QCIF for IPPPPPPPPP GOP structure….

	Bitrate Variation (%) w.r.t. FFS				PSNR Variation (dB) w.r.t. FFS			
Sequence	UMHS	SUMHS	EPZS	New	UMHS	SUMHS	EPZS	New
Carphone, C	2.53	6.03	1.79	3.25	-0.10	-0.23	-0.07	-0.13
Coastguard, C	-0.39	-1.54	-1.17	-0.58	0.01	0.06	0.05	0.02

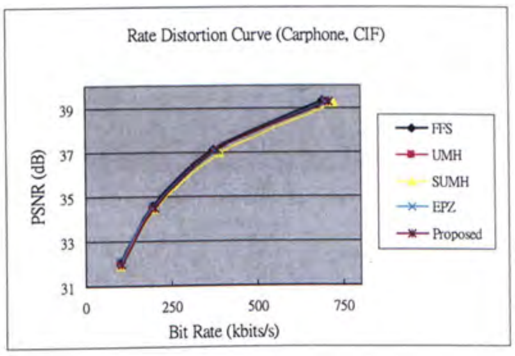
Container, C	1.03	0.79	0.30	-0.53	-0.04	-0.03	-0.01	0.02
Foreman, C	1.22	4.33	0.99	3.37	-0.05	-0.17	-0.04	-0.13
Carphone, Q	0.87	2.76	1.12	0.52	-0.04	-0.12	-0.05	-0.02
Child, Q	0.01	0.90	-0.22	-0.28	0.00	-0.06	0.01	0.02
Claire, Q	1.81	4.74	1.40	0.38	-0.08	-0.20	-0.06	-0.02
Coastguard, Q	0.08	-0.70	-0.89	-0.65	0.00	0.03	0.04	0.03
Foreman, Q	0.97	3.65	1.17	1.84	-0.05	-0.17	-0.05	-0.09
Suzie, Q	-0.33	0.05	-0.99	-0.26	0.01	0.00	0.04	0.01
Table, Q	1.65	1.90	1.33	3.05	-0.08	-0.09	-0.06	-0.15
Trevor, Q	0.12	0.13	-0.46	-0.73	-0.01	-0.01	0.02	0.04
Average	**0.80**	**1.92**	**0.36**	**0.78**	**-0.034**	**-0.082**	**-0.016**	**-0.033**

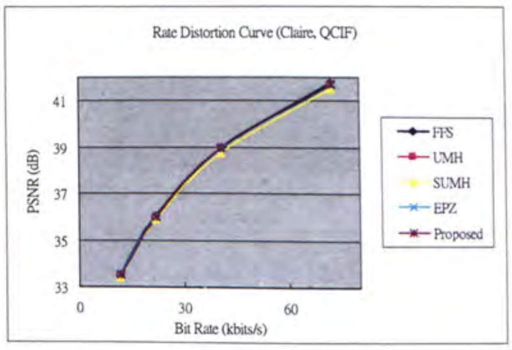
Table 4.3 Bitrate and PSNR comparisons, C is CIF, Q is QCIF for IBBPBBPBBP GOP structure....

	ME Time (%) w.r.t. FFS				ME Time (%) w.r.t. FFS			
	IPPPPPPPPP...				IBBPBBPBBP...			
Sequence	UMHS	SUMHS	EPZS	New	UMHS	SUMHS	EPZS	New
Carphone, C	15.09	10.46	16.02	9.65	15.30	10.12	15.53	7.54
Coastguard, C	25.01	17.14	22.23	10.85	26.66	17.68	23.15	9.54
Container, C	13.37	10.00	14.51	8.67	12.92	9.34	14.05	6.91

Foreman, C	17.89	11.75	22.25	9.59	18.86	12.15	18.05	8.28
Carphone, Q	15.52	10.94	16.13	9.36	16.61	9.78	15.87	7.57
Child, Q	13.31	9.71	14.51	8.32	13.80	9.47	13.85	6.89
Claire, Q	10.11	8.12	11.63	7.66	10.46	7.25	11.31	6.32
Coastguard, Q	23.16	15.74	19.45	9.25	22.67	15.81	21.35	9.62
Foreman, Q	19.61	13.88	18.60	10.17	19.43	12.88	18.61	8.34
Suzie, Q	16.25	11.28	15.40	8.69	15.84	11.39	15.79	8.45
Table, Q	18.58	13.88	17.41	8.9	19.16	13.40	18.13	8.69
Trevor, Q	16.21	11.47	15.49	8.55	16.74	11.03	15.20	6.87
Average	**17.01**	**12.03**	**16.97**	**9.14**	**17.37**	**11.69**	**16.74**	**7.92**

Table 4.4 Motion estimation time comparisons, C is CIF, Q is QCIF for IPPPPPPPPP… and IBBPBBPBBP... GOP structures.





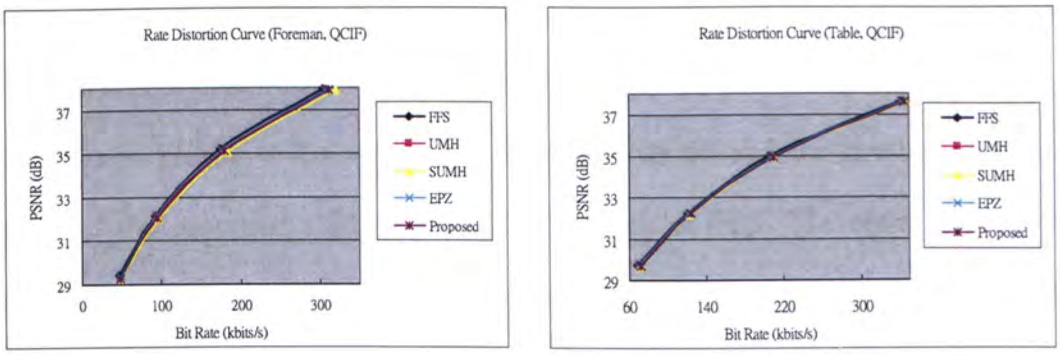


Figure 4.6 RD-curve comparisons of different video sequences with different motion estimation algorithms for IPPPPPPPPP... GOP structure.

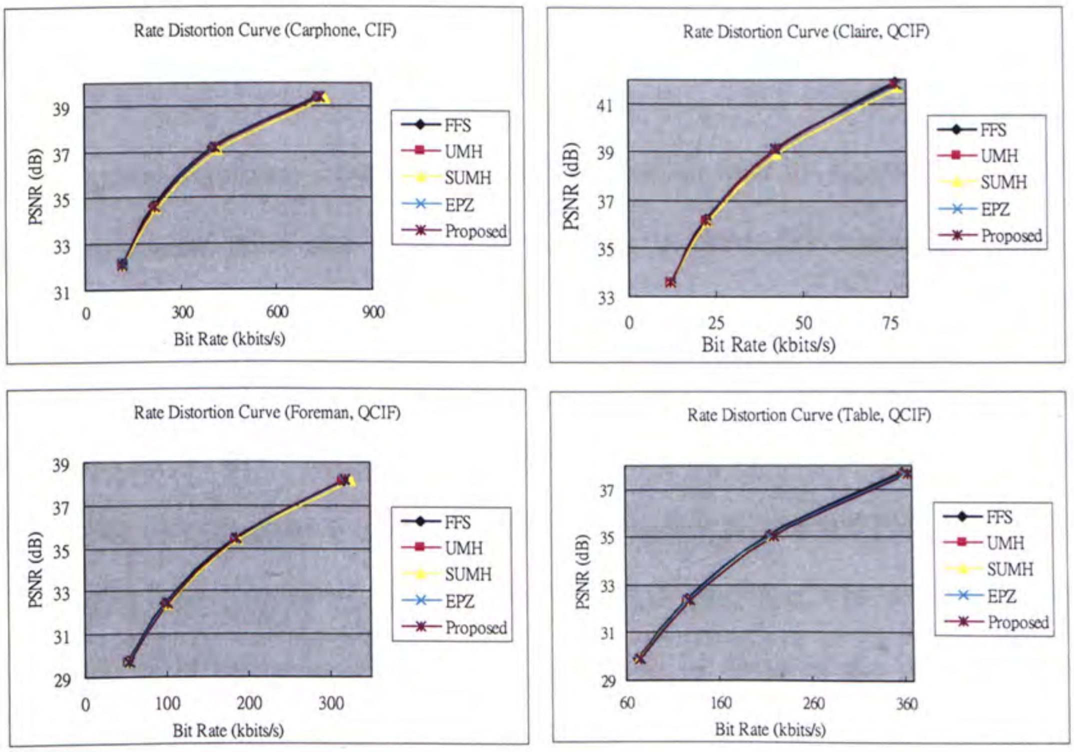


Figure 4.7 RD-curve comparisons of different video sequences with different motion estimation algorithms for IBBPBBPBBP... GOP structure.

From the experimental results, it can be observed that for videos with large motion, such as *Carphone*, *Foreman* and *Table*, the coding performance in terms of bitrate variation or PSNR variation of the proposed algorithm is not as good as that of FFS.

For the proposed algorithm, the center of the initial search pattern is fixed for all the blocks within a macroblock, but the true prediction motion vectors for these blocks may change greatly if the video contains large motion and the number of search points of the proposed algorithm is less than that of FFS. Therefore, the coding performance of the proposed algorithm is not as good as that of FFS for videos with large motion. For videos with small motion, such as *Claire*, *Coastguard*, *Container*, etc, the coding performance of the proposed algorithm is similar to that of FFS. From the experimental results, it can also be observed that the motion estimation time of the proposed algorithm is the shortest amongst the five fast motion estimation algorithms.

The proposed algorithm reduces the motion estimation time by approximately 90% when compared with that of FFS with only a little degradation of coding performance in terms of PSNR and bitrate. The worst bitrate and PSNR variations of the proposed algorithm are +4.61% and -0.17dB, the best bitrate and PSNR variations are +0.08% and 0dB for IPPPPPPPPP... GOP structure. The worst bitrate and PSNR variations are +3.37% and -0.15dB, the best bitrate and PSNR variations are -0.73% and 0.04dB for IBBPBBPBBP...GOP structure. The average coding performance of the proposed algorithm is similar to those of the others and the motion estimation time of the proposed algorithm is the shortest amongst the five fast motion estimation algorithms. Therefore, it can be concluded that the proposed algorithm reduces the complexity of variable block size motion estimation.

Chapter 5 Development of Real-Time H.264 Codec on Pocket PC

H.264 achieves excellent coding performance and outperforms previous video coding standards. However, its complexity is very high, especially on encoder side, and the processing power of mobile device is limited. Accordingly, optimizations are required for real-time applications, especially on mobile platform. This chapter focuses on the development of a real-time H.264 codec on Pocket PC platform. Based on the proposed algorithms and optimization techniques, a Pocket PC software-based real-time H.264 codec is developed. Since the whole system is implemented in software, it is suitable for technology transfer.

H.264 provides several profiles which include Baseline, Main, Extended and High Profiles for different applications. The Baseline Profile was chosen for developing the system. The main reason is that the complexity of the techniques in Baseline Profile is low when compared with those of the others and the performance is acceptable for mobile applications. Accordingly, this profile is the most suitable one for developing real-time system on mobile platform.

There are many kinds of optimization techniques such as instruction level optimization, algorithmic optimization, code level optimization, etc. Generally, instruction level optimization applies SIMD (Single Instruction Multiple Data) technology to reduce the complexity. Normally several data are loaded into a single register, SIMD technology provides instructions to process these packed data in parallel. For example, when four data are loaded into a single register, one SIMD instruction can process four data at the same time. Therefore, the complexity can be reduced. The instruction level optimization can be applied to some highly complex modules such as forward and inverse Integer Cosine Transform, SAD computation, interpolation, etc. After the instruction level optimization is performed, the complexity of these time consuming modules can be reduced significantly.

In the following sections, we will focus on discussing about the algorithmic optimizations and code level optimizations followed by experimental results. The applications of the developed real-time codec will be discussed at the end of this chapter.

5.1 Algorithmic Optimizations

Algorithmic optimizations reduce the complexity of the system through algorithms with or without degradation of coding performance. The algorithms which are proposed and applied to optimize the codec include fast interpolation, fast integer and sub-pixel motion estimation, early skip termination, etc. The details of the fast integer motion estimation, early skip termination can be found in [27, 28]. This section focuses on the proposed fast interpolation and sub-pixel motion estimation algorithms and an inverse Integer Cosine Transform and inverse quantization skipping process.

5.1.1 Fast Sub-Pixel Motion Estimation

Figure 5.1 illustrates the sub-pixel motion estimation used for Fast Full Search in the reference software of H.264. In the first step, the center square and the triangles are evaluated. Then, according to the best search point in the previous step, the surrounding eight quarter pixels are evaluated. The best one (the one with the minimum cost which is derived with Equation 4.1) amongst the nine search points is the best match. In [2], the assumption, which states that the unimodal error surface in most cases for fractional pixel search holds true [12], is utilized. In our proposed sub-pixel motion estimation algorithm, this assumption is still utilized.

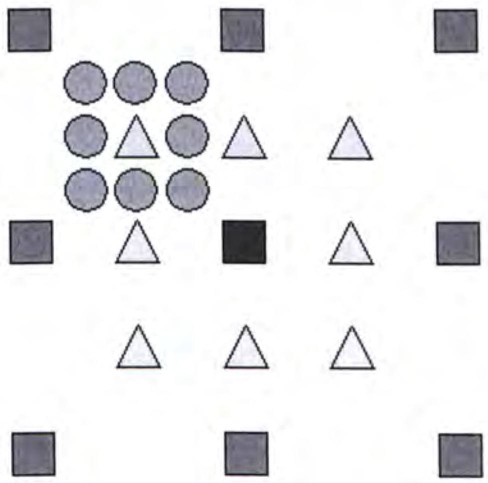


Figure 5.1 The sub-pixel motion estimation process used for Fast Full Search in the reference software of H.264.

The proposed algorithm first finds a prediction best match. Three different search points are compared and the best one is the prediction best match. The three search points P0, P1 and P2 are defined in Equation 5.1. After the prediction best match is found, it is regarded as the search center and a refinement search is performed. The search pattern selected for the refinement search is small diamond pattern as shown in Figure 2.8 (a). The search process terminates until the current best match is at the center or the cost of the best match in the current stage satisfies the condition defined in Equation 5.2. In our experiment, the values of a and b in Equation 5.2 were set to 0.6 and 0.2.

P0 = (0, 0)

P1 = (Pred – Int_best_match) % 4 (5.1)

P2 = Pred % 4

where % indicates modulus operation,

Pred is the prediction motion vector, and

Int_best_match is the best match of integer pixel block matching.

(Pred and Int_best_match are in quarter pixel precision. P0, P1 and P2 are expressed relative to Int_best_match.)

```
if (best_cost < a × Threshold) or (best_cost < b × intra_mean_cost)
    the search process is terminated
otherwise
    continue  to search                                        (5.2)
```

where intra_mean_cost is the mean of the costs of the intra coded macroblocks,

best_cost is the cost of the current best sub-pixel search point,

a and b are tuning factors for achieving a good balance between complexity and coding performance, and

Threshold is set to the cost of the integer pixel best match.

In addition to the mentioned process, during our implementation, a matrix is used to record the sub-pixel search points which have already been checked. Therefore, the search points will not be searched repetitively.

The experimental results illustrating the performance and complexity of the proposed algorithm are shown in Table 5.1. The experiment was performed on the Pocket PC platform with IPPPP… GOP structure. QP was set to 28, 32, 36 and 40. The ΔBitrate or the equivalent ΔPSNR are computed with the method proposed in [11] when compared with the sub-pixel motion estimation algorithm used for Fast Full Search in the reference software of H.264. The Encoding Time (%) is derived with Equation 5.13. From the results, it can be observed that the increment of bitrate is less than 1 % and the reduction in the total encoding time is more than 10% on average.

Sequence	ΔBitrate (%)	ΔPSNR (dB)	Encoding Time (%)
Carphone	0.1	-0.01	-15.4
Container	0.28	-0.01	-10.27
Foreman	2.3	-0.11	-17.61
Grandma	0.42	-0.01	-8.34
MissA	1.44	-0.07	-9.66
Salesman	0.26	-0.01	-11.23
Silent	0	-0.01	-13.06
Average	0.686	-0.033	-12.224

Table 5.1 The performance and complexity of the proposed algorithm.

5.1.2 Interpolation

5.1.2.1 Revision of Luma Interpolation

The luma interpolation process of H.264 includes two parts, the half and quarter pixel interpolation. For the half-pixel interpolation, a six-tap filter with tap values (1, -5, 20, 20, -5, 1) is applied. In Figure 5.2, aa, bb, cc, dd, ee, ff, gg and hh are the intermediate values of the half pixels. They are derived in a way similar to Equations 5.6 and 5.7. b, h, j, m, and s are the half pixels which are derived in a way similar to Equations 5.3, 5.4, 5.5, 5.8 and 5.9. The quarter pixels are derived in a way similar to Equations 5.10 and 5.11.

$$b = \text{Clip1}\ ((b1 + 16) >> 5) \tag{5.3}$$

$$h = \text{Clip1}\ ((h1 + 16) >> 5) \tag{5.4}$$

$$j = \text{Clip1}(\ (\ j1 + 512\ ) >> 10\ ) \tag{5.5}$$

where b1, h1 and j1 are derived with Equations 5.6, 5.7, 5.8 and 5.9.

$$b1 = (E - 5F + 20G + 20H - 5I + J) \tag{5.6}$$

$$h1 = (A - 5C + 20G + 20M - 5R + T) \tag{5.7}$$

$$j1 = cc - 5dd + 20h1 + 20m1 - 5ee + ff \tag{5.8}$$

or

$$j1 = aa - 5bb + 20b1 + 20s1 - 5gg + hh \tag{5.9}$$

$$a = (G + b + 1) >> 1 \tag{5.10}$$

$$e = (b + h + 1) >> 1 \tag{5.11}$$

$$\begin{array}{lll} \text{Clip1}(x)\text{: If} & x > (1 << \text{BitDepth}\ ) - 1,\ x = (\ 1 << \text{BitDepth}\ ) - 1 & \\ & \text{else if} \quad x < 0,\ x = 0 & (5.12) \\ & \text{else} \quad x = x & \end{array}$$

where << is arithmetic left shift,

>> is arithmetic right shift, and

BitDepth is the number of bits used to represent each luma component.

The results of Equations 5.8 and 5.9 are the same and this is proved as follows:

$cc = Y_1 - 5Y_5 + 20E + 20K - 5Z_1 + Z_5$

$dd = Y_2 - 5Y_6 + 20F + 20L - 5Z_2 + Z_6$

$h1 = A - 5C + 20G + 20M - 5R + T$

$m1 = B - 5D + 20H + 20N - 5S + U$

$ee = Y_3 - 5Y_7 + 20I + 20P - 5Z_3 + Z_7$

$ff = Y_4 - 5Y_8 + 20J + 20Q - 5Z_4 + Z_8$

$$\begin{aligned} j1 = cc - 5dd + 20h1 + 20m1 - 5ee + ff \ = & \ [\ Y_1 - 5Y_5 + 20E + 20K - 5Z_1 + Z_5 + \\ & \ \ Y_4 - 5Y_8 + 20J + 20Q - 5Z_4 + Z_8] \ - \\ 5 & \ [\ Y_2 - 5Y_6 + 20F + 20L - 5Z_2 + Z_6 + \\ & \ \ Y_3 - 5Y_7 + 20I + 20P - 5Z_3 + Z_7] \ + \\ 20 & \ [\ A - 5C + 20G + 20M - 5R + T + \\ & \ \ B - 5D + 20H + 20N - 5S + U] \end{aligned}$$

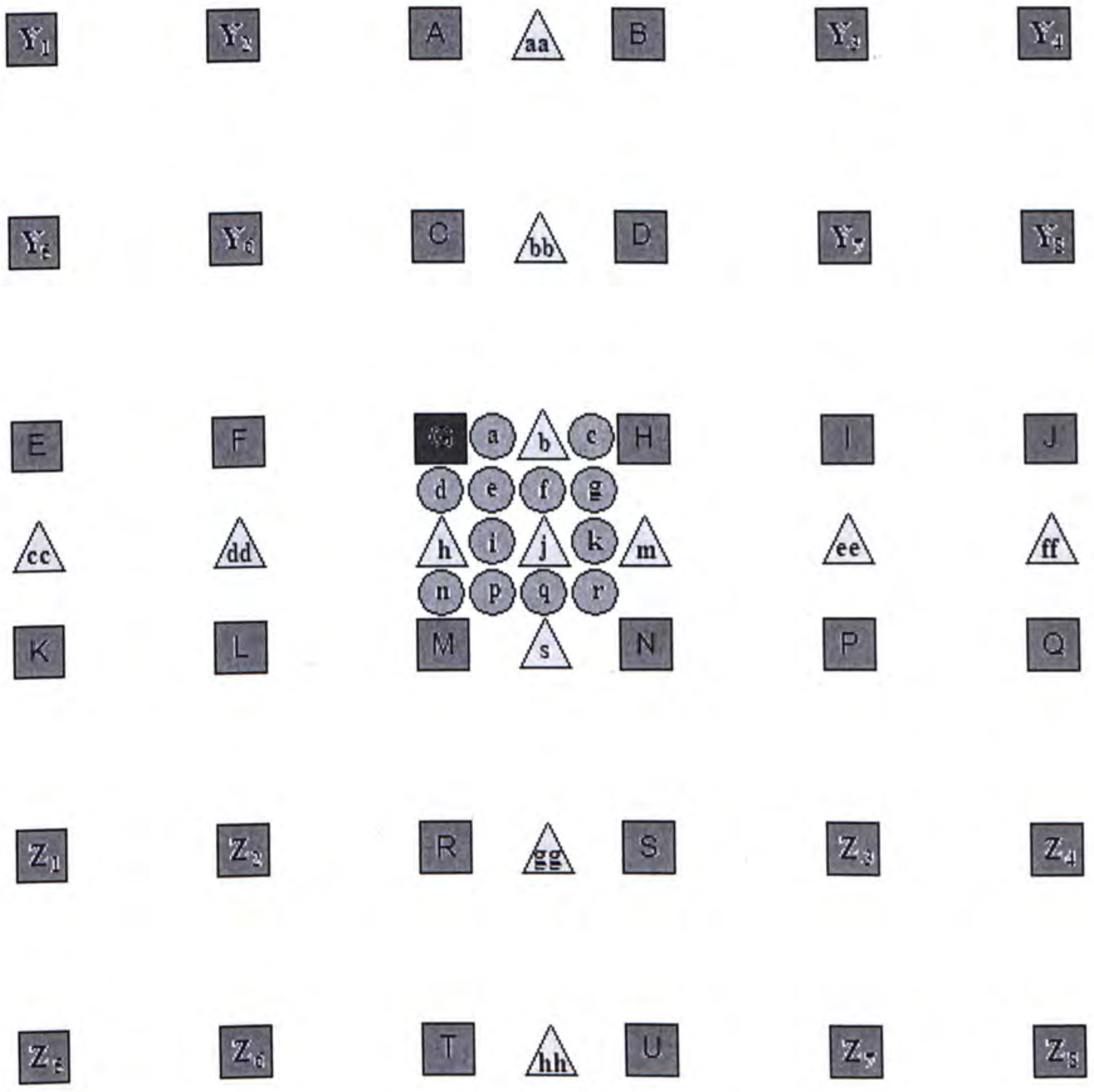


Figure 5.2 The squares are integer pixels. b, h, j, m, s are half pixels and the circles are quarter pixels.

$aa = Y_1 - 5Y_2 + 20A + 20B - 5Y_3 + Y_4$

$bb = Y_5 - 5Y_6 + 20C + 20D - 5Y_7 + Y_8$

$b1 = E - 5F + 20G + 20H - 5I + J$

$s1 = K - 5L + 20M + 20N - 5P + Q$

$gg = Z_1 - 5Z_2 + 20R + 20S - 5Z_3 + Z_4$

$hh = Z_5 - 5Z_6 + 20T + 20U - 5Z_7 + Z_8$

$$\begin{aligned} j1 = aa - 5bb + 20b1 + 20s1 - 5gg + hh &= [\ Y_1 - 5Y_2 + 20A + 20B - 5Y_3 + Y_4 + \\ &\qquad Z_5 - 5Z_6 + 20T + 20U - 5Z_7 + Z_8] - \\ &\quad 5\ [\ Y_5 - 5Y_6 + 20C + 20D - 5Y_7 + Y_8 + \\ &\qquad Z_1 - 5Z_2 + 20R + 20S - 5Z_3 + Z_4] + \\ &\quad 20\ [\ E - 5F + 20G + 20H - 5I + J + \\ &\qquad K - 5L + 20M + 20N - 5P + Q] \\ &= cc - 5dd + 20h1 + 20m1 - 5ee + ff \end{aligned}$$

The details of the luma interpolation process can be found in [30].

5.1.2.2 Fast Interpolation

In Section 5.1.2.1, the interpolation process for luma component is reviewed. Certainly, interpolation of all sub-pixel positions consumes a large amount of computing resources and thus this is not a suitable implementation for real-time applications. It can be observed that the interpolation of half-pixel is highly complex because a six-tap filter is used, and the derivation of quarter pixel requires at least one half pixel. Therefore, repetitive interpolation of the half pixel should be avoided.

For the sub-pixel motion estimation algorithm adopted in the reference software of H.264, the half pixels are derived and stored first. Then the quarter pixels can be derived from the stored half pixels. Unlike this algorithm, the fast sub-pixel motion estimation algorithm introduced in Section 5.1.1 has chances that the half pixels are not derived but they are required for deriving quarter pixels which are necessary to be evaluated in the current stage. To reduce the complexity, during our implementation, when a half pixel is derived (no matter it is derived because it is required or for deriving a quarter pixel), its value is stored for future usage. This can greatly reduce the complexity. In Figure 5.3, the triangles are the half pixels which may be stored during the proposed sub-pixel fast motion estimation process. The center square is the best integer pixel best match.

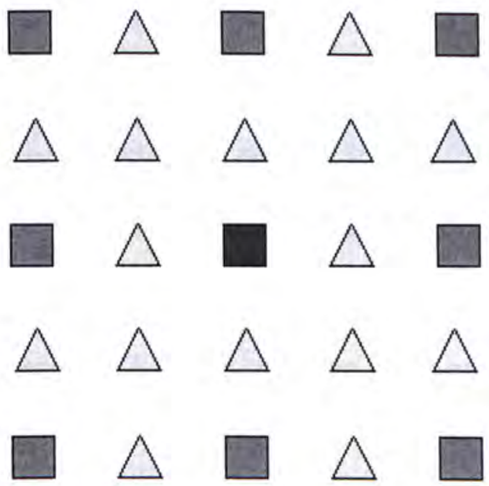


Figure 5.3 Triangles represent the half pixels which may be stored during the proposed sub-pixel fast motion estimation process.

Table 5.2 illustrates the reduction in complexity when the fast interpolation method is applied. The setting of the experiment was the same as that applied for the proposed fast sub-pixel motion estimation algorithm. The Encoding Time (%) is derived with

Equation 5.13. It can be observed that the total encoding time can be reduced by about 4.1% on average. In fact, the encoding time for the sequence *Foreman* is the longest as shown in Table 5.9. From Table 5.2, the total encoding time reduction for *Foreman* is about 7.4 % and thus this method can help to achieve real-time applications efficiently. In addition, this method reduces the complexity without degrading the coding performance.

Sequence	Encoding Time (%)
Carphone	-6.421
Container	-0.775
Foreman	-7.429
Grandma	-2.650
MissA	-3.746
Salesman	-3.183
Silent	-4.677
Average	-4.126

Table 5.2 The reduction in complexity when the fast interpolation method is applied.

$$\text{Encoding Time (\%)} = \frac{\text{New\_Encoding\_Time} - \text{Original\_Encoding\_time}}{\text{Original\_Encoding\_time}} \times 100 \quad (5.13)$$

5.1.3 Skipping Inverse ICT and Inverse Quantization Depends on Coded Block Pattern

In H.264 standard, there is a syntax element called coded_block_pattern (CBP). In the Baseline Profile, a macroblock comprises a 16x16 luma block and two 8x8 chroma blocks. The CBP is used to specify which of the luma or chroma 8x8 blocks contains non-zero transform coefficient levels. For macroblock type which is not equal to Intra_16x16, this syntax element is present in the bitstream and two variables CodedBlockPatternLuma and CodedBlockPatternChroma are derived with Equations 5.14 and 5.15. When macroblock type is Intra_16x16, the CodedBlockPatternLuma and CodedBlockPatternChroma can be derived from the macroblock type.

$$\text{CodedBlockPatternLuma} = \text{coded\_block\_pattern} \,\%\, 16 \tag{5.14}$$

$$\text{CodedBlockPatternChroma} = \text{coded\_block\_pattern} \,/\, 16 \tag{5.15}$$

where % is the modulus operation, and

/ is integer division with truncation of the result toward zero.

CodedBlockPatternLuma specifies which of the four 8x8 luma blocks contains non-zero transform coefficient levels. The meaning of CodedBlockPatternChroma is specified in Table 5.3 [30].

CodedBlockPatternChroma	**Description**
0	All chroma transform coefficient levels are equal to 0.
1	One or more chroma DC transform coefficient levels shall be non-zero valued. All chroma AC transform coefficient levels are equal to 0.
2	Zero or more chroma DC transform coefficient levels are non-zero valued. One or more chroma AC transform coefficient levels shall be non-zero valued.

Table 5.3 The meaning of CodedBlockPatternChroma [30].

It is well known that inverse ICT and inverse quantization are time consuming processes. If their implementation process can be skipped, the complexity of the codec can be reduced. From the above explanation, it can be noticed that CodedBlockPatternLuma and CodedBlockPatternChroma can be used to skip some of the inverse ICT and inverse quantization processes because they specify whether the quantized transform coefficients of a block are all zeros. The inverse ICT or inverse Hadamard Transform and the inverse quantization process for luma and chroma DC or AC blocks can be skipped when the quantized coefficient levels of a block are all zeros. The output matrix can be derived by assigning zeros to all of the elements of the matrix. Consequently, the complexity of the codec can be reduced significantly.

5.2 Code Level Optimizations

Even though algorithms can be applied to reduce the complexity of a system, it is still insufficient to develop a Pocket PC based real-time H.264 codec without applying other optimization techniques. In this section, code level optimizations are discussed. There are many code level optimization techniques, such as Loop Optimizations, Numerical Operation Optimizations, Memory Operation Optimizations, etc. The following are some examples for Loop Optimizations.

5.2.1 Merging Loops

As implied from its name, this method aims to reduce the number of loops by combining them. This method is simple but efficient. Consider the following C code:

Non-Optimized Code:

```
for (a=0; a<100; a++)
        x[a] = a;
for (b=20; b<120; b++)
```

```
        y[b-20] = b;
```

Optimized Code:

```
for (a=0; a<100; a++){
        x[a] = a;
        y[a] = a+20;
}
```

Since a *for* loop consists of initializations, comparisons and counters, remove a *for* loop can reduce the complexity in a great amount. In the above example, a *for* loop is removed and one initialization, 100 comparisons and 100 additions are removed.

5.2.2 Moving Independent Code outside the Loop

Each statement within a loop is implemented many times. Accordingly, the complexity of a loop can be reduced by pre-computed some variables or statements which are invariant within the loop. Consider the following example:

Non-Optimized Code:

```
for (y=0; y<h; y++){
        for (x=0; x<w; x++){
                N[y][x] = [(8-Fx)*(8-Fy)*P[y][x] + Fx*(8-Fy)*[y][x+1] +
                                (8-Fx)*Fy*P[y+1][x] + Fx*Fy*P[y+1][x+1] + 32] >> 6
        }
}
```

Optimized Code:

```
Aweight = (8-Fx)*(8-Fy);
Bweight = Fx*(8-Fy);
Cweight = (8-Fx)*Fy;
Dweight = Fx*Fy;
for (y=0; y<h; y++){
      for (x=0; x<w; x++){
            N[y][x] = [Aweight*P[y][x] + Bweight*[y][x+1] +
                         Cweight*P[y+1][x] + Dweight*P[y+1][x+1] + 32] >> 6
      }
}
```

It can be noticed that the above example is the implementation of Equation 1.10, i.e. the chroma interpolation process. In the above example, the loop-independent codes are implemented outside the loop. Accordingly, the number of multiplications within the loop is reduced and the complexity of the chroma interpolation process is reduced.

5.2.3 Unrolling Loops

As mentioned before, a *for* loop consists of initializations, comparisons and counters. Unrolling loops can reduce the number of comparisons and counters and thus this may reduce the complexity. The following is an example to illustrate the unrolling loops process.

Original Code:

```
int SAD = 0;

for (y=0; y<4; y++){

        for (x=0; x<4; x++){

                SAD = SAD + abs(Org[Y+y][x] – Ref[Y+y][x]);

        }

}
```

Code with unrolled loops:

```
int SAD = 0;

for (y=0; y<4; y++){

        SAD = SAD + abs(Org[Y+y][x] – Ref[Y+y][x]);

        SAD = SAD + abs(Org[Y+y][x+1] – Ref[Y+y][x+1]);

        SAD = SAD + abs(Org[Y+y][x+2] – Ref[Y+y][x+2]);

        SAD = SAD + abs(Org[Y+y][x+3] – Ref[Y+y][x+3]);

}
```

In the above example, the number of comparisons and additions required for the loops is reduced. Ideally, the complexity of the codes is reduced. However, this increases the code size and can produce a situation in which the program ceases to fit within the available RAM and the operating system would be forced to use the hard disk which is much slower. Cache also has a size limit and situations may occur in which an unrolled loop would not fit within it. Therefore, unrolling loops or not depends on different situations and can only be decided through experiment [29].

5.3 Experimental Results

The codec was implemented on a HP 4700 Pocket PC which has an embedded PXA27x processor produced by Intel. The processor runs at 624 M Hz and the Pocket PC has 92M RAM. The profile selected is the Baseline Profile. Seven QCIF 4:2:0 video sequences were tested. The sequences include *Carphone*, *Container*, *Foreman*, *Grandma*, *MissA*, *Salesman* and *Silent*. Some of them have large motion and some have less. Two tests were carried out. The first test used only intra coding and 150 pictures were tested for each video sequence. The second one used only one I picture and all of the remaining pictures were P pictures and 150 frames were tested for each video sequence. The tested QPs were 28, 32, 36 and 40. Deblocking filter was applied for both tests. All the bitstreams generated by the developed encoder can be decoded by JM 10.1 or the developed decoder correctly.

After all the optimizations, which include algorithmic optimizations, code level optimizations and instruction level optimizations, are applied to the codec, the encoding and decoding speeds of the codec were evaluated. In addition, the coding performance of the encoder was compared with those of JM 10.1, the reference software of H.264, and x.264 which is a well known open source H.264 encoder and has excellent encoding speed. The coding performance is compared in ΔPSNR or the equivalent ΔBitrate which are computed with the method proposed in [11]. The ΔPSNR and ΔBitrate are with respect to JM 10.1. In addition, the RD-curves are plotted and the subjective quality of the reconstructed video sequences is shown.

Table 5.4 and Table 5.7 illustrate the coding performance of the developed encoder for IIIII...and IPPPP... GOP structures, respectively. Table 5.5 and Table 5.8

compare the coding performance of the developed encoder with that of x.264. It can be observed that the performance of developed encoder is better than that of x.264 and the coding performance is similar to that of JM 10.1 on average. The performance of the developed encoder is even better than that of JM 10.1 when only intra coding is applied. Figure 5.4 and 5.5 show the RD curves of JM 10.1, the developed encoder and x.264 encoder for comparisons. From Figure 5.5, it can be noticed that the coding performance of our encoder is not as good as that of the reference software when the bitrate is high. The reason is that the integer pixel motion estimation algorithm [27] employed in our encoder is susceptive to be trapped in local minimum easily. When bitrate is low, the best match is most likely located near the prediction motion vector because the cost of encoding the motion vectors is large. In contrast, the cost of encoding the motion vectors is low when bitrate is high and thus the best match may not be located near the prediction motion vector. Therefore, the employed integer pixel motion estimation algorithm may not find the global minimum search point and the coding performance is affected.

Figure 5.6 shows the visual quality of the decoded pictures. All the pictures in the figure are the 150th picture of each sequence. The pictures in the first column are the original pictures, and those in the second and third columns are the decoded pictures with IIIII... and IPPPP... GOP structures, respectively and QP is equaled to 28. It can be observed that the decoded pictures have a little distortion especially in highly texture region such as the hair of a human. The pictures shown in Figure 5.6 are displayed by a desktop PC because the display of the Pocket PC is only 16-bit. Accordingly, the Pocket PC introduces some distortion to the decoded video.

From Table 5.6 and Table 5.9, it can be observed that the picture rate of the developed encoder is more than 25 frames per second (fps) for all the tested video sequences with different QPs. Accordingly, it can encode video with QCIF 4:2:0 format in real-time on the Pocket PC.

Table 5.10 and Table 5.11 show the decoding speed of the developed decoder for different video sequences with IIIII… and IPPPP… GOP structures, respectively, and with different QPs. The decoder can decode, on average, 45.028 and 163.25 fps for IIIII… and IPPPP… GOP structures, respectively. Accordingly, the decoding speed of the developed decoder is much better than 25 fps and thus the decoder is suitable for real-time applications.

QP	28		32		36		40	
Sequence	PSNR (dB)	BR (kbps)	PSNR (dB)	BR (kbps)	PSNR (dB)	BR (kbps)	PSNR (dB)	BR (kbps)
Carphone	38.16	631.3	35.26	434.1	32.59	300.9	29.88	204
Container	37.00	786.3	34.1	537.1	31.36	357.5	28.76	240.3
Foreman	36.57	791.2	33.72	519	31.05	343	28.58	236.3
Grandma	37.06	671.8	34.31	424.1	32.03	263.3	30.07	162.0
MissA	40.7	363.3	38.16	241.5	35.81	162.5	33.57	110.4
Salesman	36.03	915.9	32.99	586.7	30.34	367.2	27.99	223.9
Silent	36.22	851.6	33.36	553.7	30.89	350.5	28.5	230.0

Table 5.4 Coding performance of the developed encoder for IIIII… GOP structure.

Sequence	Developed Encoder		x.264	
	ΔPSNR (dB)	ΔBitrate (%)	ΔPSNR (dB)	ΔBitrate (%)
Carphone	0.01	-0.13	-0.17	2.46
Container	0.21	-3.03	0.04	-0.55
Foreman	0.06	-0.92	-0.09	1.26
Grandma	0.14	-2.53	-0.09	1.72
MissA	0.09	-1.23	-0.34	5.23
Salesman	0.08	-1.21	-0.07	1.17
Silent	-0.02	0.45	-0.21	3.6
Average	0.08	-1.23	-0.13	2.13

Table 5.5 Coding performance comparisons of the developed encoder and x.264 encoder for IIIII… GOP structure.

QP	28	32	36	40
Sequence	(fps)	(fps)	(fps)	(fps)
Carphone	26.28	26.35	27.9	27.84
Container	26.41	27.25	27.8	27.82
Foreman	28.03	28.76	29.41	29.05
Grandma	25.79	26.92	27.15	27.19
MissA	26.52	26.6	26.8	26.84
Salesman	27.72	28.01	28.43	28.15
Silent	28.3	28.63	28.54	27.78
Average	27	27.5	28.04	27.81

Table 5.6 Encoding speed of the developed encoder in terms of fps for IIIII… GOP

structure.

QP	28		32		36		40	
Sequence	PSNR (dB)	BR (kbps)	PSNR (dB)	BR (kbps)	PSNR (dB)	BR (kbps)	PSNR (dB)	BR (kbps)
Carphone	37.18	131.12	34.12	66.49	31.43	33.07	28.76	17.33
Container	36.04	57.63	33.25	29.15	30.60	14.58	28.07	8.11
Foreman	35.85	166.94	33.05	86.40	30.32	46.37	27.65	27.50
Grandma	36.65	48.44	33.97	22.69	31.67	11.39	29.71	6.36
MissA	40.03	39.67	37.37	20.79	34.99	11.99	32.40	7.89
Salesman	35.48	76.87	32.50	41.90	29.82	22.57	27.38	12.13
Silent	35.76	96.97	32.86	56.28	30.37	31.84	27.87	18.02

Table 5.7 Coding performance of the developed encoder for IPPPP… GOP structure.

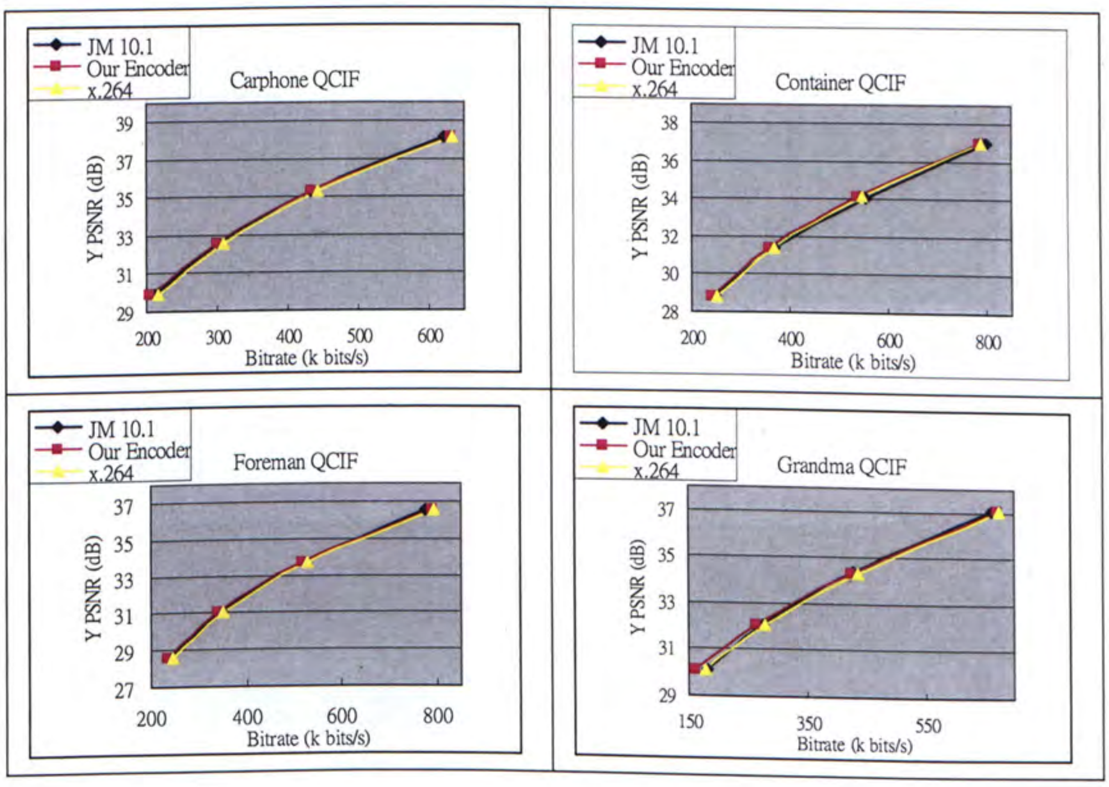
Sequence	Developed Encoder		x.264	
	ΔPSNR (dB)	ΔBitrate (%)	ΔPSNR (dB)	ΔBitrate (%)
Carphone	-0.39	9.52	-0.51	12.44
Container	-0.35	8.14	-0.17	3.53
Foreman	-0.47	10.5	-0.6	13.54
Grandma	-0.27	7.18	-0.3	8.21
MissA	0.64	-10.46	0.07	-1.34
Salesman	-0.47	10.73	-0.31	6.79
Silent	-0.39	8.66	-0.31	6.56
Average	-0.24	6.32	-0.3	7.1

Table 5.8 Coding performance comparisons of the developed encoder and x.264

encoder for IPPPP… GOP structure.

QP	28	32	36	40
Sequence	(fps)	(fps)	(fps)	(fps)
Carphone	31.46	39.06	49.08	62.5
Container	60.88	73.24	81.52	88.44
Foreman	28.07	33.24	39.72	48.32
Grandma	59.34	69.44	79.45	90.14
MissA	54.19	63.34	71.56	79.45
Salesman	55.15	62.09	71.02	82.24
Silent	43.6	49.87	57.87	68.43
Average	47.53	55.75	64.32	74.22

Table 5.9 Encoding speed of the developed encoder in terms of fps for IPPPP… GOP structure.



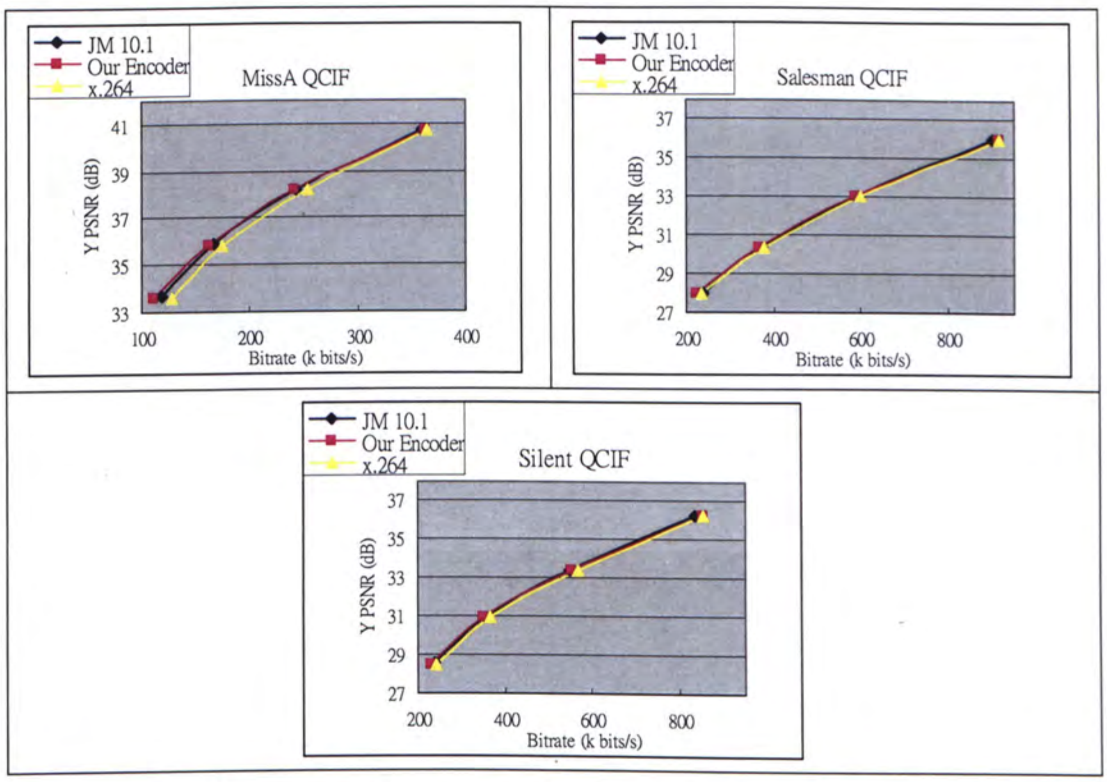
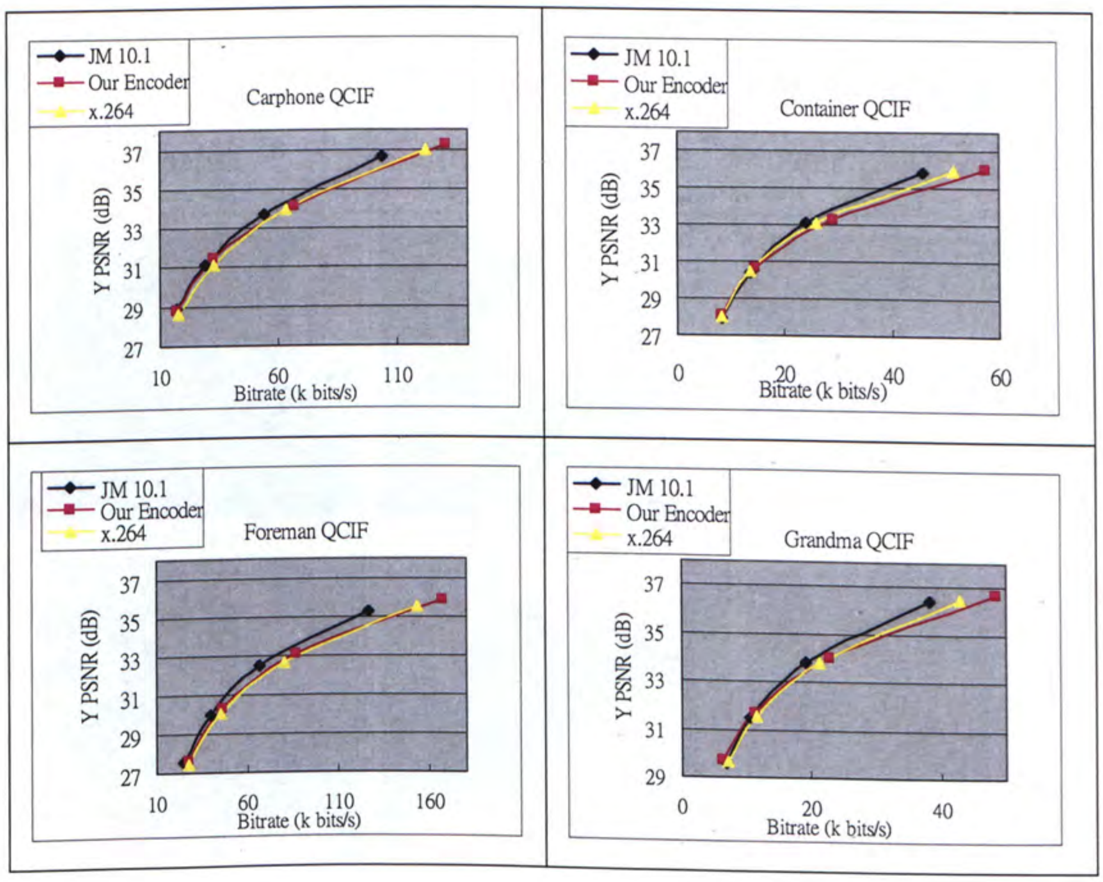


Figure 5.4 RD curves of JM 10.1, the developed encoder and x.264 encoder for different video sequences with IIIII… GOP structure.



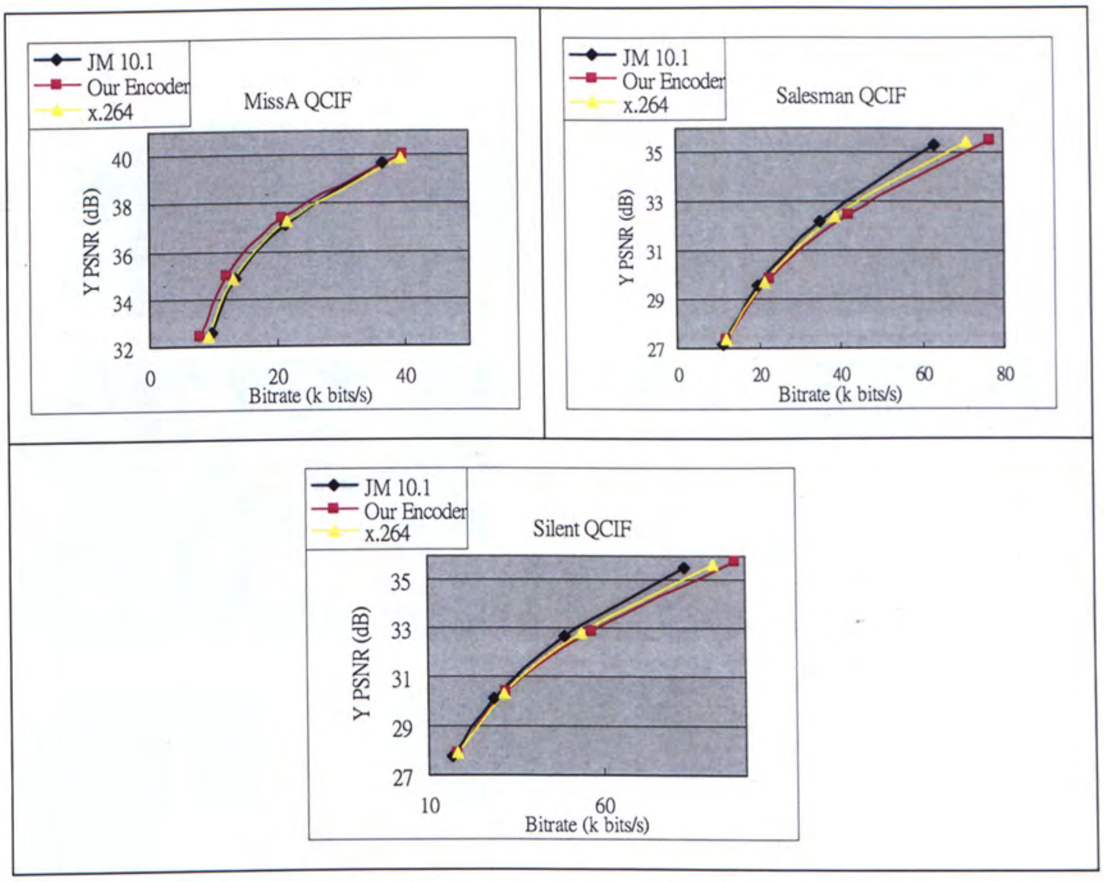
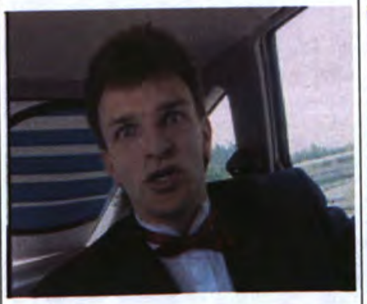
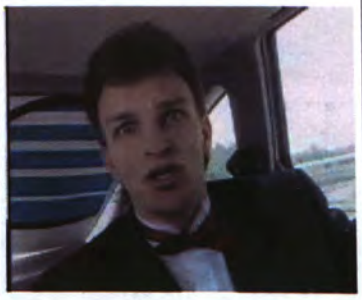
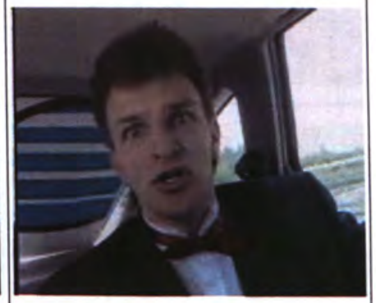
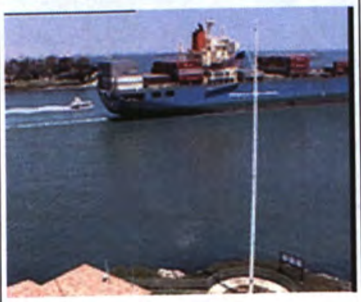
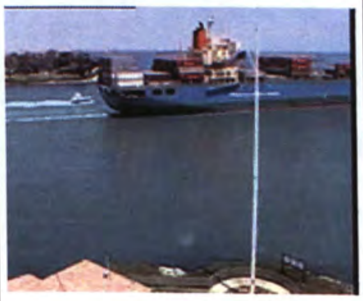
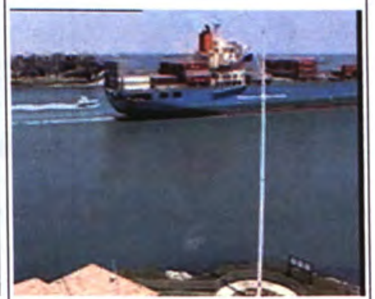


Figure 5.5 RD curves of JM 10.1, the developed encoder and x.264 encoder for different video sequences with IPPPP… GOP structure.

Original	IIIII… GOP structure	IPPPP… GOP structure
		
		

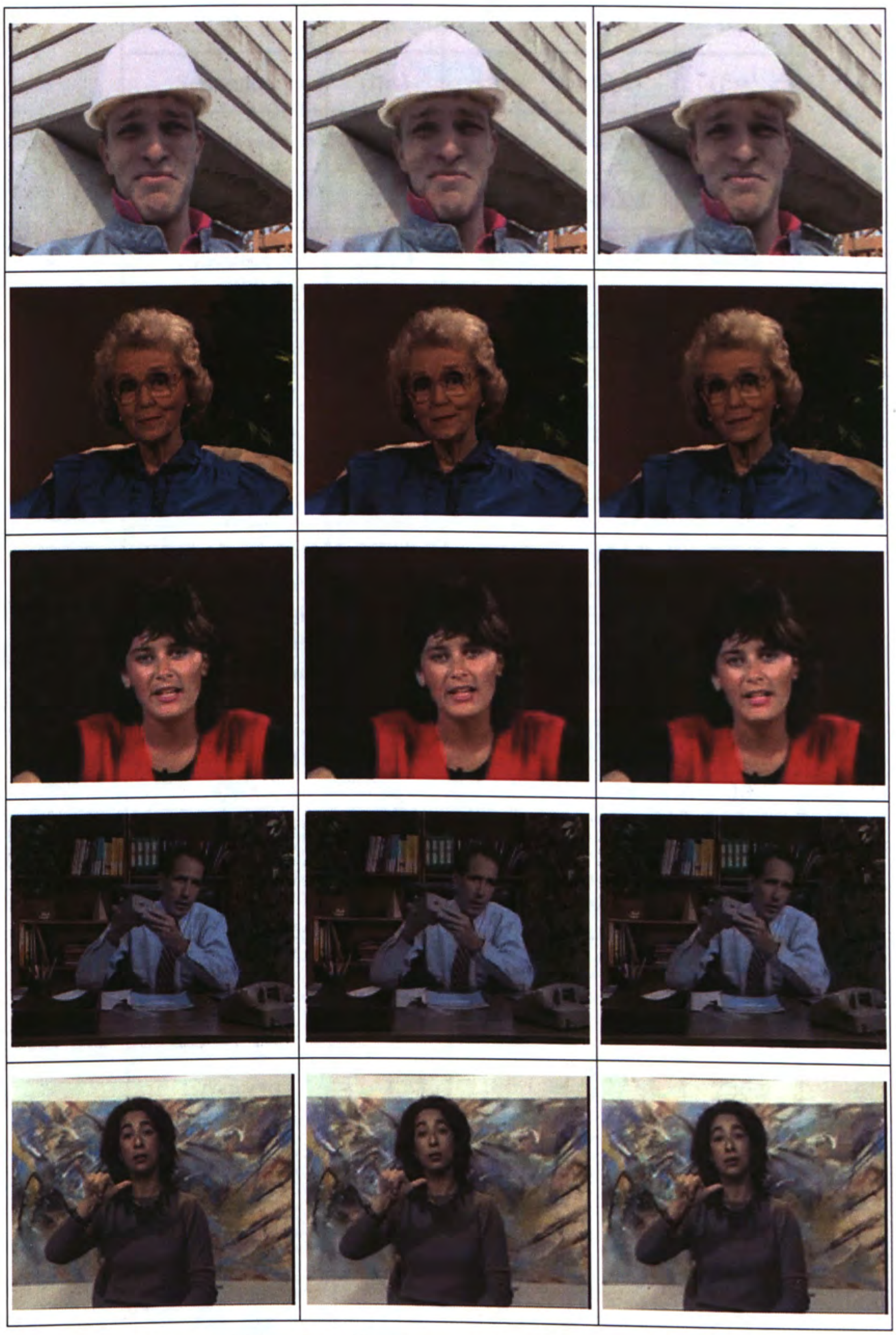


Figure 5.6 Visual quality comparisons of different sequences.

QP	28	32	36	40
Sequence	(fps)	(fps)	(fps)	(fps)
Carphone	35.59	42.28	48.91	55.58
Container	33.34	39.26	46.08	52.34
Foreman	32.03	39.54	47.69	54.88
Grandma	34.28	42.02	50.56	57.92
MissA	44.76	51.44	57.25	61.93
Salesman	29.18	36.66	45.57	54.09
Silent	30.56	37.75	46.27	53.02
Average	34.25	41.28	48.90	55.68
Overall Average				45.028

Table 5.10 The decoding speed of the developed decoder for IIIII… GOP structure.

QP	28	32	36	40
Sequence	(fps)	(fps)	(fps)	(fps)
Carphone	74.89	103.45	141.91	185.87
Container	142.05	182.48	220.91	254.24
Foreman	64.05	87.11	113.9	144.51
Grandma	144.23	186.1	227.96	266.9
MissA	143.68	179.86	212.77	242.72
Salesman	121.16	150.91	185.41	222.22
Silent	101.01	125.1	155.76	189.87
Average	113.0	145.00	179.80	215.19
Overall Average				163.25

Table 5.11 The decoding speed of the developed decoder for IPPPP… GOP structure.

5.4 Applications

The developed Pocket PC based real-time H.264 codec has wide applications and competitive abilities. Firstly, it can encode and decode video with QCIF 4:2:0 format in real-time (more than 25 fps) on Pocket PC which has relative low processing power while the coding performance is still very good. Secondly, the developed codec is software-based and thus could be used on other platforms. Users only require to download and run the execution files on their mobile platform and the codec can work properly. Thirdly, besides the codec, other modules which include display module, video capture module and wireless transmission module are successfully developed for the codec. Thus, in fact, a system is developed. This system has commercial value and diverse applications such as video conferencing, video telephony, video streaming and all real-time related video applications.

Chapter 6 Conclusions and Future Development

6.1 Conclusions

6.1.1 Enhancement Techniques for Intra Block Matching

Intra block matching performs prediction with the pixels of the current slice in a way similar to that of motion estimation and compensation. Within a search range, a best match is found with a predefined cost function. The best match is regarded as the prediction. Even though this technique can improve the coding performance, the number of bits used to encode intra blocks is still very high. Therefore, several enhancement techniques, which include best match prediction, multiple best matches, novel padding method, skip mode, etc., are proposed to further improve the coding performance. Moreover, several techniques are proposed for reducing the complexity of the variable block size intra block matching process. Experimental results show that the coding performance can be improved significantly and more than 1 dB and 0.6 dB gains in PSNR were achieved for intra and hybrid coding, respectively.

6.1.2 Enhanced SAD Reuse Fast Motion Estimation

The complexity of motion estimation is high. In H.264, variable block size motion estimation is employed and its complexity is much higher than that of fixed block size motion estimation. Therefore, fast algorithms are necessary. In Chapter 4, a fast variable block size motion estimation algorithm is introduced. It reuses the SAD to reduce the complexity and uses pattern-based motion estimation and refinement search to maintain good coding performance in terms of PSNR and bitrate. With the proposed algorithm, the encoding speed can be improved significantly while the coding performance can be maintained. Experimental results illustrate that the proposed algorithm reduces the variable block size motion estimation time by approximately 90% when compared with that of FFS. This algorithm is the fastest when compared with the fast motion estimation algorithms adopted in the H.264 reference software.

6.1.3 Development of Real-Time H.264 Codec on Pocket PC

The development of a Pocket PC software-based real-time H.264 codec is described in Chapter 5. Various algorithms are proposed and applied during the development. These algorithms include fast interpolation, fast integer and sub-pixel motion estimation, early skip termination, etc. In addition, code and instruction level optimizations are applied to further reduce the complexity. Consequently, the developed codec can encode and decode video in real-time on the Pocket PC. The developed Pocket PC software-based real-time H.264 codec has wide applications and competitive abilities. Firstly, it can encode and decode video with QCIF 4:2:0 format in real-time (more than 25 fps) on the Pocket PC which has relatively low processing power while the coding performance is still very good. Secondly, the developed codec is software-based and thus could be used on other platforms. Thirdly, besides the codec, other modules which include display module, video capture module and wireless transmission module are successfully developed for the codec. Accordingly, this system has commercial value and diverse applications such as video conferencing, video telephony, video streaming and all real-time related video applications.

6.2 Future Development

The techniques proposed for intra block matching improve the coding performance of H.264. However, the complexity is also increased. All the proposed modes are evaluated one by one and the RD-costs are compared with each other. Accordingly, the complexity is high. Fast intra block matching mode decision algorithm is necessary to be developed. In addition, as mentioned in Chapter 3, some of the proposed techniques can also be applied to inter prediction for further improving the coding performance of the encoder. For the fast variable block size motion estimation algorithm, early termination techniques can be applied to further reduce the complexity.

Bibliography

[1] K.K. Ma and P.I. Hosur, "Performance Report of Motion Vector Field Adaptive Search Technique (MVFAST)", *ISO/IEC JTC1/SC29/WG11 MPEG99/ m5851*, Noordwijkerhout, NL, Mar'00.

[2] Zhibo Chen, Peng Zhou, Yun He, Yidong Chen, "Fast Integer Pel and Fractional Pel Motion Estimation for JVT", JVT-F017, in: 6th Meeting, Awaji, JP, 2002, pp. 5–13.C. Jones (private communication).

[3] A. M. Tourapis, " Enhanced Predictive Zonal Search for Single and Multiple Frame Motion Estimation", *proceedings of Visual Communications and Image Processing 2002 (VCIP-2002)*, pp. 1069-79, San Jose, CA, January 2002.

[4] Hasan F. Ates, Yucel Altunbasak, "SAD reuse in hierarchical motion estimation for the H.264 encoder", Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Philadelphia, PA, March 2005.

[5] Chang-Hsing Lee and Ling-Hwei Chen, "A Fast Motion Estimation Algorithm Based on the Block Sun Pyramid", IEEE Trans. Image Processing, vol. 6, Nov. 1997.

[6] Renxiang Li, Bing Zeng, and Ming L. Liou, "A New Three-Step Search Algorithm for Block Motion Estimation", IEEE Trans. Circuit Syst. Video Technol., vol. 3, Aug. 1994.

[7] Thomas Wiegand, Gary J. Sullivan, Gisle Bjontegaard, and Ajay Luthra, "Overview of the H.264 / AVC Video Coding Standard", IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, JULY 2003

[8] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block matching motion estimation", IEEE Trans. Image Processing, vol.9 po.287-290 Feb 2000.

[9] J. Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, "A novel unrestricted center-biased diamond search algorithm for block motion estimation", IEEE Trans. Circuit Syst. Video Technol., vol. 8, pp. 369-377, Aug. 1998.

[10] H.264 Reference Software JM 10.2.

[11] G. Bjontegaard, "Calculation of average PSNR differences between RD-curves", ITU-T SG16 Doc. VCEG-M33, 2001.

[12] Zhibo Chen, Yun He, etc. "Fast Integer and Fractional Pel Motion estimation", JVT-E045.doc, 5th Meeting: Geneva, Switzerland, 9-17 October, 2002.

[13] J.Jain, A.Jain, "Displacement measurement and its application in interframe image coding", IEEE Transactions on Communications, vol.COM-29, pp.1799-1806, Dec. 1981.

[14] Ce Zhu, Xiao Lin, and Lap-Pui Chau, "Hexagon-Based Search Patten for Fast Block Motion Estimation", IEEE Trans. on CSVT, pp.349-355, Vol.12, No.5, May, 2002.

[15] S. Zhu and K.-K. Ma, "A New Diamond Search Algorithm for Fast Block Matching Motion Estimation", *in Proc. Int. Conf. Inform., Commun.*, Signal Process., pp.292–296, Singapore, Sept.9–12, 1997.

[16] H. Y. Cheong, A. M. Tourapis, and P. Topiwala, "Fast Motion Estimation within the JVT codec", *ISO/IEC JTC1/SC29/WG11 and ITU-T Q6/SG16,* document JVT-E023, Oct'02.

[17] A. M. Tourapis, H. Y. Cheong, and P. Topiwala, "Fast ME in the JM reference software", *ISO/IEC JTC1/SC29/WG11 and ITU-T SG16 Q.6*, document JVT-P026, July 2005.

[18] Iain E.G. Richardson, H.264 and MPEG-4 video compression, Wiley, 2003.

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

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

[21] Gisle Bjøntegaard and Karl Lillevold, Context-adaptive VLC (CVLC) coding of coefficients, JVT document JVT-C028, Fairfax, Virginia, USA, 6-10 May, 2002.

[22] S. W. Golomb, Run-length encoding, IEEE Trans, on Inf. Theory, IT-12, pp. 399-401, 1966.

[23] S.L. Yu and C. Chrysafis, "New intra prediction using intra-macroblock motion compensation", JVT-C151r1-L.doc, Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG Meeting, May 2002

[24] Satoshi Kondo, Hisao, Sasai and Shinya Kadono, "Tree Structured Hybrid Intra Prediction", International Conference on Image Processing (ICIP), 2004

[25] Ki-Hun Han and Yung-Lyul Lee, "Fast Macroblock Mode Decision in H.264", TENCON 2004

[26] Enhanced SAD Reuse Fast Motion Estimation, Kai Lam Tang and King N. Ngan, Proceedings of SPIE, Vol. 6508, Visual Communications and Image Processing 2007

[27] WEI Zhenyu, JIANG Baochen, ZHANG Xudong, CHEN Yu, "A New Full-pixel and Sub-pixel Motion Vector Search Algorithm for Fast Block-matching Motion Estimation in H.264", ICIG 2004

[28] Zhenyu Wei, King Ngi, Ngan, "A Fast Macroblock Mode Decision Algorithm for H.264", IEEE Asia-Pacific Conference on Circuits and Systems, Singapore, 2006

[29] Kris Kaspersky, "Code Optimization: Effective Memory Usage", ISBN: 1931769249

[30] ISO/IEC 14496-10 and ITU-T Rec. H.264, Advanced Video Coding for generic audiovisual services, 2005

[31] Enhancement Techniques for Intra Block Matching, Kai Lam Tang and King N. Ngan, ICME'07 - Multimedia Coding and Processing Track