Content-Based Motion Compensation and its application to Video Compression

Marc Servais

Submitted for the degree of
Doctor of Philosophy
from the University of Surrey

Centre for Vision, Speech and Signal Processing
School of Electronics and Physical Sciences
University of Surrey
Guildford, United Kingdom

February 2006

Summary

Content-based approaches to motion compensation offer the advantage of being able to adapt to the spatial and temporal characteristics of a scene. Three such motion compensation techniques are described in detail, with one of the methods being integrated into a video codec.

The first approach operates by performing spatio-temporal segmentation of a frame. A split and merge approach is then used to ensure that motion characteristics are relatively homogeneous within each region. Region shape information is coded (by approximating the boundaries with polygons) and a triangular mesh is generated within each region. Translational and affine motion estimation are then performed on each triangle within the mesh. This approach offers an improvement in quality when compared to a regular mesh of the same size. However, it is difficult to control the number of triangles, since this depends on the segmentation and polygon approximation stages. As a result, this approach is difficult to integrate into a rate-distortion framework.

The second method involves the use of variable-size blocks, rather than a triangular mesh. Once again, a frame is first segmented into regions of homogeneous motion, which are then approximated with polygons. A grid of blocks is created in each region, with the block size inversely proportional to the motion compensation error for that region. This ensures that regions with complex motion are populated by smaller blocks. Following this, bi-directional translational and affine motion parameters are estimated for each block. In contrast to the mesh-based approach, this method allows the number of blocks to be easily controlled. Nevertheless, the number and shape of regions remains very sensitive to the segmentation parameters used.

The third technique also uses variable size blocks, but the spatio-temporal segmentation stage is replaced with a simpler and more robust binary block partitioning process. If a particular block does not allow for accurate motion compensation, then it is split into two using the horizontal or vertical line that achieves the maximum reduction in motion compensation error. Starting with the entire frame as one block, the splitting process is repeated until a large enough binary tree of blocks is obtained. This method causes partitioning to occur along motion boundaries, thus substantially reducing blocking artifacts compared to regular block matching. In addition, small blocks are placed in regions of complex motion, while large blocks cover areas of uniform motion. The proposed technique provides significant gains in picture quality when compared to fixed size block matching at the same total rate.

The binary partition tree method has been integrated into a hybrid video codec. (The codec also has the option of using fixed-size blocks or H.264/AVC variable-size blocks.) Results indicate that the binary partition tree method of motion compensation leads to improved rate-distortion performance over the state-of-the-art H.264/AVC variable-size block matching. This advantage is most evident at low bit-rates, and also in the case of bi-directionally predicted frames.

Keywords: motion estimation, motion compensation, video coding, video compression, content-based, variable-size block matching, binary partition tree

e-mail: servais@ieee.org

web: http://www.ee.surrey.ac.uk/CVSSP/VMRG/hdtv/

Acknowledgements

For this I will extol you, O Lord, among the nations,
and sing praises to your name. *PSALM 18:49*

I would like to thank my supervisors, Dr Theo Vlachos and Dr Thomas Davies for their guidance and encouragement throughout the project. Their extensive and up-to-date experience in the field of video coding has been invaluable in the process of exploring and evaluating new ideas. Many thanks to both of you for sharing your insight and understanding!

I am also extremely grateful to Prof. Josef Kittler for the opportunity of having been able to undertake research at the Centre for Vision, Speech and Signal Processing. Much thanks also goes to all my friends and colleagues at CVSSP for providing a rewarding and exciting research environment, that I have enjoyed being a part of.

This research would not have been possible without the funding offered by BBC Research and Development and the CVSSP. The corporation's generous financial support and the part-time work provided is greatly appreciated.

To my family and friends who have provided so much encouragement over the last four years (and before) – thank you immensely! I truly appreciate your support, friendship and prayers. In particular, thanks to Susan and Paul (for proof-reading many of the chapters) and Nick (for help with binding the thesis and burning the DVDs). A special word of gratitude goes to my parents for the many ways in which they have contributed to my education over the years.

To Susan – it's been fantastic having a wife who's also a fellow PhD student to share this experience with. Thank you so much for your enthusiasm, patience, and selfless help, particularly during the last few months. I hope you've enjoyed learning about video compression as much as I've enjoyed finding out about weirs and fish!

Finally, a special word of thanks goes to Rebecca — thanks for arriving two weeks late... that extra fortnight of normal sleep really helped!

Glossary

AC	The varying (non-DC) component of a signal
AVC	Advanced Video Coding
B (frame)	Bi-directionally predicted (inter) frame
bpp	Bits Per Pixel
CIF	Common Intermediate Format
CPU	Central Processing Unit
dB	Decibels
DC	The average component of a signal
DCT	Discrete Cosine Transform
DFD	Displaced Frame Difference
FSBM	Fixed-Size Block Matching
HVS	Human Visual System
I (frame)	Intra frame
ISO	International Organisation for Standardisation
ITU	International Telecommunication Union
JPEG	Joint Photographic Experts Group
JSEG	An image segmentation method [16, 17] (not an acronym)
kbps	kilobits per second
MPEG	Motion Picture Experts Group
MSE	Mean Square Error
OBMC	Overlapped Block Motion Compensation
P (frame)	Predicted (inter) frame
PNG	Portable Network Graphics
PSNR	Peak Signal to Noise Ratio
QCIF	Quarter CIF
QP	Quantisation Parameter
SIF	Source Input Format
SSE	Sum of Squared Error
VOP	Video Object Plane
VSBM	Variable-Size Block Matching

Contents

1 Introduction **1**

1.1 Background 1

1.2 Research Aims 2

1.3 Dissertation Structure 3

1.4 Contributions 4

1.5 Publications 6

2 *Setting the Scene*: The Basics of Video Coding **7**

2.1 Introduction 7

2.2 Displaying Digital Video 8

2.2.1 Display Formats 8

2.2.2 The Human Visual System 10

2.2.3 Colour Spaces 12

2.2.4 Measuring Picture Quality 13

2.3 Entropy Coding 14

2.3.1 Huffman Coding 15

2.3.2 Arithmetic Coding 15

2.4 Transform Coding 19

2.4.1 The Discrete Cosine Transform 19

2.4.2 The Wavelet Transform 21

2.4.3 Matching Pursuit 22

2.5 Block-based Motion Compensation 24

2.5.1 Block Matching 24

2.5.2 Variable-Size Block Matching 26

2.5.3 Overlapped Block Motion Compensation 26
2.6 Rate-Distortion Optimisation 28
2.7 Image and Video Compression Standards 31
2.7.1 JPEG 32
2.7.2 JPEG-2000 33
2.7.3 H.261 34
2.7.4 MPEG-1 35
2.7.5 MPEG-2 36
2.7.6 H.263 36
2.7.7 MPEG-4 37
2.7.8 H.264 / MPEG-4 AVC 37

3 Region Shape Coding 39
3.1 Introduction 39
3.2 Related Work 40
3.3 Coarse Polygon Approximation 43
3.3.1 Finding the Initial Polygon Segments 43
3.3.2 Coding the Segment Graph 45
3.4 Progressive Refinement 47
3.5 Results 49
3.6 Conclusion 53

4 Motion Compensation using a Triangular Mesh 55
4.1 Introduction 55
4.2 Related Work 56
4.2.1 Segmentation 56
4.2.2 Motion Estimation 58
4.2.3 Mesh Generation 60
4.3 Mesh Generation 62
4.3.1 Spatio-Temporal Segmentation 62
4.3.2 Polygon Approximation 65
4.3.3 Triangulation 66

4.4 Motion Estimation 68
4.4.1 Translational Motion 68
4.4.2 Affine Motion 70
4.5 Results 73
4.6 Conclusion 79

5 Motion Compensation using Region-Based Block Matching 81
5.1 Introduction 81
5.2 Related Work 82
5.3 Region Segmentation 84
5.3.1 Initial Segmentation and Error Surfaces 84
5.3.2 Region Splitting 85
5.3.3 Region Merging 86
5.4 Variable-Size Block Generation 87
5.5 Motion Estimation 90
5.6 Results 93
5.7 Conclusion 97

6 Motion Compensation using a Binary Partition Tree of Blocks 99
6.1 Introduction 99
6.2 Related Work 100
6.2.1 Fixed-Size Block Matching 100
6.2.2 Variable-Size Block Matching 101
6.3 Motion Compensation Error Surfaces 104
6.4 Block Partitioning 105
6.5 Generating a Binary Partition Tree 108
6.5.1 Growing the Tree 108
6.5.2 Pruning the Tree 110
6.5.3 Coding the Tree 112
6.6 Motion Estimation 117
6.7 Results 117
6.8 Conclusion 122

7 Integration into a Hybrid Video Codec 125

7.1 Introduction . . . 125

7.2 An Overview of the Codec Structure . . . 126

7.2.1 Picture Types and Grouping . . . 126

7.2.2 Codec Input Parameters and File Formats . . . 128

7.2.3 Rate and Quality Control . . . 130

7.2.4 Entropy Coding . . . 134

7.3 Intra-Frame Coding . . . 134

7.4 Inter-Frame Coding . . . 135

7.4.1 Block Partitioning and Motion Estimation . . . 135

7.4.2 Block Mode Selection . . . 145

7.4.3 Coding Block Motion Information . . . 147

7.4.4 Overlapped Block Motion Compensation . . . 151

7.4.5 Residual Coding . . . 152

7.5 Results and Analysis . . . 153

7.5.1 Rate-Distortion Performance . . . 154

7.5.2 Frame-by-Frame Results . . . 158

7.5.3 Bit-Stream Analysis . . . 163

7.5.4 Subjective Performance . . . 165

7.6 Conclusion . . . 179

8 Conclusion 181

8.1 Summary . . . 181

8.2 Contributions . . . 184

8.3 Future Work . . . 185

A The RGB to YC_bC_r Colour Transform 187

B The use of Binarisation for Modelling and Coding 189

C The 1D Wavelet Transform 193

Chapter 1

Introduction

Things in motion sooner catch the eye
Than what stirs not.

Troilus and Cressida
WILLIAM SHAKESPEARE

1.1 Background

Motion estimation and compensation are techniques that are used frequently within the fields of image and video processing. *Motion estimation* describes the process of determining the motion between two or more frames in an image sequence. *Motion compensation* refers to the technique of predicting and reconstructing a frame using a given reference frame and a set of motion parameters. Although they describe two distinct processes, motion compensation can only be performed once an estimate of motion is available.

Motion compensation is perhaps most widely used in the field of video compression, since frames are often predicted from nearby frames in the sequence. However, motion estimation and/or compensation have a variety of other important applications, such as: spatio-temporal segmentation, scene cut detection, frame rate conversion, de-interlacing, object tracking, etc.

There are several different approaches to estimating the motion present within a scene. In general though, these operate using fixed methods that are neither dependent on (nor adaptive to) scene content. Content-based methods attempt to perform some analysis of a scene prior to determining how the motion estimation should be performed. For example, pre-processing might be used to try and decompose a scene into its constituent moving objects, so that motion estimation can be used to determine the motion properties of each region.

As mentioned above, motion estimation and compensation are used in video compression. This (i.e. video compression) is the process of representing raw video data in an efficient way by exploiting any redundancy that may be present.[1] In practice, compression/coding is achieved using a video codec[2], which is an implementation of a particular video coding algorithm. Most codecs allow some degradation in picture quality in order to achieve a greater degree of compression.

1.2 Research Aims

The PhD research project described in this dissertation consists of two phases. During the first stage, the aim was to investigate existing methods of content-based motion estimation and compensation, followed by the design of alternative content-based techniques. It was envisaged that this would involve decomposing a scene into its constituent objects, followed by some form of object-based motion estimation and compensation.

For the second phase of the project, the aim was to integrate any promising content-based motion compensation methods (from the first phase) into a video codec. This would help in determining the effectiveness of the chosen content-based methods, by allowing them to be compared to traditional block-matching

[1] The term *video coding* is often used synonymously for *video compression*. It refers to the process of representing (usually compressed) video data for storage or transmission.

[2] The term *codec* is short for **co**mpression/**de**compression (or **co**ding/**de**coding).

approaches to motion compensation. In particular, one of the goals was to investigate the performance of the selected content-based methods over a range of bit-rates.

1.3 Dissertation Structure

This remainder of this dissertation is divided into seven chapters, the contents of which are summarised below:

Chapter 2 provides an introduction to the building blocks of image and video compression, including motion estimation and compensation. A review of coding standards is also provided.

Chapter 3 presents a method for coding the shape of multiple regions within an image. It describes how regions can be approximated using polygons, which are then coded in a progressive way (i.e. starting with a coarse representation that is gradually refined). Efficient region coding is clearly important in any object-based video compression scheme, since there is an overhead associated with representing the shape of each region.

Chapter 4 describes the first of three methods of content-based motion compensation. Frames in a sequence are segmented into regions with similar motion characteristics, allowing for the creation of a content-based triangular mesh. Triangles in the mesh are then used when performing motion estimation and compensation.

Chapter 5 proposes another region-based motion compensation technique. Once again, frames are divided into regions with different motion characteristics. Variable-size, square blocks are then generated within each region — with small blocks in areas of complex motion and larger blocks in regions dominated by uniform motion.

Chapter 6 considers a content-based motion compensation technique that attempts to combine the block generation and segmentation processes. Starting with one large block that spans the frame, blocks are repeatedly split in two using the straight line that minimises the motion compensation error for the two halves. This results in the creation of a binary tree of blocks, with leaf nodes used for performing motion compensation.

Chapter 7 describes the design of a hybrid video codec, in which motion compensation is performed using both fixed-size and variable-size blocks. The codec provides a framework for comparing the Binary Partition Tree method of block matching to existing block-based techniques. The rate distortion behaviour of the codec is analysed and discussed, based on the results obtained for four test sequences.

Chapter 8 concludes the thesis by summarising the research performed and the results obtained. In addition, the novel contributions of this work are highlighted, and some ideas for further work are also suggested.

1.4 Contributions

The research described in this dissertation provides a number of novel contributions, which are summarised below:

- Shape coding is generally applied to individual objects (as discussed in Section 3.2), rather than a collection of regions in the form of a segmentation map. While vertex-based methods for coding segmentation maps do exist [48, 80, 73], the progressive polygon approximation method described in Chapter 3 is believed to be novel, in that it allows for an embedded bit-stream to be produced. A coarser representation of a coded segmentation map can then be obtained by decoding only the initial portion of the bit-stream.

- The triangular mesh-based motion compensation method described in Chapter 4 uses a variety of existing techniques when performing segmentation, triangulation and motion estimation. Thus, although none of the components of the system are novel, it combines them in a new way into a chain of processes.

- Most implementations of Variable-Size Block Matching (VSBM) tend to use either a binary-tree or quad-tree approach to generating variable-size blocks. The method of varying the block size by region, as described in Chapter 5, is believed to be novel. This approach operates by making the number of blocks in a region proportional to the motion compensation error for that region.

- Much of this Binary Partition Tree VSBM technique presented in Chapter 6 is believed to be novel — in particular the method of partitioning blocks using the (horizontal/vertical) straight line that minimises motion compensation error. The way in which the partition tree structure is coded also appears to be original.

- The hybrid video codec described in Chapter 7 employs a variety of known components and techniques, which are combined in a novel way. Most significantly, the codec allows for the comparison of four block-based motion compensation techniques. These include the method of VSBM used in the state-of-the-art H.264/AVC [39] video codec, as well as the Binary Partition Tree VSBM approach presented in Chapter 6.

The two region-based methods (discussed in Chapters 4 and 5) demonstrate some new ideas and show some definite promise, yet their performance was not considered to be consistent enough for integration into a video codec. The Binary Partition Tree VSBM method (presented in Chapter 6) was found to provide some significant advantages over state-of-the-art H.264/AVC VSBM, while at the same time containing a number of original ideas.

1.5 Publications

Some of the research conducted during the project was published in a number of papers, which are listed below:

- M.P. Servais, T. Vlachos, and T. Davies. Bi-Directional, Affine Motion Compensation Using a Content-Based, Non-Connected, Triangular Mesh. In *Proceedings of the IEE European Conference on Visual Media Production (CVMP)*, pages 49-58, London, March 2004.

- M.P. Servais, T. Vlachos, and T. Davies. Progressive Polygon Encoding of Segmentation Maps. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, pages 1121-1124, Singapore, October 2004.

- M.P. Servais, T. Vlachos, and T. Davies. Motion Compensation using Content-based Variable-Size Block-Matching. In *Proceedings of the Picture Coding Symposium (PCS)*, San Francisco, December 2004.

- M.P. Servais, T. Vlachos, and T. Davies. Affine Motion Compensation using a Content-based Mesh. *IEE Proceedings on Vision, Image and Signal Processing*, volume 152, number 4, pages 415-423, August 2005.

- M.P. Servais, T. Vlachos, and T. Davies. Motion-Compensation using Variable-Size Block-Matching with Binary Partition Trees. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, Genova, September 2005.

The reader is also referred to the project website, which contains MATLAB source code (including a demonstration) and details of any additional publications. The address is: `http://www.ee.surrey.ac.uk/CVSSP/VMRG/hdtv/`

Chapter 2

Setting the Scene: The Basics of Video Coding

2.1 Introduction

When pictures are represented in a digital format, they require substantial amounts of storage. For example, a raw digital photograph of medium resolution occupies more storage space than an entire copy of the Bible.[1] And the demands of *video* are many times greater.

Although the capabilities of networks and digital storage devices have increased significantly in recent years, so too have the expectations of users. As a result, the need for effective multimedia compression techniques remains undiminished.

This chapter provides a review of the main components and standards in the field of video compression. It also introduces some concepts related to motion estimation and compensation that are expanded upon in later chapters.

[1] A 24-bit, 2 mega-pixel photo requires 6 MB of space, while the King James version of the Bible can be represented in uncompressed text format using just over 4 MB of data [2].

2.2 Displaying Digital Video

2.2.1 Display Formats

Digital video is used to represent the projection of a three-dimensional (3D) scene onto a two-dimensional (2D) plane (as shown in Figure 2.1). Both the content of the scene and the position of the plane can vary over time, with the latter being true in the case of a moving camera. Although the original scene is continuous in time and space, it needs to be sampled both temporally and spatially in order to be digitised.

Spatial sampling is performed using a 2D rectangular grid of pixels, which taken together form a still image of the scene at a particular point in time. Temporal sampling is then achieved using a sequence of these images (frames) at regular time intervals, as illustrated in Figure 2.2. Certain video formats use *interlacing*, in which the even and odd rows of samples in a frame are captured at slightly different points in time. Interlacing helps to reduce flicker in displayed video, although it can also introduce some undesirable artifacts.

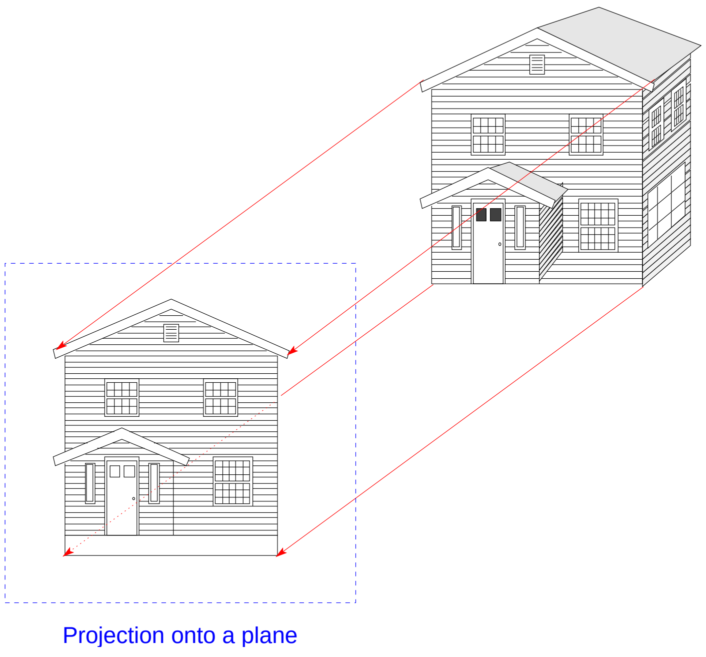


Figure 2.1: Projection of a 3D scene onto a 2D plane

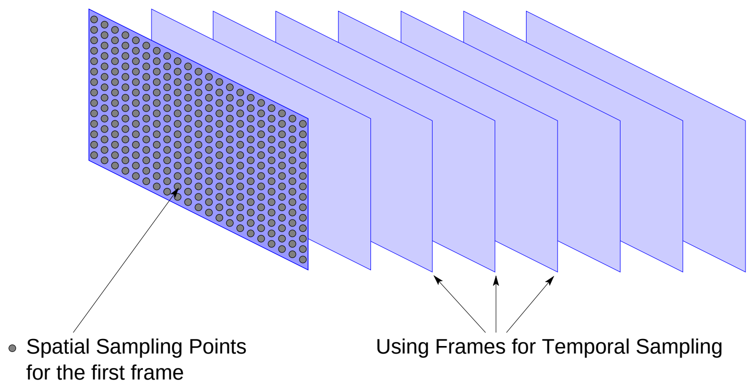


Figure 2.2: Spatial and temporal sampling for digital video

The closer together the samples are (in time and space), the greater the perceived quality of the video is likely to be. However, increasing the number of samples leads to a corresponding growth in the amount of information to store or transmit. As a result, there is a wide range of video display formats available, which vary according to their intended application. Table 2.1 provides a summary of some of the digital formats commonly in use today.

Table 2.1: Some common digital video display formats

Format	**Resolution** (width × height)	**Frame Rate** (fps)	**Frame Type**
Quarter CIF (QCIF)	176 × 144	30	progressive
Source Input Format (SIF)	352 × 240	30	progressive
	352 × 288	25	progressive
Common Intermediate Format (CIF)	352 × 288	30	progressive
ITU Rec. BT.601 (Standard Definition)	720 × 480	30	interlaced
	720 × 576	25	interlaced
ITU Rec. BT.709 (High Definition)	1280 × 720	24,25,30,50,60	progressive
	1920 × 1080	25,30	interlaced
	1920 × 1080	24,25,30	progressive

2.2.2 The Human Visual System

From an observer's point of view, it is not necessary for digital pictures to contain detail that is not perceptible. Consequently, images need only contain information which can be perceived by the human eye, or more precisely the Human Visual System (HVS).[2] Compression systems can take advantage of the characteristics of the HVS by not coding data that is not naturally observable.

A. Spatial Sampling

The sensitivity of the HVS varies according to the spatial frequency of an image. For example, Figure 2.3 varies in both contrast and spatial frequency. Although the change in contrast in this figure is regular across all frequencies, the human observer perceives it as variable, due to the behaviour of the HVS.

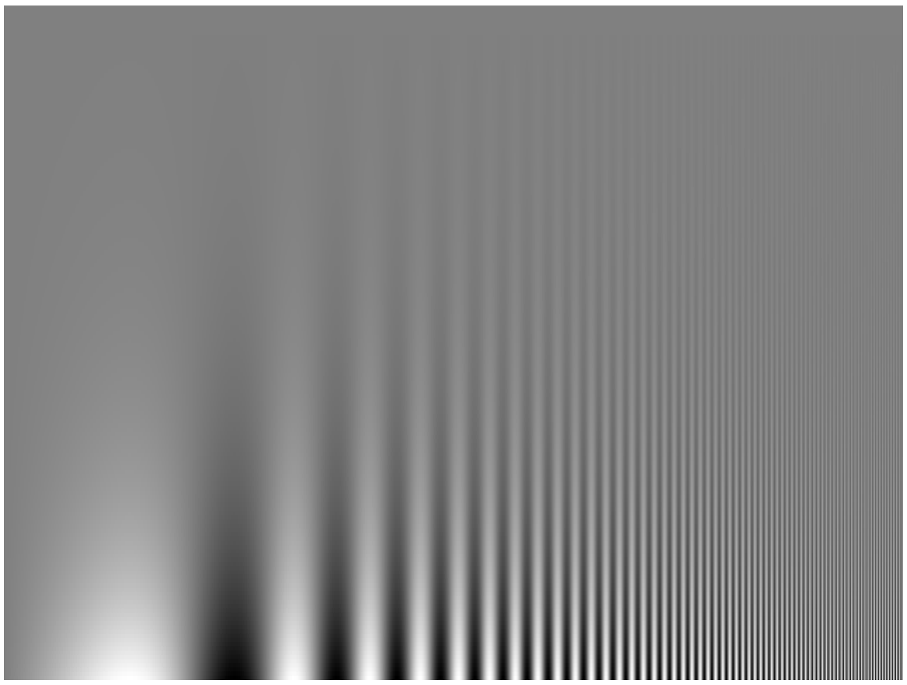


Figure 2.3: An image used to demonstrate the variation of sensitivity to contrast at varying spatial frequencies. Contrast decreases linearly from bottom to top, while spatial frequency increases from left to right.

[2] The HVS consists of the eyes, parts of the brain and the nerve fibres connecting them.

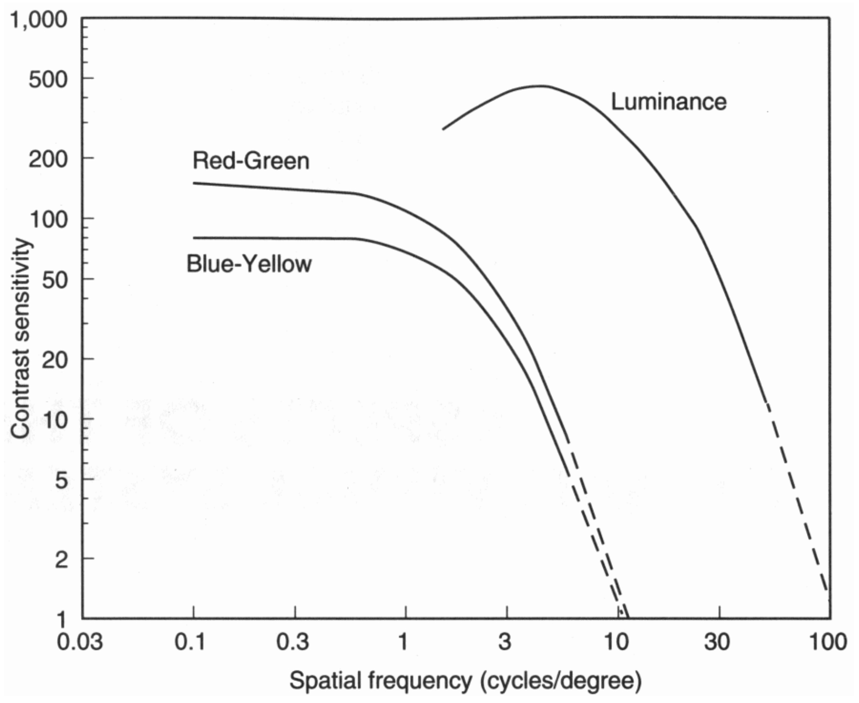


Figure 2.4: The contrast sensitivity function for luminance and for colour differences. Reproduced from Pennebaker and Mitchell [82].

The response of the HVS to changes in intensity is represented by the luminance curve in Figure 2.4.[3] This figure also shows the sensitivity of the eye to coloured stimuli.[4] It can be seen that the HVS is much less able to discern colour differences, and later in the chapter it is shown how this observation can be utilised by compression systems.

B. Sample Precision

When using a digital representation of an image, the value of each sample needs to be quantised using some finite precision. It has been shown experimentally that the HVS can distinguish around 100 different luminance levels [28]. However, the spacing between these levels is not constant, so in practice 8 bits are used per luminance sample.

[3] This is based on measurements performed by Van Nes and Bouman [71].

[4] These are based on experiments by Mullen [66] using red-green and blue-yellow gratings.

C. Temporal Sampling

A video consists of a sequence of images, displayed in rapid succession, to give an appearance of continuous motion. If the time gap between consecutive frames is too large, the viewer will observe jerky motion. From a compression point of view, it is important not to transmit any redundant information. Thus the frame rate of a video should be as low as possible, without causing any significant distortion in perceived motion.

Figure 2.5 shows the contrast sensitivity function of the HVS for a range of spatial and temporal frequencies. From this graph it is evident that the sensitivity of the HVS drops off significantly at high frame rates. In practice, most video formats use temporal sampling rates of 24 frames per second, and above.

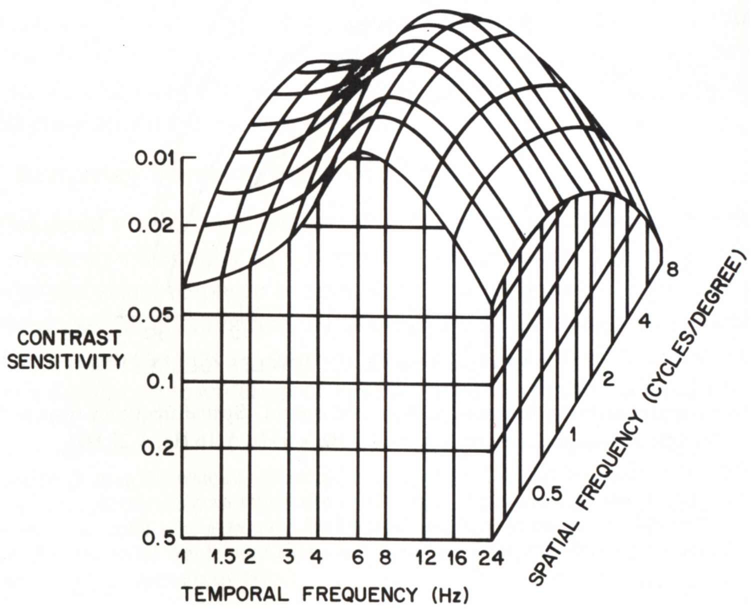


Figure 2.5: Sensitivity to contrast as a function of spatial and temporal frequency. Reproduced from Netravali and Haskell [72].

2.2.3 Colour Spaces

The acquisition and display of video material is usually performed in the *RGB* (red, green and blue) Colour Space. However, most coding and transmission standards convert the data to the YC_bC_r format as an intermediate step. The Y (luma) component is a weighted sum of the R, G, and B channels, while the two chrominance (chroma) components, C_b and C_r, represent the differences between Y and the blue and red channels, respectively.

As was shown in Figure 2.4, the HVS is less sensitive to chrominance, particularly at high spatial frequencies. Many compression systems make use of this fact by low-pass filtering and sub-sampling the chrominance components. Figure 2.6 illustrates two of the sub-sampling profiles commonly used.

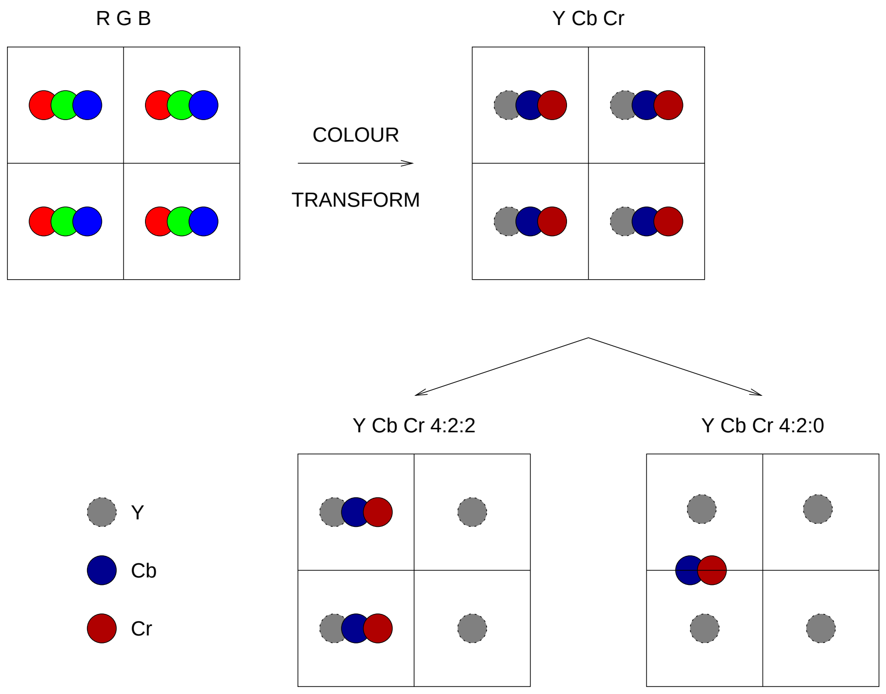


Figure 2.6: Chrominance Subsampling. Each square represents one pixel.

2.2.4 Measuring Picture Quality

When an image or video is compressed using a lossy technique, artifacts are introduced into the scene. The greater the degree of compression, the more severe the distortion is likely to be. It is therefore necessary to be able to measure the quality of a (compressed) picture relative to the original.

Picture quality can be measured either subjectively (using a group of human viewers) or objectively (using an appropriate mathematical formula). The problem with subjective testing is that it is generally a time-consuming and person-intensive task, and thus often impractical. Furthermore, it can be influenced by the viewing environment. On the other hand, objective methods are consistent and easy to implement. However, results obtained from objective testing do not always match the human understanding of picture quality.

The most commonly used objective method of measuring picture quality is to calculate the Peak-Signal to Noise Ratio (PSNR) between the original and a compressed image. For video, PSNR is measured for each frame, and then averaged over the sequence. PSNR is quoted in decibels (dB) and defined as:

$$\text{PSNR} = 10 \log_{10} \left(\frac{{X_{max}}^2}{MSE} \right)$$

where X_{max} is the maximum possible intensity in the image (e.g. 255 for a sample precision of 8 bits), and the Mean Square Error (MSE) is given by:

$$\text{MSE} = \frac{1}{N_C N_R} \sum_{i=1}^{N_C} \sum_{j=1}^{N_R} (X(i,j) - Y(j,j))^2$$

where the number of rows and columns in the image is N_R and N_C respectively. $X(i,j)$ is the intensity of a pixel at position (i,j) in the original picture, while $Y(i,j)$ is the value of the corresponding pixel in the compressed image.

2.3 Entropy Coding

The compression of data can only be achieved when certain symbols are more probable than others. Entropy[5] coding is a generic term for methods used to compress data by assigning each symbol a code that is dependent on the probability of that symbol. The two most common entropy coding methods are Huffman coding and Arithmetic coding.

[5] In the context of Information Theory, entropy is defined as the randomness or information content of a system.

It is important to note that in order for entropy coding to be successful, the encoder and decoder both need to use the same statistical model of the data. Often, they (i.e. the encoder and decoder) start with some initial assumption of statistics that is updated as more and more symbols are encountered.

2.3.1 Huffman Coding

Huffman coding [27] operates by assigning each symbol a code-word with a length that is proportional to the negative logarithm of the symbol's probability. (Note that only integer-length code-words are possible.)

As an example, consider an alphabet comprising the symbols a, b, c and d (with probabilities of 25%, 12.5%, 50% and 12.5% respectively). A Huffman coder will assign these code-words of length 2, 3, 1 and 3 bits respectively. Thus the following binary code-words would constitute a Huffman code: a:10, b:110, c:0 and d:111. These allow for unambiguous decoding and enable the most common symbols to be represented most efficiently. (As is the case in this example, where c requires only one bit.)

2.3.2 Arithmetic Coding

The main problem with Huffman coding is that it requires code-words to be of integer length. As a result, each symbol with a probability that is not a negative power of two, is represented using a code-word with an approximated length. In such a case, Huffman coding operates inefficiently and is not able to achieve the maximum possible compression.

Arithmetic coding [92, 123] avoids this problem by effectively allowing code-words of non-integer length. In arithmetic coding, a sequence of symbols is represented by a real number within the range $[0, 1)$. As successive symbols are encoded, the range becomes narrower, requiring more and more precision to specify the low and high boundaries.

Consider an alphabet of N symbols, i.e. $s_1, s_2, s_3, \ldots s_N$. Let $p(i)$ be the probability of symbol s_i, with the sum of probabilities equal to one. Also, let $C(i)$ represent the cumulative probability of all symbols up to and including s_i in the alphabet, so that:

$$C(i) = \begin{cases} \sum_{k=1}^{k=i} p(k), & \text{for } 1 \leq i \leq N \\ 0, & \text{for } i = 0 \end{cases}$$

The current interval is represented by $[Low, High)$, which includes all points less than $High$ and greater than or equal to Low. To start with, $Low = 0$ and $High = 1$. The encoder then proceeds by adjusting the interval as follows:

Encoding Algorithm:

```
$Low = 0.0$
$High = 1.0$
**while** More symbols to encode **do**
    Input symbol $s_i$
    $Range = High - Low$
    $High = Low + Range \times C(i)$
    $Low = Low + Range \times C(i-1)$
**end while**
Output value $v \in [Low, High)$
```

The process of arithmetic coding is perhaps best explained using an example. Table 2.2 shows a three-symbol alphabet and the associated symbol probabilities, which are assumed to be known prior to encoding or decoding. Figure 2.7 provides an illustration of how the interval rescaling progresses as symbols are

Table 2.2: Probabilities for a three-symbol alphabet

i	Symbol, s_i	Probability, $p(i)$	Cumulative probability, $C(i)$	Initial interval
0	-	-	0	-
1	a	0.3	0.3	[0.0, 0.3)
2	b	0.6	0.9	[0.3, 0.9)
3	c	0.1	1.0	[0.9, 1.0)

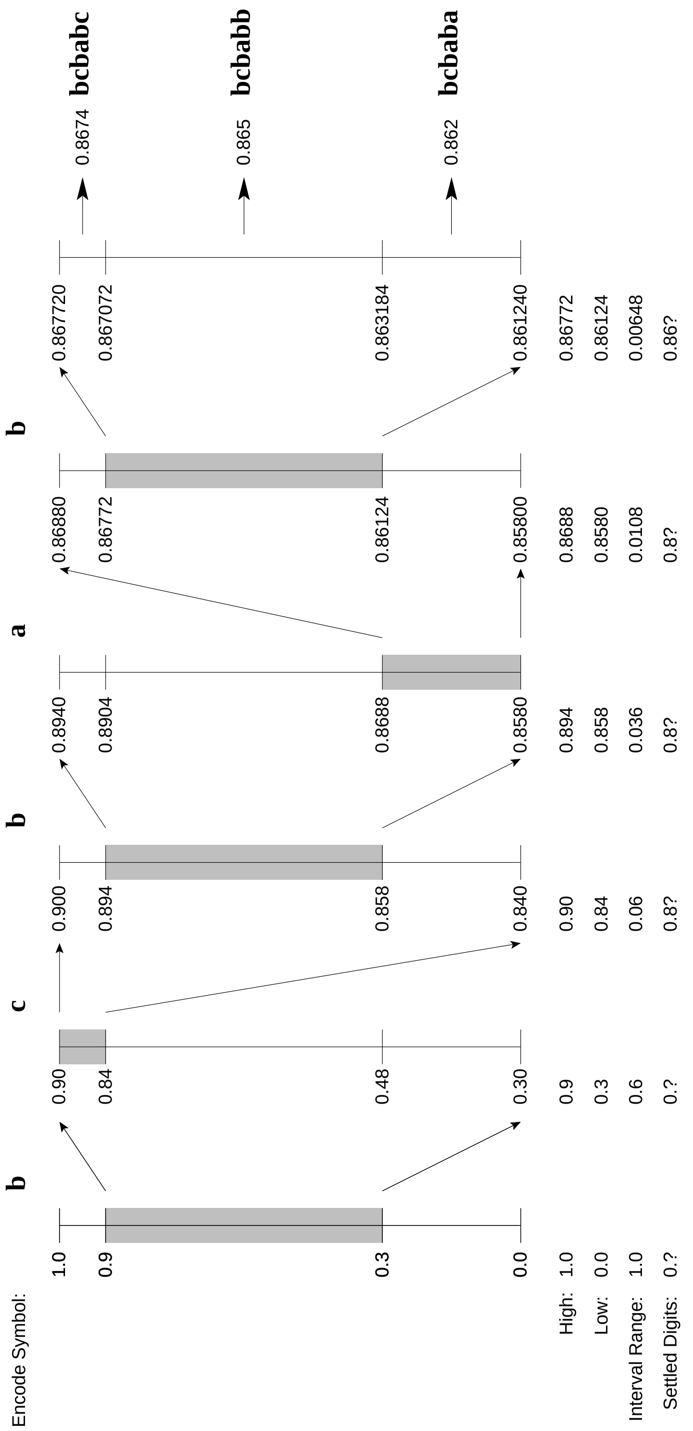


Figure 2.7: An example of arithmetic coding applied to a three-symbol alphabet. The encoding process is shown for three sequences: *bcbaba*, *bcbabb* and *bcbabc*. As more and more symbols are encoded, so the range of $[Low, High)$ shrinks accordingly.

encoded. It can be seen that as *High* and *Low* converge, so their most significant digits equal one another, and thus the corresponding output digits become "settled." The example also demonstrates that the least probable of the three sequences (*bcbabc*) results in a longer coded message than the other two — which is what one would expect from a compression system.

Decoding operates in a similar way to encoding. It is essential that the decoder use the same probability model as the encoder when determining ranges. Probabilities can either be fixed throughout, or vary adaptively based on the previously encoded (decoded) symbols. Given a number v in the range $[0, 1)$, the decoder proceeds as follows:

Decoding Algorithm:

```
$Low = 0.0$
$High = 1.0$
Input value $v \in [0, 1)$
while More symbols to decode do
    $Range = High - Low$
    Find $i$ for which $Low + Range \times C(i-1) \leq v < Low + Range \times C(i)$
    $High = Low + Range \times C(i)$
    $Low = Low + Range \times C(i-1)$
    Output symbol $s_i$
end while
```

Much of the initial work on arithmetic coding involved its application to coding symbols from a binary alphabet, such as bi-level images. More recently, binarisation (i.e. representing non-binary data in a binary format) has come to be used as a relatively simple and effective way of combining context modelling and arithmetic coding [61]. This is discussed in more detail in Appendix B.

A variant of arithmetic coding is *range coding* [62], which uses a large integer range to represent a sequence of numbers, as opposed to a sub-interval within $[0, 1)$. In addition, re-normalisation is performed after every output byte, which marginally reduces the performance of the method, while increasing its speed.

2.4 Transform Coding

Transform coding is an important tool in both image and video compression. It is motivated by the idea that transforming an image to another space may lead to a representation of the image which is easier to compress. This section discusses three transforms that are well-suited to image coding, and considers some of the techniques commonly used to compress the transformed data.

2.4.1 The Discrete Cosine Transform

The Discrete Cosine Transform (DCT) [87] has been used in the field of image compression for more than three decades. It is closely related to the Fourier transform, but produces real-valued coefficients.

The DCT can in theory be applied to a whole image, although this is computationally expensive. In practice, an image is divided into 8×8 blocks, and each block is transformed separately [83]. The equation for a 2D 8×8 DCT is:

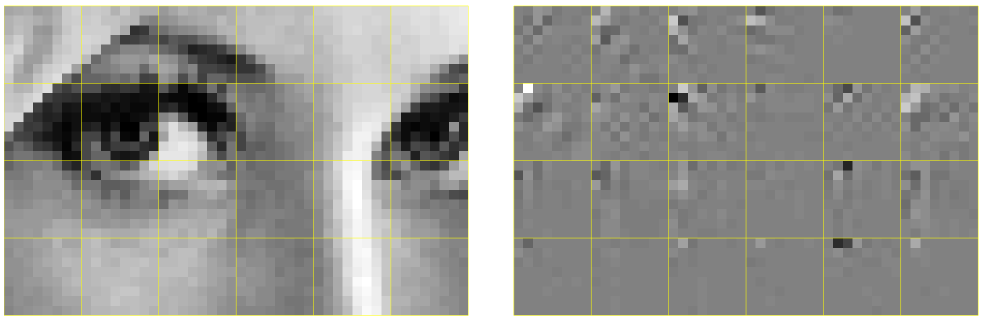
$$S(u,v) = \frac{1}{4} C(u)\ C(v) \sum_{x=0}^{7} \sum_{y=0}^{7} s(x,y)\ cos\left(\frac{(2x+1)u\pi}{16}\right) cos\left(\frac{(2y+1)v\pi}{16}\right)$$

$$\text{where}\ \ C(w) = \begin{cases} 1/\sqrt{2} & \text{for } w = 0 \\ 1 & \text{for } w > 0 \end{cases}$$

The intensity of the image at pixel position (x, y) is given by $s(x, y)$, while $S(u, v)$ represents the DCT coefficient corresponding to horizontal and vertical spatial frequencies, u and v. The *inverse* DCT is then calculated as:

$$s'(x,y) = \frac{1}{4} \sum_{u=0}^{7} C(u) \sum_{v=0}^{7} C(v)\ S(u,v)\ cos\left(\frac{(2x+1)u\pi}{16}\right) cos\left(\frac{(2y+1)v\pi}{16}\right)$$

Figure 2.8 provides an example of an image and its equivalent representation in the DCT domain. It is noticeable that those blocks containing areas of high contrast or significant texture in the spatial domain contain a relatively



(a) Pixels in the Spatial Domain (b) Coefficients in the DCT Domain

Figure 2.8: An example of a DCT. The image is partitioned into 8×8 blocks, which are then transformed to blocks of 8×8 DCT coefficients. Note that in this example, the DC (top left) coefficient in each DCT block has been set to zero for illustrative purposes. This is because the magnitude of the DC coefficient in a block is normally many times greater than any of the other 63 AC coefficients.

large number of high-magnitude DCT coefficients. In comparison, those blocks with a fairly smooth spatial profile (such as the bottom row of blocks) can be represented using only a few DCT coefficients.

The DCT has two characteristics that are particularly advantageous for image coding. First, it acts as a de-correlating transform when applied to typical image data. This is useful in that it allows samples to be coded independently of one another.

Secondly, the DCT coefficients all correspond to different spatial frequencies. As a result, each coefficient can be quantised according to the sensitivity of the HVS to the spatial frequency represented by that coefficient. Quantisation in this manner results in many coefficients having values equal to or close to zero, thus allowing substantial compression.

The DCT has been widely used in image and video coding. It is a core component of JPEG [36] and is also used to compress intra (i.e. non-predicted) frames in all of the major video coding standards. H.264/AVC [39] uses an efficient 4×4 integer approximation of the DCT for both intra and residual coding.

2.4.2 The Wavelet Transform

The Wavelet Transform [55, 12] provides a means of describing a signal in terms of both its spatial and frequency content. It allows signals to be represented as a weighted sum of dilated and translated versions of a *mother wavelet* function.

A more detailed description of the transform is provided in Appendix C, which also describes how the Discrete Wavelet Transform (DWT) can be applied to a signal using a series of high-pass and low-pass filters. From an image processing perspective, this filtering approach is perhaps the most intuitive way of understanding the wavelet transform.

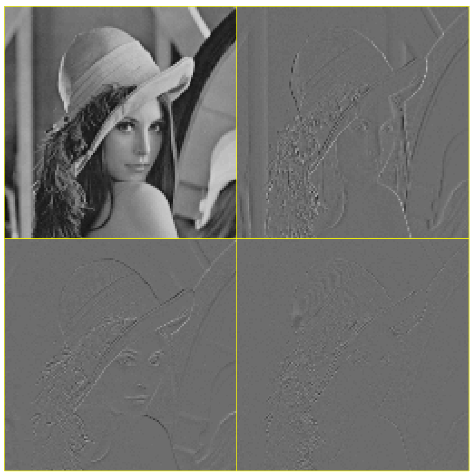
Figure 2.9 provides an example of a DWT applied to an image. To start with, the image is filtered both horizontally and vertically using low-pass and high-pass filters.[6] The output is then sub-sampled by a factor of two. This yields the four sub-bands shown in Figure 2.9(b), which have the following characteristics:

- The LL sub-band (top left) results from horizontal and vertical low-pass filtering, and is therefore a low-pass version of the original image.
- The HH sub-band (bottom right) results from horizontal and vertical high-pass filtering, and thus emphasises diagonal edges.
- The HL sub-band (top right) results from horizontal high-pass and vertical low-pass filtering. It emphasises vertical edges in the original image.
- The LH sub-band (bottom left) results from horizontal low-pass and vertical high-pass filtering. It emphasises horizontal edges.

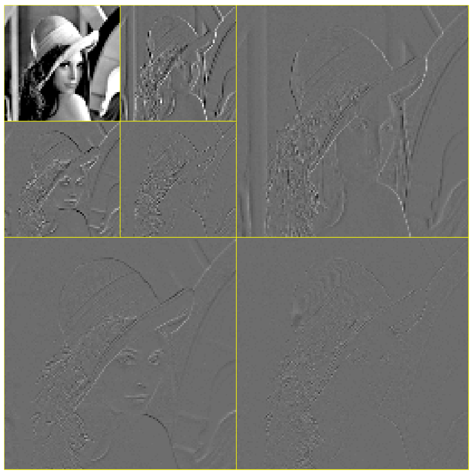
The process is then re-applied to the low-pass sub-band (as shown in Figure 2.9(c)) and repeated as many times as required. As can be seen in this example, many detail (high-pass) coefficients have values close to zero. This allows for good compression, particularly after quantisation. The wavelet transform is used in the JPEG 2000 [34] standard, which is discussed later in the chapter.

[6] The filter taps for a DWT depend on the particular choice of mother wavelet.

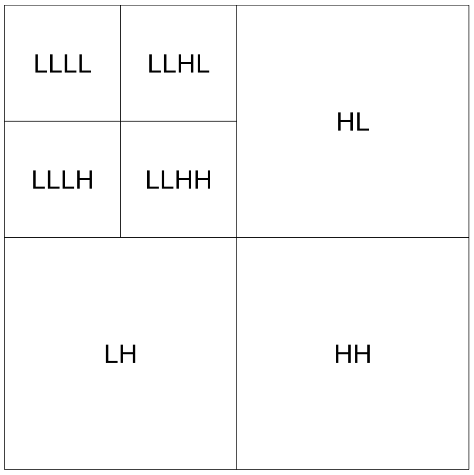
(a) Original Image



(b) After first level of decomposition



(c) After second level of decomposition



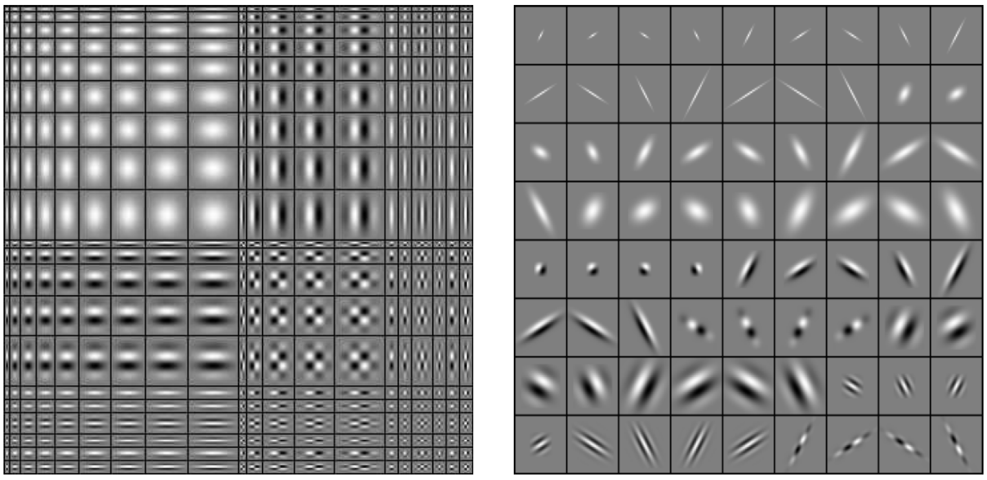
(d) Wavelet sub-bands

Figure 2.9: An example of a two-level, 2D wavelet transform

2.4.3 Matching Pursuit

The DCT and the DWT are commonly used in video coding to compress both images and residual (difference) images.[7] Although most DCT and DWT based coding techniques have been designed for natural images, they have been found to perform reasonably well in the coding of residual images [90].

[7] Residual images arise when a frame is predicted (imperfectly) from another frame. As a result, there is a prediction error between the original and predicted frame.



(a) A 20 × 20 Gabor dictionary (b) Basis functions at various orientations

Figure 2.10: An example of a 2D Matching Pursuit dictionary

In recent years, *Matching Pursuit* has gained in popularity as a method for efficiently representing residual information. Matching Pursuit is a technique that was first proposed by Mallat and Zhang [56]. It allows for a signal to be approximated using a redundant dictionary of functions. The matching pursuit algorithm does not specify a single dictionary of functions, but it is generally recommended to use a set of basis functions that match the characteristics of the signals to be approximated. One of the most popular dictionaries is a set of Gabor basis functions [13], an example of which is illustrated in Figure 2.10.

Using a dictionary of basis functions at various scales and orientations (known to both the encoder and decoder), an image (natural or residual) can then be approximated as follows:

- Perform matching between the image and *atoms* derived from the dictionary. (In Matching Pursuit, an atom is a basis function that has been translated and multiplied by a weighting factor.)

- Find the atom that minimises the total image energy when it (the atom) is subtracted from the image. Note that atoms can be translated to any location within the image, and multiplied by an appropriate weight.

- The index of this atom's basis function within the dictionary is coded, along with its translation vector and weighting factor.
- The (translated and weighted) atom is subtracted from the image, and the process is repeated. This continues until either a target number of atoms have been coded, or the image energy reaches an acceptable threshold.

Residual images tend to have most values close to zero, with very localised signals (such as along the edges of moving objects). This type of content is well suited to coding using a Matching Pursuit technique, because it allows atoms to be placed where they provide the greatest reduction in error. As a result, Matching Pursuit performs particularly well at low bit-rates, since it requires relatively few bits to approximate the largest error signals [68, 69, 70].

2.5 Block-based Motion Compensation

There are a number of different methods of estimating the motion between frames in a video sequence. The most common technique involves dividing a frame into many small blocks, and then performing motion estimation within each block. This section discusses some aspects of block-based motion estimation and compensation.

2.5.1 Block Matching

Block matching [41] is used to estimate the motion between the current frame and a reference frame. To start with, the current frame is divided into many small square blocks of equal size. (Blocks of size 16×16 are commonly used). For each block, matching is performed as outlined below (and illustrated in Figure 2.11):

- Let b_i be a block in the current frame, $F_C(x, y)$ the value of a pixel at position (x, y) in the current frame and $F_R(x, y)$ the value of a pixel at position (x, y) in the reference frame.

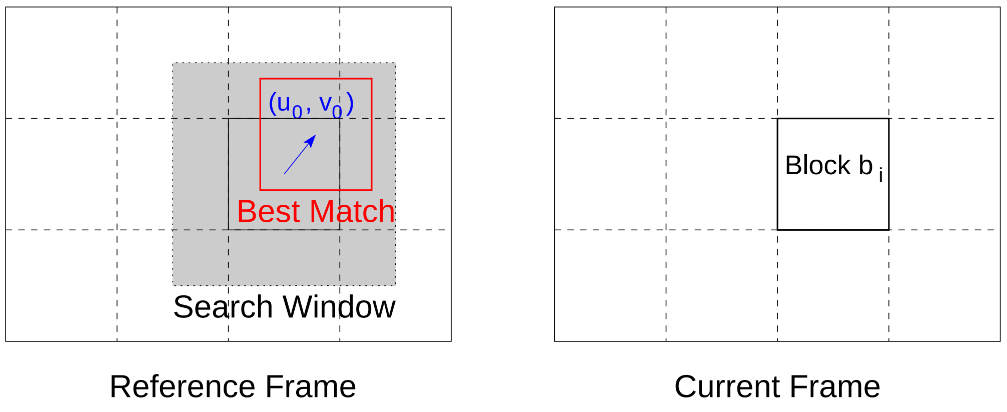


Figure 2.11: Block matching

- Loop through all possible (pixel-accurate) motion vectors, (u, v), within some search window. For each motion vector, calculate the resulting distortion (after motion compensation).

- The distortion metric, $D_{b_i}(u, v)$, is evaluated by comparing pixels in block b_i to a translated block in the reference frame. Two such metrics are commonly used — either the Sum of Squared Error (SSE)[8],

$$D_{b_i}(u, v) = \sum_{(x,y)\in b_i} (F_C(x, y) - F_R(x + u, y + v))^2$$

or the Sum of Absolute Error (SAE)[9],

$$D_{b_i}(u, v) = \sum_{(x,y)\in b_i} |F_C(x, y) - F_R(x + u, y + v)|$$

The former metric (SSE) is slightly more accurate (since it matches the PSNR measure of quality), although the latter method (SAE) is often used because it is faster to compute.

- The chosen motion vector, (u_0, v_0) is the one that yields the minimum distortion for block b_i.

Because block matching is computationally expensive, it is generally performed using only the luma component. In addition, much research has been done on

[8] The SSE is sometimes referred to as the Sum of Squared Difference (SSD).

[9] The SAE is sometimes referred to as the Sum of Absolute Difference (SAD).

developing fast methods of block matching. This is typically achieved by only searching a subset of possible motion vectors in order to find the optimal one.

The method outlined above describes how motion vectors can be calculated to pixel accuracy. However, using motion vectors with sub-pixel accuracy can allow for a significant reduction in matching error. In order to do this, pixel values at non-integer positions can be estimated by using bi-linear interpolation from the surrounding pixels.

2.5.2 Variable-Size Block Matching

In general, block matching performs reasonably well. However, if there is more than one type of motion present in a block, then it cannot be represented accurately using only one motion vector. Variable Size Block Matching (VSBM) [9, 86, 111] provides a way of obtaining more accurate motion vectors in the case where a block straddles one or more motion boundaries.

There are a number of alternative implementations of VSBM. Typically, they operate by allowing a block with a large distortion metric (after motion compensation) to be split into two or four child blocks. In some implementations, these children can themselves be further partitioned if the motion compensation error remains unacceptably large. One example of VSBM is the *advanced prediction mode* of the H.263+ [11, 38] video coding standard. This has the option of allowing each 16×16 macro-block to be motion compensated using either one 16×16 block or four 8×8 blocks.

In video coding, there is clearly a trade-off between using more blocks in order to minimise error, and having to code the resulting additional motion vectors. This is discussed further in the review of VSBM provided in Section 6.2.2.

2.5.3 Overlapped Block Motion Compensation

When performing motion compensation of blocks, each reconstructed block is simply copied from the reference frame using the position specified by its motion vector. (If the motion vector has a non-integer value, then the pixel values are interpolated accordingly.)

When neighbouring blocks have different motion vectors, motion compensating these blocks can result in discontinuities being introduced along their common boundary. Such discontinuities may be intentional (e.g. if the block boundary coincides with a motion vector boundary), however, they often introduce distortion into a motion compensated frame. This type of blocking artifact is common to many block-based video codecs.

De-blocking filters [53] are often combined with the motion compensation process in order to reduce such blocking artifacts. They operate by performing low-pass filtering perpendicular to spatial edges, in order to smooth the block boundaries. Another approach is to use overlapping blocks when performing motion compensation. The latter approach is described below in more detail, together with an example.[10]

Overlapped Block Motion Compensation (OBMC) [76] operates by extending the influence of each block's motion vector beyond the block boundary. In doing this, it helps to provide a more gradual transition at block edges, thus reducing blocking artifacts. A raised cosine filter is typically used to provide a weighting factor for each block in the vicinity of its boundary. At every pixel location, the block weights need to be normalised to sum to unity.

For a pixel that lies in the sphere of influence of n blocks, the value of the motion compensated pixel is calculated n times (using the n blocks' motion vectors). The pixel is then set to the weighted sum of these n values.

Figure 2.12 demonstrates the advantage of using overlapping blocks when performing motion compensation. Block boundaries are clearly present around the eyes and mouth in Figure 2.12(b). When exactly the same set of motion vectors are used with OBMC enabled, there is a significant reduction in distortion, as can be seen in Figure 2.12(c). In this example (which is based on an overlap of

[10] Overlapping blocks are used in the video codec that is presented in Chapter 7.

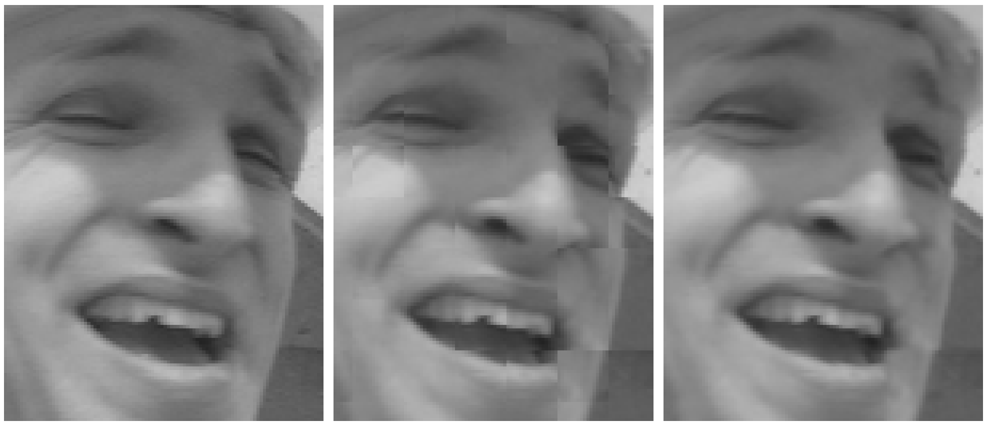


Figure 2.12: Frame 13 of *Foreman*, motion compensated from frames 12 and 14 using 16×16 blocks. (Only the central part of the frame is shown.)

two pixels either side of the block boundary), the use of OBMC increases the PSNR from 34.2 to 34.6 dB.

2.6 Rate-Distortion Optimisation

When performing lossy compression of images or video there is a trade-off between the size of the coded representation (i.e. the bit-rate or rate) and the degradation in quality (i.e. the distortion). As the rate increases, so the distortion diminishes. However, the relationship is variable and is dependent upon the scene content as well as the compression algorithm.

At each point on a rate-distortion graph, the tangent to the curve represents the change in distortion per additional bit. To start with, the absolute value of this tangent is large, but it decreases in magnitude as the number of bits increases.[11] This can be seen from the convex nature of rate-distortion plots, such as the one in Figure 2.13. The "optimum" point on this curve depends on the relative importance of rate and distortion, and may also be subject to

[11] For example, increasing the size of a JPEG image from 200 to 300 bytes provides a greater reduction in distortion than increasing the file size from 1200 to 1300 bytes.

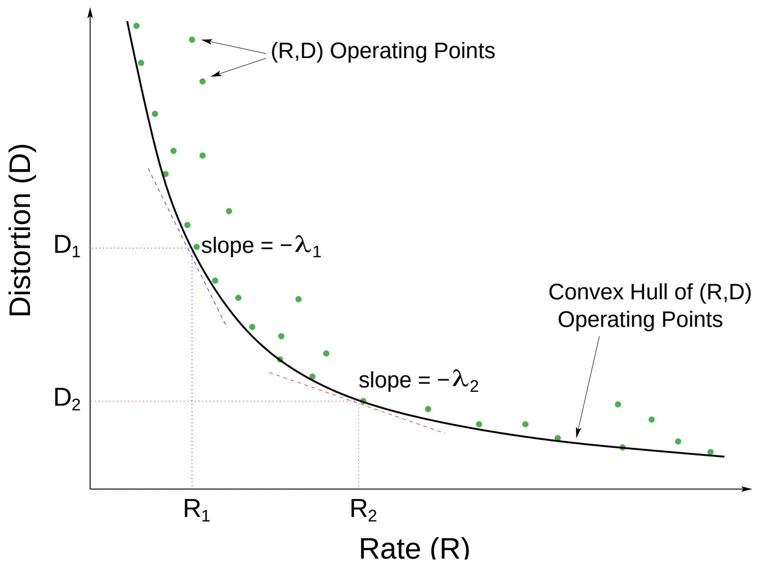


Figure 2.13: An example of a typical rate-distortion graph. For large values of λ (such as λ_1) the priority is to minimise the rate. As λ decreases (to λ_2, for example), it becomes more important to reduce the distortion.

other constraints such as a maximum bit-rate.

Many image and video coding algorithms use several stages in order to achieve compression. In addition, they may operate on a number of different parts of a scene (e.g. frames in a sequence or blocks in an image). In such a case it may not be immediately obvious how to allocate resources (i.e. bits from the total bit budget) among each component.

Using (DCT-based) JPEG [36] as example, one approach might be to code each 8×8 block using the same number of bits, or alternatively to code each block using the same quantisation factor. The former method amounts to coding all blocks at the same rate, while the latter method corresponds to coding blocks at roughly the same quality.

It is not immediately obvious which of these two methods is preferable, since this depends on the intended application. Ideally, each component (block) should be coded in a way that allows a compromise between the opposing goals of minimising both rate and distortion.

Rate-distortion theory [77, 106] provides a way of coding multiple (independent) components that allows the overall performance to be optimised. The essential concept is that each component should be coded using the same rate-distortion trade-off slope. If this were not the case,[12] it would be possible to re-allocate n bits from one component to another in order to achieve lower overall distortion for the same total rate.

Lagrangian Rate-Distortion Optimisation

Measuring the slope of a rate-distortion plot at a particular point can be difficult, especially if only a few points are known. A popular method of improving rate-distortion performance is to use Lagrangian optimisation [77, 113]. For some specified value, λ, a function (J) of both rate (R) and distortion (D) is created:

$$J = D + \lambda R$$

The optimal operating point is then determined by finding the (R, D) pair that minimises J. Clearly, this is dependent on the value of λ, which effectively acts as a relative weighting of rate and distortion. For large λ, it becomes more important to minimise the rate, while for small λ, the emphasis is on achieving low distortion. The optimal (R, D) point for a specified λ has a gradient of $-\lambda$, as shown in Figure 2.13.

Lagrangian rate-distortion optimisation helps when selecting the best of a set of (R, D) points. However, it is still necessary to determine these points by performing the coding process using a variety of codec parameters. This is a computationally expensive approach, and so R and D are usually estimated by the encoder. The estimation process can be difficult, and it often requires that the codec be tested on a range of typical source material in order to establish a rate-distortion model [77, 113].

[12] i.e. if some components were not operating at the same rate-distortion slope

Rate-Distortion Optimisation with Inter-Dependent Components

The rate-distortion optimisation techniques described above assume that each component is processed separately (as in the case of 8×8 blocks in JPEG). In practice, though, rate-distortion optimisation methods do not apply in the same way when there is a chain of dependencies present in a codec.

As an example, consider a video codec in which a frame is coded and then used as a reference in order to predict the following frame. In such a scenario, it is not optimal to code the reference and predicted frames using the same λ value. This is because each bit that is used to reduce the distortion of the reference frame also contributes towards increasing the quality of the predicted frame.

The rate-distortion relationships between different frame types are complex, since they depend on scene content and the type of motion present. This is particularly so for bi-directional prediction, which is used in most video codecs. In practice, inter-dependencies between components are either approximated or ignored when incorporating rate-distortion optimisation into video codecs [113].

Complexity

In the majority of practical codecs, there is also a trade-off involving the computational complexity of the compression method(s). This is because the best coding techniques (in terms of rate and distortion) are often also the ones requiring the most computation. Since many applications require real-time encoding or decoding, this can be a significant constraint. Minimising computational complexity was considered to be beyond the scope of this project. Nevertheless, it is an important aspect of codec design and remains an area of active research.

2.7 Image and Video Compression Standards

During the last two decades, several standards have emerged for the coding of digital images and video. This section provides a brief review of some of the

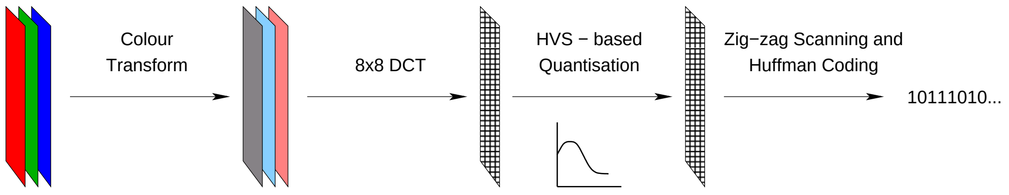


Figure 2.14: The (baseline) JPEG encoding process

most significant ones, and highlights advances from one standard to the next.

2.7.1 JPEG

The Joint Photographic Experts Group (JPEG) was founded in 1986, with the task of developing a standard for the coding of grey-scale and colour images. The JPEG [36, 83] compression standard gained widespread use, particularly on the World Wide Web. JPEG is based on the following core components (illustrated in Figure 2.14):

- The input (colour) image is transformed from the RGB colour space to one where the colour components are significantly less correlated. A popular choice is the YC_bC_r colour space, with one luma and two chrominance components. This is often followed by a sub-sampling of the two chrominance components.
- Regions of pixels are then grouped into 8×8 blocks, and the Discrete Cosine Transform (DCT) [87] is applied to each block. Each DCT coefficient represents the information present at a particular spatial frequency within a block.
- This is followed by the quantisation of the DCT coefficients in each block. As a result of quantisation, the coefficients are represented more coarsely, resulting in some loss of information. However, not all coefficients are quantised to the same degree. For example, the Human Visual System (HVS) is less sensitive to high spatial frequencies, so the corresponding

coefficients are quantised more coarsely, without significantly affecting the image quality from the point of view of a human observer.

- Following this, the quantised coefficients in each 8×8 block are re-ordered in a *zig-zag* fashion so that the low frequency components are listed first, and finally the high spatial frequency coefficients. Many high frequency coefficients are quantised to zero, resulting in long runs of zero-valued coefficients. This is effectively exploited by the use of run-length coding.
- Finally, Entropy Coding (often in the form of Huffman Coding) is applied to the result, to produce the JPEG representation of the image.

2.7.2 JPEG-2000

Developments in image coding continued throughout the 1990's, and in 1997 the ISO and ITU-T initiated work on the next JPEG standard. JPEG-2000 [34, 115] utilises wavelet / sub-band coding, since the wavelet transform has been shown to offer some significant advantages over the DCT.

As a first step, an image is broken down in to large tiles (typically 128×128 pixels) which are coded separately. This is done to reduce memory requirements. As with its predecessor, JPEG-2000 also performs a RGB to YC_bC_r transform, but this is followed by a Discrete Wavelet Transform (DWT).

The DWT is applied to the tiles, decomposing each one into several sub-bands. Quantisation is performed on a bit-plane level, encoding the more significant coefficients first. In addition, parent-child dependencies between wavelet coefficients are exploited, as shown in Figure 2.15. Arithmetic coding is then applied as a final compression step. The decoding process mirrors the encoding one.

As well as rate-distortion performance improvements over JPEG in the order of 50%, JPEG-2000 supports two notable features: In addition to the standard *lossy* mode, *lossless* compression is also catered for, resulting in typical compression ratios of 2:1 or 3:1. Secondly, it is also possible to define regions of interest within an image. The encoder then allocates a greater portion of

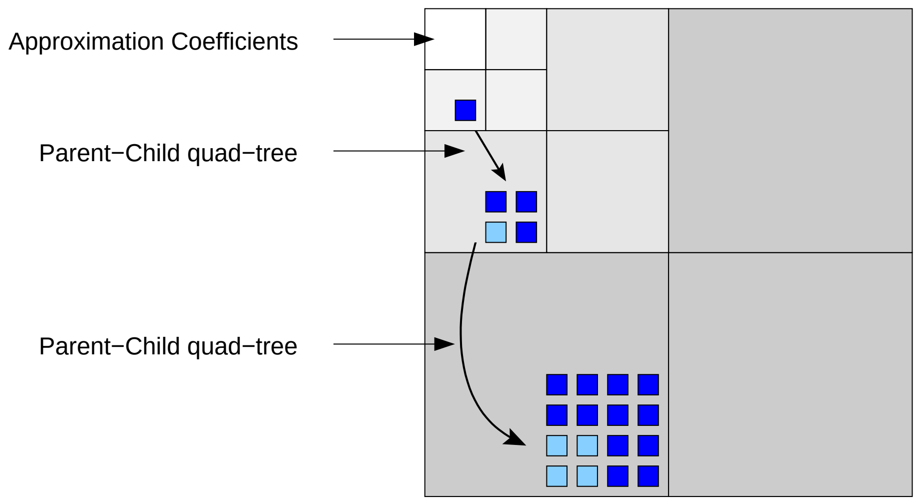


Figure 2.15: A three-level, sub-band decomposition of an image. An example of the parent-child relationship in a quad-tree is also shown. Coefficients that are insignificant (relative to some threshold) tend to have children that are also insignificant (relative to the same threshold).

the bit-stream to these regions, resulting in them being reproduced with higher quality, though at the expense of the remainder of the image.

2.7.3 H.261

Recommendation H.261 [7] was adopted as an international standard by the CCITT (now ITU-T) in 1990. It was designed for use in video-conferencing applications at bit rates which are integer multiples of 64 kbps. Progressive-scan CIF and QCIF formats (with 4:2:0 chrominance sub-sampling) are supported.

H.261 uses the 8×8 DCT, and groups together sets of four luminance and two chrominance blocks into *macro-blocks*. Macro-blocks may be intra-coded (independently of the previous frame) or inter-coded (using motion compensation from the previous frame). During the motion estimation process, motion vectors are calculated to pixel accuracy.

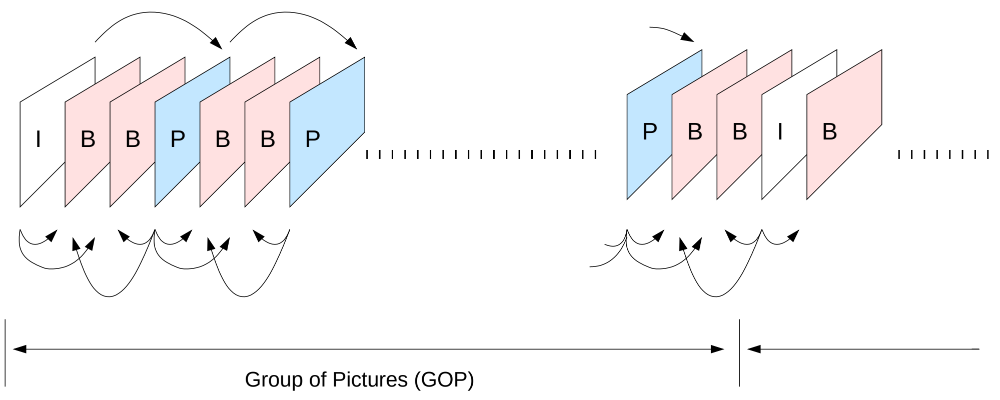


Figure 2.16: The MPEG Group of Pictures (GOP) Structure. The arrows indicate which frames are used to predict others.

2.7.4 MPEG-1

At around the same time, the ISO Moving Picture Experts Group (MPEG) released its first standard, which was designed to provide video (and audio) compression for storage and playback at rates of 1.0 to 1.5 Mbps.[13] In common with H.261, MPEG-1 [31] uses 8 × 8 DCT blocks and a similar macro-block structure.

However, MPEG-1 uses more advanced motion compensation techniques, including bi-directional prediction. Three frame types are possible: Intra-coded (I) pictures are encoded in a similar way to still images in JPEG; Inter-coded (P) pictures are predicted using motion compensation from the preceding I or P picture; and Bi-directionally predicted (B) pictures, which are motion compensated from two reference frames (i.e. the preceding *and* subsequent I or P pictures). Figure 2.16 illustrates the typical structure. Frames are arranged into a *Group of Pictures* (GOP), with each GOP commencing with an I picture. Motion vectors are calculated to half-pixel accuracy.

[13] At the time, the target application was (progressive) VHS-quality video stored on a CD.

2.7.5 MPEG-2

MPEG-2 [32] was developed as the standard for digital (standard definition) television, and it is currently in widespread use. It was designed to handle higher resolutions than MPEG-1, as well as interlaced frames (although progressive video is also supported). MPEG-2 borrows many techniques from MPEG-1, with some modifications to handle interlacing. It supports bit-rates in the range of 2 to 10 Mbps.

MPEG-2 also offers scalability through the use of enhancement layers. Temporal scalability is achieved by encoding the *base layer* at low temporal resolution (i.e. low frame rate), while enhancement layer(s) provide the additional information to decode the data at a higher frame rate. Spatial and rate (PSNR) scalability can be achieved in a similar way. Data partitioning can also be used so that the more critical information (such as headers and motion vectors) can be coded separately.

The MPEG committee commenced work on a alternative standard for HDTV, which was due to be called MPEG-3. However this was abandoned when it became clear that MPEG-2 offered reasonable support for higher resolutions.

2.7.6 H.263

Following advances in video coding, the ITU-T released H.263 [37] as a standard for use in video telephony in 1995. It was targeted at low bit rates (64 kbps and under), but is suitable for higher rates as well. H.263 also offers rate, spatial and temporal scalability in a similar way to MPEG-2.

Subsequent versions of H.263 [38] include several other features and options, such as an advanced prediction mode. This allows each macro-block to use either 8×8 or 16×16 blocks for motion compensation, depending on which method provides a superior rate-distortion performance. In addition both forward and backward motion estimation is possible, at half-pixel accuracy.

2.7.7 MPEG-4

The MPEG-4 [35, 84] standard was developed with the goal of being more than just an incremental improvement on the previous two standards. MPEG-4 supports a wide range of bit-rates, but is mainly focused on low bit-rate video. The first version of the standard provides a very low bit-rate video core, which is actually very similar to baseline H.263 [90].

A fundamental concept in MPEG-4 is the idea of *object-based* coding. This allows a scene to be described in terms of foreground and background objects, which may be coded independently. However, since the standard only defines how the *decoder* should operate, there is no prescribed method for the difficult task of segmenting a scene into its constituent objects. This has resulted in a slow uptake in the use of object-based coding for practical applications.

2.7.8 H.264 / MPEG-4 AVC

The ITU-T and ISO/IEC established a Joint Video Team to develop a new video compression standard using a "back to basics" approach [118]. In 2003, they proposed the H.264 standard [39, 91, 122], which has also been incorporated into MPEG-4 under the name of Advanced Video Coding (AVC).

H.264/AVC has many similar characteristics to previous standards, but some of the main new features are outlined below:

- Intra-frame coding is performed using 4×4 blocks, based on a fast integer approximation of the DCT. Spatial prediction within frames is also used to achieve additional de-correlation.
- Up to five reference fames may be used for motion estimation (as opposed to the one or two frames used in previous standards).
- For each 16×16 macro-block, variable-size block matching (VSBM) is used. This allows a range of different block sizes for motion compensation — from 16×16 down to 4×4 pixels.

- Motion vectors can be specified to one-quarter pixel accuracy (or one-eighth pixel in the case of chrominance components).
- An adaptive de-blocking filter is used within the motion compensation loop in order to improve picture quality.
- Context-adaptive binary arithmetic coding (CABAC) is employed in order to achieve efficient entropy coding of data using prediction within multiple contexts. (This is discussed further in Appendix B.)

H.264/AVC has been demonstrated to provide significant rate-distortion gains over previous standards, and it is widely accepted as the state-of-the-art in video compression [91, 114, 121]. As a result, efficient implementations of the standard are increasingly being used in a range of applications.

Chapter 3

Region Shape Coding

3.1 Introduction

One of the advantages of object-based video compression methods (such as MPEG-4 [84]) is that they allow a scene to be decomposed into its constituent parts. This is intuitively reasonable when one considers that each object within a scene will generally have its own specific motion and texture characteristics.[1] Nevertheless, using an object-based approach means that there is an additional overhead introduced, since it becomes necessary to encode the shape of each object within a scene.

If region boundaries are represented precisely, the shape information overhead can be substantial. However, if regions are only approximated very coarsely (with a corresponding low overhead), then many of the advantages of object-based coding might be lost. This is because the approximated region boundaries would not then describe objects with homogeneous motion and texture properties.

Some material in this chapter is based on the author's previously published work [102].

[1] In contrast, using a traditional block-based approach can often lead to blocking artifacts. These arise when a block spans two or more regions with different motion properties, but is only able to represent one type of motion.

From a coding point of view, it is thus desirable to achieve the optimal combination of shape, motion and texture information for each object. In practice, such a joint optimisation would be computationally intensive due to the interdependent nature of shape, motion and texture information [95].

Instead, this chapter describes a method for the progressive coding of shape information for *all* regions in a scene. It allows for an embedded bit-stream to be produced, so that the initial portion of the bit-stream provides a coarse region description, while subsequent bits allow this to be further refined.

In order to encode a segmentation map (which represents the set of all regions in a scene), the proposed method proceeds as follows: First, common boundary segments between neighbouring regions are identified. Using these segments, an initial polygon structure is created, which only coarsely approximates the actual map, but which fully specifies the connectivity between neighbouring regions. Following this, the existing segments are progressively refined until either the desired bit-rate is reached, or an acceptable error is achieved.

3.2 Related Work

Shape compression (for individual foreground objects) has been researched extensively in recent years and two main approaches have emerged: direct bitmap coding of the object mask, and coding of the shape boundary. The latter method has been shown to allow a greater degree of compression [95]. Region boundary coding is typically achieved through the use of (lossless) chain-coding [23, 20] or (lossy) polygon/spline approximation.

In the case of lossy region boundary coding (with either polygons or splines), the goal is to minimise both the coding cost and the distortion[2] caused by approximating the boundary. Some of the main contributions to vertex-based shape approximation are outlined below:

[2] Two error metrics are commonly used: the maximum perpendicular distance between the shape contour and the polygon approximation; and the area between the same two lines.

- O'Connell [75] presents a polygon-based region approximation method. Vertices are chosen so that the polygon they form does not deviate from the original shape contour by more than a given distance. The selected vertices are then coded using an octant-based representation. This incorporates the use of a variable dynamic range when coding the difference between consecutive vertices in a shape's polygon approximation.

- Le Buhan Jordan *et al* [51, 52] describe how region boundaries can be coded using a progressive polygon approximation method. This allows a coarse polygon shape to be coded first, followed by the transmission of successive refinement layers. This can proceed until either the desired bit-rate is reached, or the error is sufficiently small.

- Katsaggelos *et al* [44] provide a good review of object-based coding and focus on five shape coding techniques considered during the development of the MPEG-4 standard. Both intra and inter modes are considered: the former relates to the direct coding of an object's shape, while the latter involves coding the shape of an object relative to its previously coded representation in a reference frame.

  Of the five methods tested, a vertex-based approach is shown to offer the best coding efficiency in the case of lossy shape compression. However, they explain that a macro-block, context-based shape coder was selected as the MPEG-4 baseline method because it allows for a simpler implementation in hardware.

- Schuster *et al* [96, 95] discuss the importance of rate-distortion optimisation when coding an object's boundary. With the use of Directed Acyclic Graphs, they propose an operationally optimal shape coding algorithm which yields a polygon/spline approximation of the original boundary. The joint optimal coding of multiple objects within a frame is also addressed, although it is assumed that the coding of each boundary is accomplished independently of the others.

- Hu *et al* [26] present a scalable, layer-based, vertex coding method. As a pre-processing stage, majority filtering and contour refinement are used in order to impose some constraints on a region's boundary. Starting with a coarse approximation, more vertices are added to refine the boundary representation as the number of layers is increased. They also describe a vertex coding scheme that performs well — particularly in the case of near-lossless approximation.

- Kondi *et al* [47] propose a content-based method that considers the surrounding texture information when coding an object's shape. This is done by using an adaptive distortion measure which is dependent on the texture profile along each point of the object boundary. Thus, if a sharp edge is present along a part of boundary, the region's shape can be approximated more precisely in this area. This leads to improved performance, both objectively and subjectively.

- Wang *et al* [120] introduce a new skeleton-based approach to shape coding, which involves decomposing an object into two components: a skeleton and a boundary distance from the skeleton. Both the skeleton and distance signals are then optimised within a rate-distortion framework. This method is reported to offer improved performance over vertex-based shape coding techniques.

In general, a scene may contain multiple objects. In this case, the shape and position of objects can be described using a *segmentation map*, where each region in a segmentation map has a unique label and consists of a connected neighbourhood of pixels.

The region coding methods outlined above are for *individual* regions (e.g. video objects in MPEG-4). However, a segmentation map is a collection of multiple regions. This suggests that many of the approaches used in shape coding would be useful in the coding of segmentation maps. Furthermore, the coding cost can be reduced because common boundaries between neighbouring regions only need to be coded once.

Some research into the coding of segmentation maps has been reported within the context of region-based video coding:

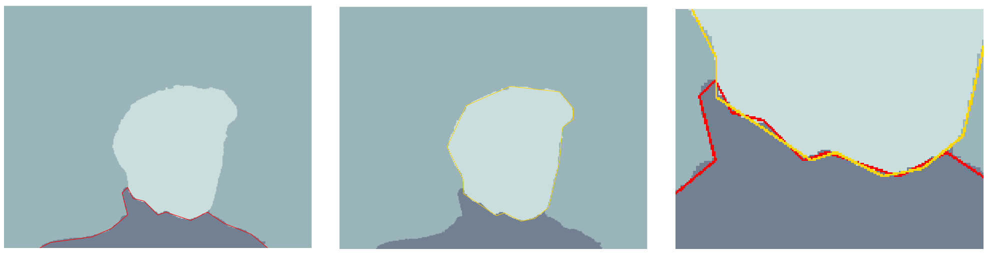
- Konrad *et al* [48] describe an object-based video codec which incorporates a polygon approximation of each region. The cost of coding a segmentation map is shown to be roughly proportional to the number of contour points. Chain coding is used to represent the boundaries of small regions, while large regions are coded using a polygon approximation method that involves coding the distance between consecutive vertices.
- Pateux *et al* [80, 73] propose a method for the lossy compression of segmentation maps, for use in video coding. A Minimum Description Length (MDL) minimisation approach is used when determining the vertices required to approximate the segmentation map. Note that the MDL criterion takes into account the coding of not only the segmentation map, but also the motion vectors and the resulting displaced frame difference. The scheme performs well when the boundaries of the segmentation map match motion discontinuities present in the scene.

The remainder of this chapter discusses the progressive polygon coding of segmentation maps. It is assumed that some form of spatio-temporal segmentation is used to generate the segmentation map, and that this segmentation process is performed prior to encoding the map.

3.3 Coarse Polygon Approximation

3.3.1 Finding the Initial Polygon Segments

Given a segmentation map comprising a number of regions, the goal is to approximate each region with a polygon. However, a simple polygon approximation of each region in turn is likely to result in a new segmentation map in which the the polygon-shaped regions are not correctly aligned (as illustrated

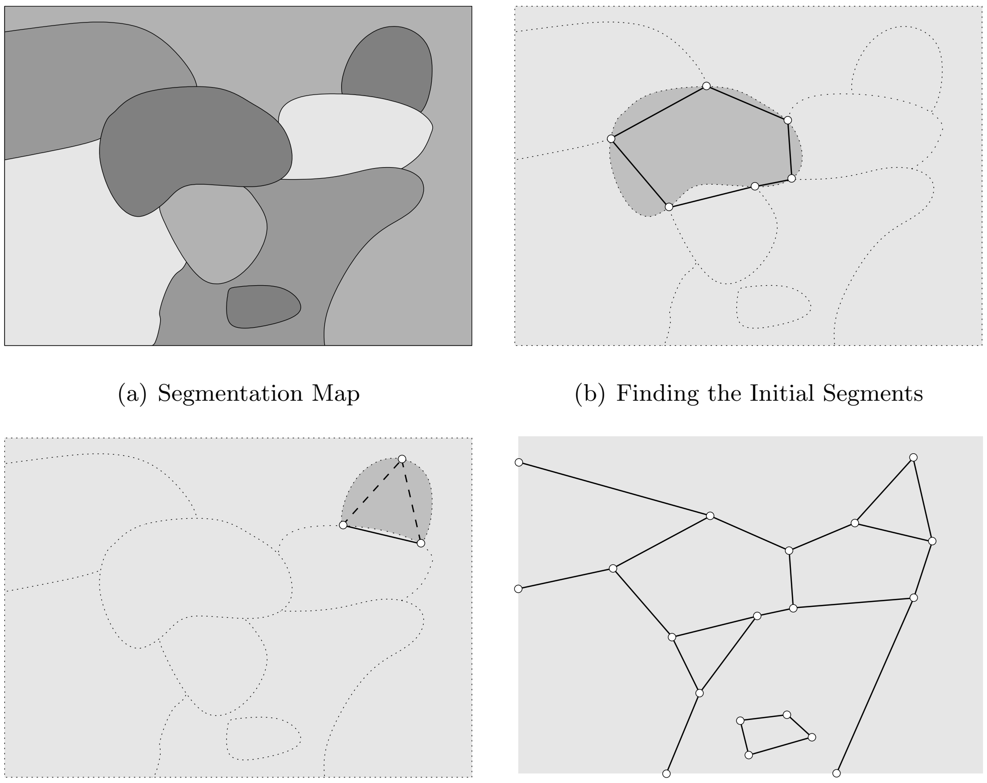


(a) Region 1 (b) Region 2 (c) Common boundary

Figure 3.1: Polygon approximation applied to two neighbouring regions *independently*. The polygons do not co-incide along the regions' common boundary.

in Figure 3.1). This is because the polygons corresponding to neighbouring regions could either overlap or leave gaps along their common boundary. This can be avoided by the method outlined below and illustrated in Figure 3.2:

- For each pair of neighbouring regions, the common region boundaries are identified.
- Each common boundary is approximated by a straight-line segment connecting the two common boundary endpoints.
- A region can then be approximated by a polygon comprising the segments along that region's boundary, as depicted in Figure 3.2(b).
- If a region only has two neighbours, it will just have one segment associated with it. In this scenario, two extra segments are created. This is done by joining the two existing endpoints with the boundary point furthest from the first segment, as illustrated in Figure 3.2(c).
- Those regions totally surrounded by another region are also treated differently, since their boundaries are circular. Such regions are approximated by four segments according to the initial polygon approximation algorithm in [1]. This involves selecting the two boundary points that are furthest apart, plus another two points furthest from the line joining the first pair.


(a) Segmentation Map (b) Finding the Initial Segments

(c) Region with 2 neighbours (d) The Segment Graph

Figure 3.2: Generating the initial segments

- It can occasionally happen that some of the initial segments intersect one another. Should this occur, the crossing segments are refined (using the process outlined in Section 3.4) until they no longer cross.

Taken collectively, the initial segments provide a rough approximation of region shapes, as well as a complete representation of the adjacency between regions.

3.3.2 Coding the Segment Graph

Having determined the initial segments, the encoder's task is to compress them efficiently. This is achieved by first representing the segments as a graph of connected nodes, as shown in Figure 3.2(d). Graph-based coding [73] with relative addressing of node positions is used in combination with entropy coding to achieve compression.

When encoding or decoding the graph, each family of connected segments is processed separately. (For example, the graph in Figure 3.2(d) contains two such families: one with 15 nodes, and the other with four nodes.) The process for coding each set of connected segments is outlined below using pseudo-code.

Algorithm for encoding a set of connected line segments:

```
Stack = {};
CurrentNode = ChooseAnyStartingNode();
ListOfCodedNodes = {CurrentNode};
Encode(CurrentNode); // absolute position
while ( CurrentNode $\neq$ {} ) do
  if (Arrived at Image Boundary) then
    CurrentNode = PopStack();
  else
    if CurrentNode has a neighbour $\notin$ ListOfCodedNodes then
      Encode(HasNewNeighbour = 1);
      Encode(NeighbouringNode - CurrentNode); // relative position
      PushStack(NeighbouringNode);
      CodedNodes = {CodedNodes, NeighbouringNode};
    else
      Encode(HasNewNeighbour = 0);
      CurrentNode = PopStack();
    end if
  end if
end while
```

The decoding process operates in a similar way and is simply a mirror of the encoder. Note that segments along the image boundary do not need to be coded, since these can be determined implicitly from other segments that terminate on the boundary. (This assumes that the width and height of the segmentation map are known to the decoder.)

3.4 Progressive Refinement

Once the initial segments have been specified, a more accurate representation of the original segmentation map can be obtained, by refining each of the segments in turn. As illustrated in Figure 3.3, the progressive refinement algorithm proceeds as follows:

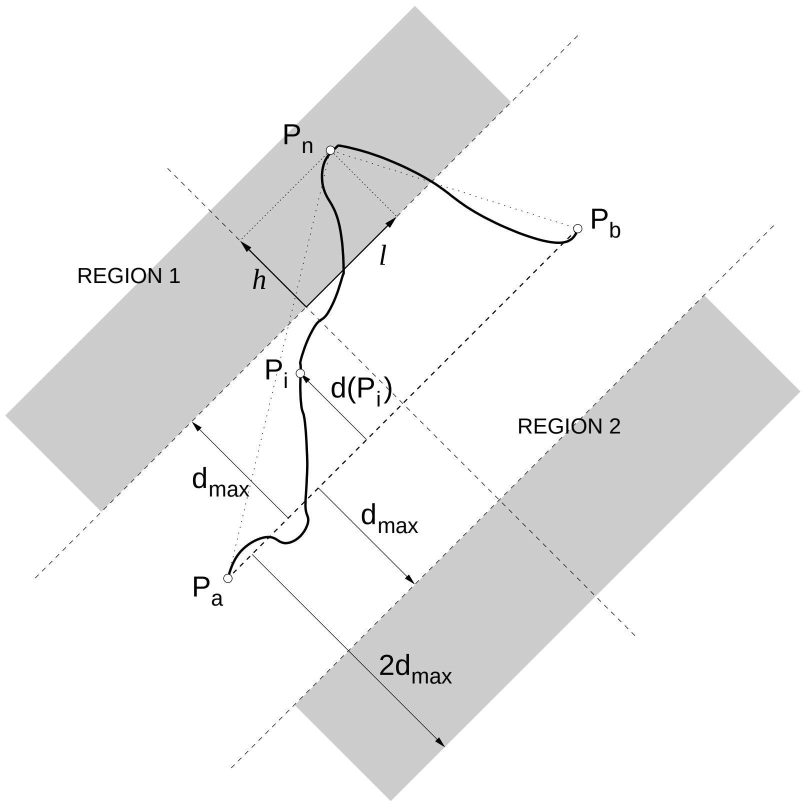


Figure 3.3: Refining a segment (between the points P_a and P_b)

- Consider the contour from P_a to P_b and the straight-line segment between the same two points.
- For each point P_i along the contour, calculate the perpendicular distance, $d(P_i)$, between P_i and the line segment from P_a to P_b.
- For some given threshold, d_{max}: if there exists a point P_i for which $d(P_i) > d_{max}$, then the segment needs to be split. If there is no such point, then the segment is considered sufficient.

- If splitting is required then:
  - Encode $SPLIT =$ true.
  - Find the point along the contour that has the greatest perpendicular distance from the straight line segment P_aP_b. Call this point P_n.
  - Encode the position of point P_n.
  - Discard segment P_aP_b and replace it with two new straight-line segments: P_aP_n and P_nP_b.
- Otherwise (if no splitting required): Encode $SPLIT =$ false.
- Proceed to the next segment.

The above process is applied to all segments for a large initial value of d_{max} (e.g. $max(N_R, N_C)$, where N_R and N_C are the number of rows and columns in the segmentation map). Once all segments have been checked, d_{max} is halved and the process is repeated. This continues until either a small enough value of d_{max} is reached, or the target number of bits is attained.

When performing entropy coding of the *SPLIT* flag, the following rules of thumb were found to provide a reasonable guide to actual statistics:

- For large d_{max}: Probability($SPLIT =$ true) $\approx \frac{1}{d_{max}}$.
- When d_{max} is halved, the number of segments likely to split will double.

Furthermore, when coding the position of the new node, P_n, the following properties were found to be useful:

- Clearly $d(P_n) > d_{max}$. Note that if this were not the case, the segment would not need to be split.
- In the vast majority of cases $d(P_n) < 2d_{max}$. This is because for $d(P_n) > 2d_{max}$ the segment would (almost always) have been split in the previous iteration.

- In general, P_n is more likely to occur close to the midpoint of the line between P_a and P_b, than far beyond either of the two endpoints.

Thus in the vast majority (roughly 95 %) of cases, P_n falls into one of the two regions shaded grey in Figure 3.3. The above properties allow for the position of P_n to be encoded efficiently, using entropy coding of the following three values:

- Side: It is necessary to specify whether P_n lies on the left or the right of the straight line from P_a to P_b. (In Figure 3.3 it lies on the left.)
- Perpendicular distance: h is equivalent to $d(P_n) - d_{max}$.
- Parallel distance: l is the component parallel to P_aP_b and is measured from the midpoint of the line P_aP_b.

The algorithm outlined above allows for progressive refinement of the segmentation map in such a way that an *embedded* bit-stream is produced. Thus a decoder can decode the bit-stream until the desired level of refinement is achieved (i.e. a small enough value of d_{max}).

3.5 Results

The proposed algorithm was tested on five segmentation maps with between 10 and 50 regions each. The rate-distortion curves for two of these segmentation maps are plotted in Figure 3.4, and are representative of the full set of five. The two segmentation presented here were derived from the *Lena* and *Goldhill* images.[3]

It can be seen that the cost of encoding the coarse segmentation maps is approximately 0.2 bits per contour point (bpcp). After this, the distortion falls off rapidly as the bit-rate is increased. (The total number of contour points does not include those along the image boundary, since these are trivial to encode.)

[3] A MATLAB demonstration can be downloaded from [99]. This includes the segmentation maps used in testing, as well as source code for the decoding of segmentation maps.

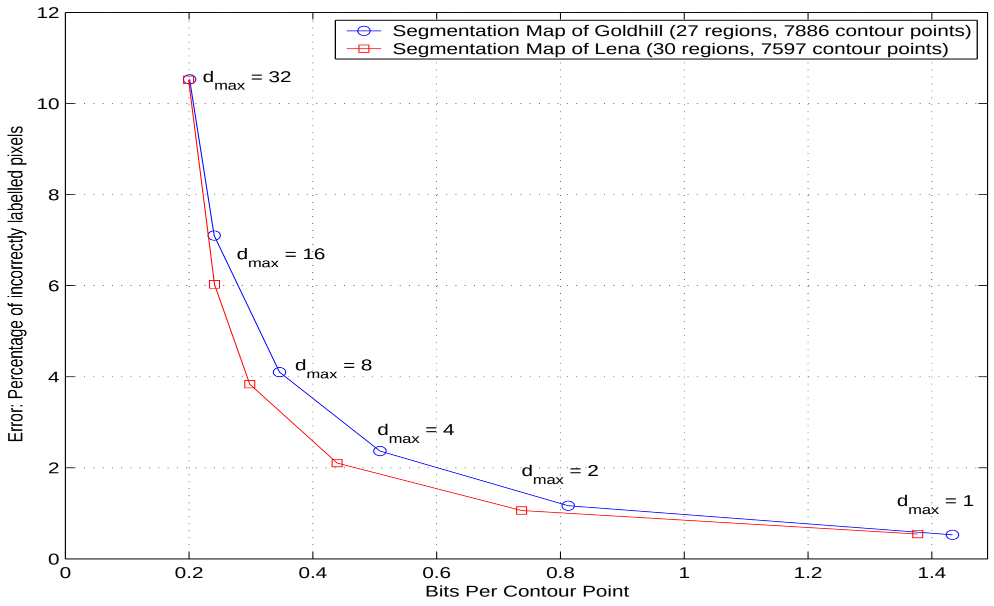
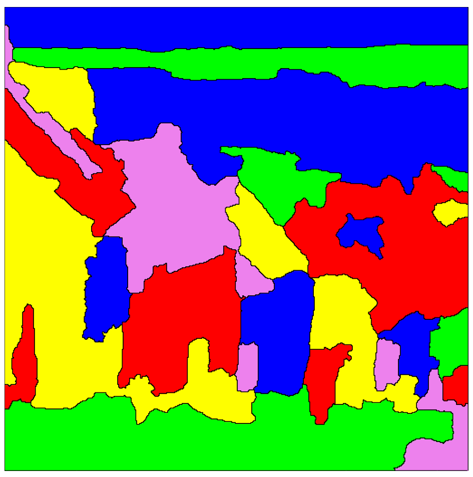


Figure 3.4: Rate-distortion curves for progressive polygon encoding of two segmentation maps

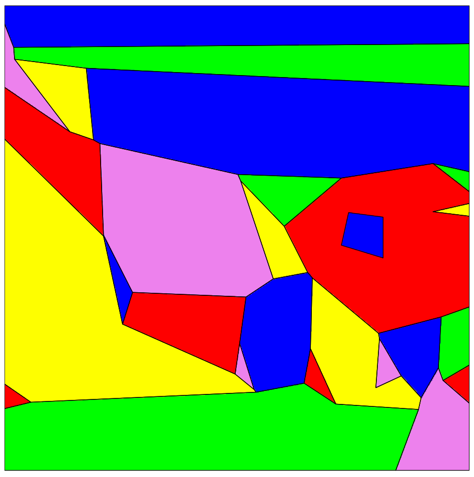
No standard segmentation maps exist, so a direct comparison between different methods is difficult. However, the observed results compare favourably with the values of 0.65 bpcp (for $d_{max} = 4$) and 0.5 bpcp (for $d_{max} = 8$) obtained in [75]. Lossy compression rates of 0.4 to 0.6 bpcp are reported in [80], though the corresponding approximation error is not quoted.

Figures 3.5 and 3.6 demonstrate the operation of the proposed polygon shape coding technique for segmentation maps derived from the 512×512 *Goldhill* and *Lena* images. In each case the original segmentation map is shown together with the initial coarse polygon approximation, as well as reconstructions for $d_{max} = 16$ and $d_{max} = 4$.

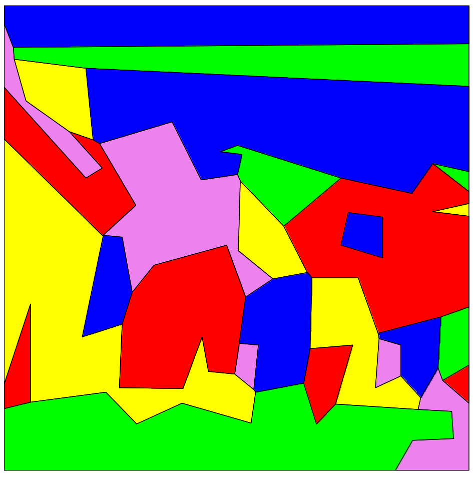
In both examples, the initial coarse approximation produces a segmentation map which at first glance bears little resemblance to the original. When decoding with the maximum distance d_{max} set to 16, the reconstruction process produces segmentation maps that are significantly more accurate, although the regions still have an artificial, “angular” appearance. Using a value of $d_{max} = 4$



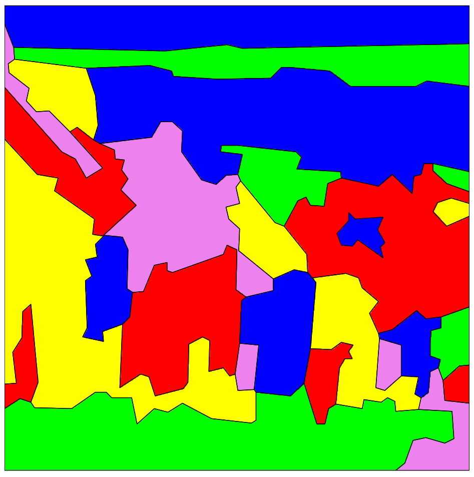
(a) Original Segmentation Map



(b) Initial coarse polygon approximation, using 162 bytes; Labelling error: 17.0%.

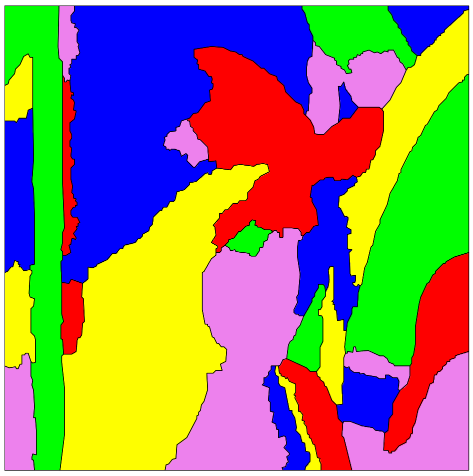


(c) Polygon approximation for $d_{max} = 16$ using 235 bytes; Labelling error: 7.1%.

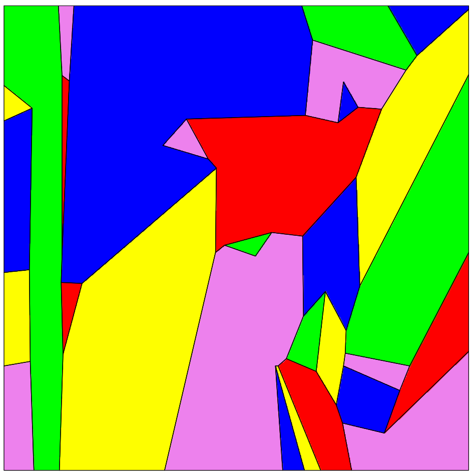


(d) Polygon approximation for $d_{max} = 4$, using 499 bytes; Labelling error: 2.4%.

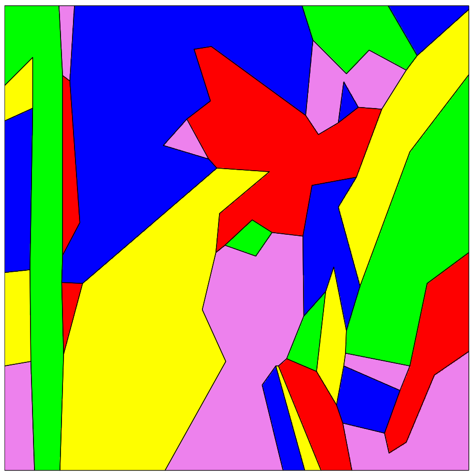
Figure 3.5: An example of progressive polygon approximation using a 27-region segmentation map derived from the 512 × 512 "Goldhill" image.



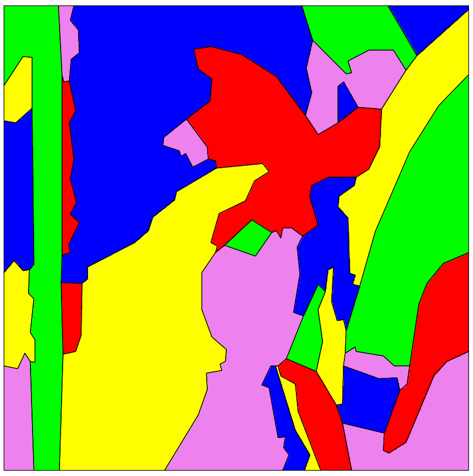
(a) Original Segmentation Map



(b) Initial coarse polygon approximation, using 176 bytes; Labelling error: 13.3%.



(c) Polygon approximation for $d_{max} = 16$ using 227 bytes; Labelling error: 6.1%.



(d) Polygon approximation for $d_{max} = 4$, using 416 bytes; Labelling error: 2.2%.

Figure 3.6: An example of progressive polygon approximation using a 30-region segmentation map derived from the 512×512 "Lena" image.

yields regions with more realistic contours and a much smaller region labelling errors.

In general, a polygon representation of a segmentation map can provide significant savings in rate, while introducing only a relatively small distortion. For example, a lossless PNG[4] compression of Figure 3.5(a) requires 6072 bytes, while the approximation in Figure 3.5(d) requires only 499 bytes. Similarly, 6078 bytes are needed for lossless PNG coding of Figure 3.6(a), though the representation in Figure 3.6(d) requires just 416 bytes.

3.6 Conclusion

The results reported in this chapter demonstrate that efficient lossy coding of segmentation maps is possible at rates in the order of 0.5 bits per contour point. The proposed method first encodes a set of initial segments in order to represent each region coarsely as a polygon. This is followed by the progressive refinement of these segments, until the desired accuracy or bit-rate is reached.

The progressive coding of segmentation maps offers advantages for object-based video coding, since it allows shape coding to be drawn within a rate-distortion framework in a video codec. Ideally, the segmentation process itself needs to be similarly controllable to allow for a variable number of regions.

Another option worth further investigation is allowing d_{max} to be weighted according to the similarity between neighbouring regions [47]. Thus if two regions have widely differing spatial and/or temporal characteristics, d_{max} should be reduced along their common boundary in order to allow for a more precise approximation of that boundary.

The next two chapters discuss region-based motion compensation methods which make use of the proposed technique for polygon shape coding. As will be shown, it is particularly advantageous when region/polygon boundaries cor-

[4] Portable Network Graphics (PNG) image format for lossless bitmap compression [33].

respond to motion boundaries, since this allows regions with different motion characteristics to be coded separately.

Chapter 4

Motion Compensation using a Triangular Mesh

4.1 Introduction

Block-based methods have traditionally been used to perform motion estimation and compensation. They offer the advantage of being fast, easy to implement and fairly effective over a wide range of video sequences.

Nevertheless, regular block-based approaches suffer from two drawbacks. First, the size and shape of blocks is fixed, and consequently independent of the scene content. Thus a block covering two regions with different motion will not be able to accurately model the motion within both regions simultaneously.[1] Secondly, block-matching is typically used to estimate only translational motion, and in this case it cannot accurately model more complex types of motion such as rotation and zooming.

Triangular and quadrilateral meshes have been proposed as a means of addressing these limitations. They allow for atomic motion compensation units (i.e.

Some material in this chapter is based on the author's previously published work [100, 103].

[1] Variable-size block matching addresses this problem to a limited degree by allowing a range of block sizes (e.g. from 16×16 down to 4×4 in the H.264/AVC video coding standard).

blocks or triangles) to vary in size and shape according to scene content, and are often used in combination with complex motion models.

This chapter describes how a content-based, triangular mesh can be generated, and then used for affine motion compensation. Spatio-temporal segmentation is performed using the current frame and a reference frame. Once a segmentation map has been obtained for the current frame, regions in the frame are approximated with polygons. A triangular mesh is then created within each polygon-shaped region, resulting in the frame being fully covered by triangles. Finally, translation and affine motion parameters are estimated for each triangle.

4.2 Related Work

4.2.1 Segmentation

Image and video segmentation is a complex field and remains an active area of research. The purpose of segmentation is to partition a scene into its constituent objects. Ideally, an object will have homogeneous motion, texture and colour characteristics, however, this is seldom true in practice. (For example, is the windscreen of a car part of the car or the background?) Another problem is the degree to which segmentation should be performed. (For example, are a nose and a mouth distinct objects, or are they both part of a "face" object?)

Spatial segmentation is applied to 2D-images and generally operates by separating an image into regions that are homogeneous in terms of texture and/or colour. Spatio-temporal (3D) segmentation operates on a sequence of images and uses motion information as well.

Providing a detailed review of segmentation methods is beyond the scope of this dissertation. However, the role of segmentation in the mesh generation process is important, since poor segmentation will lead to an ineffective mesh design. Therefore, some pointers to recent work on (and reviews of) segmentation methods are provided. This includes a brief overview of the spatial segmentation technique used in the mesh design process.

- Salembier and Marques [94] provide a good review of spatio-temporal segmentation methods. They discuss two classes of segmentation techniques: transition-based methods and homogeneity-based approaches. The former involves determining the transition or edge areas in an image/video. The latter operates by grouping together homogeneous elements into regions. Following this review of segmentation methods and tools, they go on to propose the use of a partition tree for representing regions within images. It is demonstrated how partition trees can be pruned according to the level of detail required for a particular application. (For example, a branch of a tree corresponding to a head, torso and four limbs can be pruned until there is one region representing the whole body.)
- Some more recent spatio-temporal segmentation methods include those of Patras *et al* [81] (who propose a watershed-based segmentation method in which regions are merged if they exhibit similar motion properties); Shamim and Robinson [105] (who use a top-down approach to identify and classify regions with different motion characteristics within a frame); Izquierdo and Ghanbari [40] (who describe an advanced segmentation system that combines the outputs from six segmentation methods); and Mezaris *et al* [64] (who adopt a Bayesian approach to the tracking of objects, and perform region merging using long-term motion trajectories).
- Deng and Manjunath [16] describe JSEG, a method for the robust segmentation of colour-texture regions in images and video. JSEG operates by first quantising colours in the scene into a number of classes within the CIE LUV [14] colour space. (Quantisation is performed more coarsely in textured areas.) The quantised colours are then segmented spatially by attempting to maximise the class homogeneity within each region, across a range of scales. When operating on video, a region tracking scheme is incorporated into the process so as to achieve consistent results across

several frames. JSEG demonstrates good segmentation performance on a variety of images, and a software implementation of the method is available for downloading [17].

4.2.2 Motion Estimation

There are many different ways of estimating motion within a sequence. This section focuses on two aspects of motion estimation that are used later in the chapter: optical flow and model-based motion. Optical flow plays a role in the segmentation stage, when it is used to determine the motion properties of regions in a scene. Model-based motion is considered from the point of view of representing more complex motion than simply translation.

Optical Flow

Knowing the motion vectors at each pixel position within a frame allows one to determine the motion properties of any region in the frame. This is useful for refining segmentation boundaries originally based on spatial information only.

When objects move in reality, their motion can be represented with 3D motion vectors. A *dense motion vector field* is simply a mapping of these motion vectors onto the 2D image plane, with one motion vector for each pixel in the image. In contrast, *optical flow* is defined as a 2D velocity field that represents how pixels in a frame can be mapped in order to reconstruct another frame in the sequence. In general, the optical flow is not unique, since it may be possible to map a pixel in two or more different ways - particularly in areas with little texture. Nevertheless, optical flow and dense motion fields are usually very similar for a pair of images, and the terms are often used interchangeably.

The *gradient constraint* equation [25] is defined as

$$\frac{\partial I}{\partial x}v_x + \frac{\partial I}{\partial y}v_y + \frac{\partial I}{\partial t} = 0 \tag{4.1}$$

where $I(x, y, t)$ is the intensity of a pixel at point (x, y) and time t, and v_x and v_y are the horizontal and vertical components of the optical flow. Note that this equation has two unknowns (i.e. v_x and v_y). Thus in order to solve it, an additional constraint is necessary.

- Horn and Schunk [25] assume that changes in the scene are only due to motion and not because of any variation in brightness. They add a global smoothness term in order to constrain the estimated velocity field.
- Lucas and Kanade [54] introduce a local smoothness constraint using a least squares fit within a small spatial neighbourhood. If the neighbourhood is too small, it may not constrain the problem sufficiently. However, if it is too large, there may be multiple motions present within the neighbourhood.
- Black and Anandan [4] describe how the *single motion assumption* as well as the *constant brightness condition* are not always valid. They discuss how these assumptions can be relaxed in order to develop a more robust estimation process. This robustness is achieved by detecting and rejecting outliers that occur when assuming that the constant brightness and single motion conditions apply. A software implementation of their robust method for determining optical flow is available for downloading [3].

Model-based Motion

The vast majority of motion estimation methods in use today operate by determining the translational motion of regions (usually blocks) within a frame, relative to one or more reference frame(s). Dufaux and Moscheni [19] and Stiller and Konrad [109] provide good reviews of motion estimation techniques, and list a variety of motion models in addition to the two-parameter translation model.

The most commonly used higher order motion models are probably the affine and projective ones. The six-parameter affine model allows for translation, zooming, rotation and shearing to be represented. In addition to these, the eight-parameter projective model helps to convey depth and perspective by allowing a plane to be mapped to any other plane.

Higher order models enable the motion of regions (usually blocks or semantic objects) to be represented much more accurately. However, there are two reasons why the use of such models has been fairly limited. First, estimating six or eight parameters requires significantly more computation than finding the best translational match using a 2D search. Secondly, from a rate-distortion point of view, the advantage provided by the use of a complex motion model (in terms of reduction in distortion) needs to outweigh the cost of coding the extra motion parameters.

4.2.3 Mesh Generation

Once some initial pre-processing and/or segmentation has been performed, a 2D mesh can be overlaid on a frame. This allows the mesh to be constructed around objects in the scene. Content-based motion compensation is then possible, generally allowing for superior visual quality compared to motion compensation using either fixed-size blocks or a regular mesh. Many different mesh design strategies have been proposed, and a brief review of selected methods is provided below:

- Nieweglowski *et al* [74] and Nakaya and Harashima [67] propose the use of triangular and quadrilateral 2D meshes as an alternative to block-based systems in order to allow for affine motion compensation. As each node within a mesh moves, so it causes the triangles of which it is a vertex to warp. This warping effect allows for more complex motion to be modelled.

- In a regular mesh, nodes are placed at fixed intervals throughout the image. However, Dudon *et al* [18] show that it is advantageous (and intuitively plausible) to position the nodes along object boundaries, or even to have a separate mesh for each foreground object.

- The use of meshes for object-based video has been encouraged by the development of the MPEG-4 standard [84], in which Video Object Planes are used to represent arbitrarily shaped regions within a scene [116]. One disadvantage of using an object-based mesh is that the boundary of each object needs to be specified, resulting in a significant overhead. In practice, regions are often approximated using polygons/splines, or from region boundaries in preceding frames.

- Bradshaw and Kingsbury [6, 5] consider the fact that when using a fully-connected mesh for motion compensation, neither occlusion nor uncovering can be accurately modelled. Their proposed solution involves causing the mesh to rip or tear along occlusion boundaries, and allowing overlapping of the mesh along these tears.

- Altunbasak and Tekalp [1] present another solution to the problems caused by occlusion and uncovering. They propose first identifying the *background to be covered* within a frame, and allowing no nodes to be placed there. In addition, a *model failure* region is detected and the mesh is refined inside this region. This method was demonstrated to perform well for "head and shoulder" type sequences.

- Van Beek *et al* [117] discuss the design, tracking and coding of 2D meshes for use in video compression. In particular, they focus on a hierarchical approach. During the mesh design phase, a fine-to-coarse combination of Delaunay meshes is generated, allowing for a mesh to be created for each object with a varying density of nodes and triangles. When tracking a mesh from one frame to the next, a coarse-to-fine strategy is adopted for determining the motion vectors. The shape and motion information is coded in such a way as to allow for the progressive transmission of this information. Celasun and Tekalp [8] extend this hierarchical approach by optimising the mesh design process. They also allow the mesh structure to be updated when occlusion is detected.

- Most of the approaches outlined above employ motion compensation of the current frame from *one* previously encoded reference frame. This is similar to an MPEG P-frame (with the obvious exception that mesh-based motion compensation is used). Eren and Tekalp [21] describe the design of a mesh for video objects that allows for bi-directional motion compensation. In a similar way to MPEG B-frames, this method uses two reference frames - one in the past and one in the future.

Delaunay Triangulation

Many mesh design methods employ Delaunay triangulation [15]. This process can be applied to a set of points in order to create a mesh of triangles that uses this set of points as the triangle vertices. Delaunay triangulation operates in such a way that no vertex lies inside the circumcircle[2] of any triangle in the mesh. Shewchuck [107, 108] provides a software implementation of 2D Delaunay triangulation. In addition to being applied to a set of points, Shewchuck's implementation allows for the triangulation of a *planar straight line graph*.[3] This allows for Delaunay triangulation to be applied to a polygon, resulting in the creation of a triangular mesh within such a polygon.

4.3 Mesh Generation

4.3.1 Spatio-Temporal Segmentation

General-purpose spatio-temporal segmentation methods divide a scene into regions that differ in both motion and spatial characteristics (e.g. colour, texture and intensity). However, from the point of view of motion compensation for video coding applications, it is not necessary to segment a group of objects that are moving similarly (and thus have the same motion properties). Consequently, regions only need to be segmented if they have different motion characteristics.

[2] The circumcircle of a triangle is the circle that passes through all three of its vertices

[3] A planar straight line graph is a collection of straight-line segments (with vertices as endpoints) whose presence in the mesh is enforced.

Segmentation based on motion information *only* was found to provide unsatisfactory results, because motion at object boundaries is often difficult to measure precisely due to occlusion. It was therefore decided to use two established schemes to obtain reasonable segmentation results across a range of different sequences: JSEG [16, 17] is used to achieve spatial segmentation, and Black and Anandan's software [4, 3] is used to estimate the dense optical flow between two images. The motivation behind this is that spatial segmentation can be used to determine object boundaries quite precisely, and in addition optical flow provides the data for determining regions' motion characteristics.

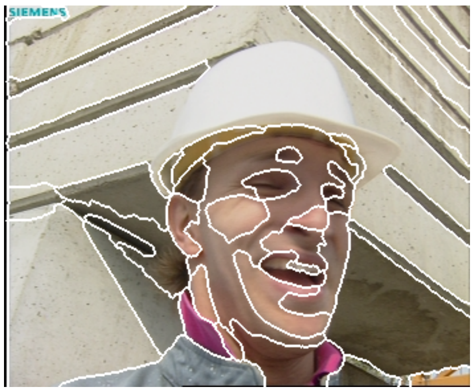
Given a current frame and a reference frame, the segmentation process then proceeds as follows (with Figure 4.1 illustrating the process for frame 13 of the *Foreman* sequence):

- Perform spatial segmentation of the current frame using JSEG. Figure 4.1(b) shows the regions obtained after spatially segmenting frame 13 of *Foreman*.
- Estimate the dense motion-vector field, relative to a reference frame (frame 11 in the example). This yields a motion vector at each pixel position. Figures 4.1(c) and 4.1(d) provide a representation of the horizontal and vertical components of the dense motion vector field.
- Split stage: Find all regions with a high motion vector variance and re-segment them spatially using JSEG. For a region r of size $A(r)$, the motion vector variance is calculated as

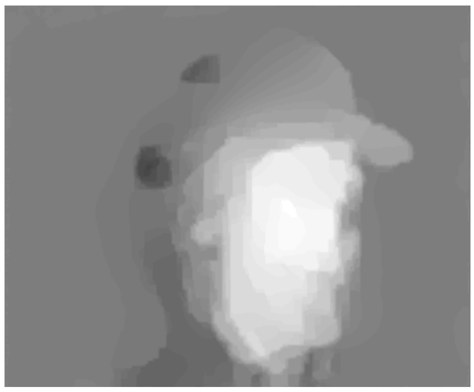
$$\sigma_{mv}(r) = \frac{1}{A(r)} \sum_{(x,y)\,\in\, r} \left((v_x - \overline{v_{r,x}})^2 + (v_y - \overline{v_{r,y}})^2 \right) \tag{4.2}$$

  where $\{(v_x, v_y)\}$ are individual motion vectors and $(\overline{v_{r,x}}, \overline{v_{r,y}})$ is the mean motion vector for region R_i. Regions with $\sigma_{mv} > T_1$ are selected for splitting, where T_1 is some threshold. The process is repeated until all regions have a sufficiently low motion vector variance. Figure 4.1(e) shows the result after regions have been further segmented during the splitting stage.

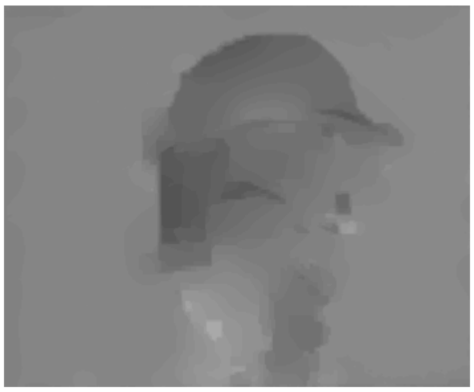
(a) Frame 13 of the *Foreman* sequence



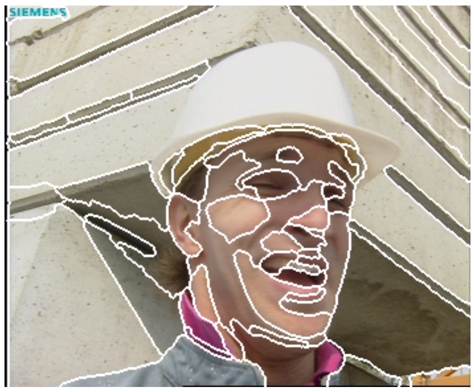
(b) After JSEG spatial segmentation



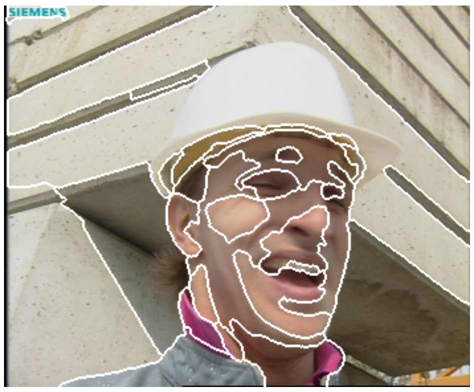
(c) A representation of the horizontal motion-vector field (frame 13 relative to frame 11); black: left, white: right



(d) A representation of the vertical motion-vector field (frame 13 relative to frame 11); black: down, white: up



(e) After region splitting



(f) After region merging

Figure 4.1: Spatio-temporal segmentation using region splitting and merging.

- Merge stage: If two neighbouring regions (r_i and r_j) have similar mean motion vectors *and* their joint variance remains low, then merge the two regions. Thus if:

  $$(\overline{v_{r_i,x}} - \overline{v_{r_j,x}})^2 + (\overline{v_{r_i,y}} - \overline{v_{r_j,y}})^2 < T_2 \tag{4.3}$$

  for some threshold T_2, and

  $$\sigma_{mv}(r_i \cup r_j) \leq max\,(\sigma_{mv}(r_i), \sigma_{mv}(r_j)) \tag{4.4}$$

  then regions r_i and r_j are considered to have similar motion characteristics and can consequently be merged. The process is repeated until there are no regions left that satisfy the above merging criteria. Figure 4.1(f) illustrates the result of merging neighbouring regions with similar motion properties.

The sensitivity of the spatio-temporal segmentation stage clearly depends on the thresholds T_1 and T_2. Various values were tested during the design stage, and reasonable values were determined empirically to be 0.8 and 2.0 respectively.

The end result of this spatio-temporal segmentation process is a segmentation map that defines regions in the current frame. If the process works as designed, then each region will represent an area of homogeneous motion, and no pair of neighbouring regions will have identical (or very similar) motion characteristics.

4.3.2 Polygon Approximation

Once the current frame has been segmented into regions, it is necessary to represent the shape of each region. Since the ultimate intention is to perform motion compensation for the purpose of video compression, it is important that the shape information should be coded efficiently.

This is done using progressive polygon approximation as described in Chapter 3. Note that this requires the user to specify the value of d_{max}, which is the maximum distance by which an approximated (polygon) boundary may differ from the original boundary.

Using this value of d_{max}, progressive polygon approximation is applied to the segmentation map obtained during the spatio-temporal segmentation stage. Figure 4.2(a) provides an example using the segmentation map shown in Figure 4.1(f), for $d_{max} = 4$,

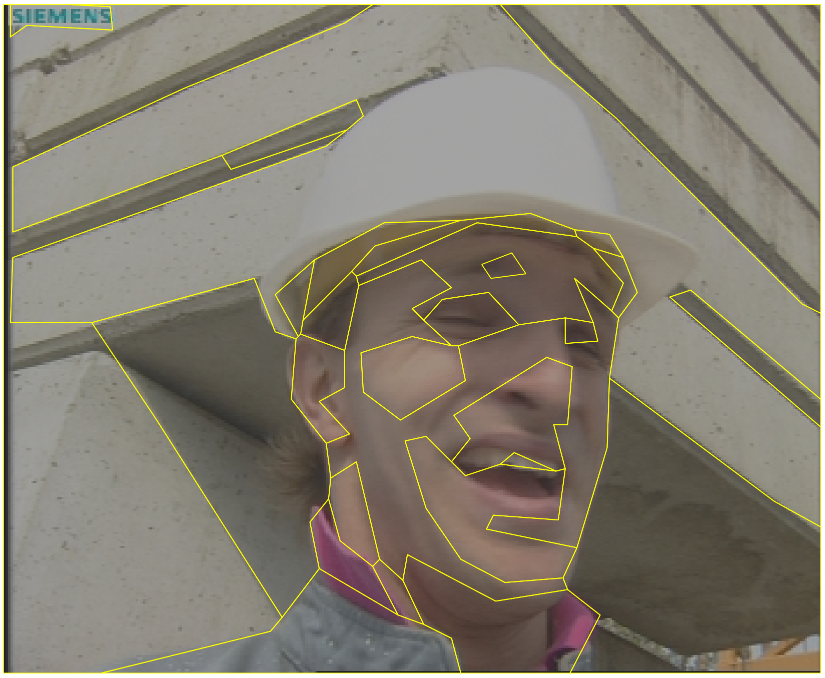
4.3.3 Triangulation

Following the approximation of each region with a polygon, the next step is to create a triangular mesh within each polygon. This is done by applying Shewchuck's implementation [107] of Delaunay triangulation to each polygon, and specifying that each polygon vertex should be a triangle vertex within the mesh.

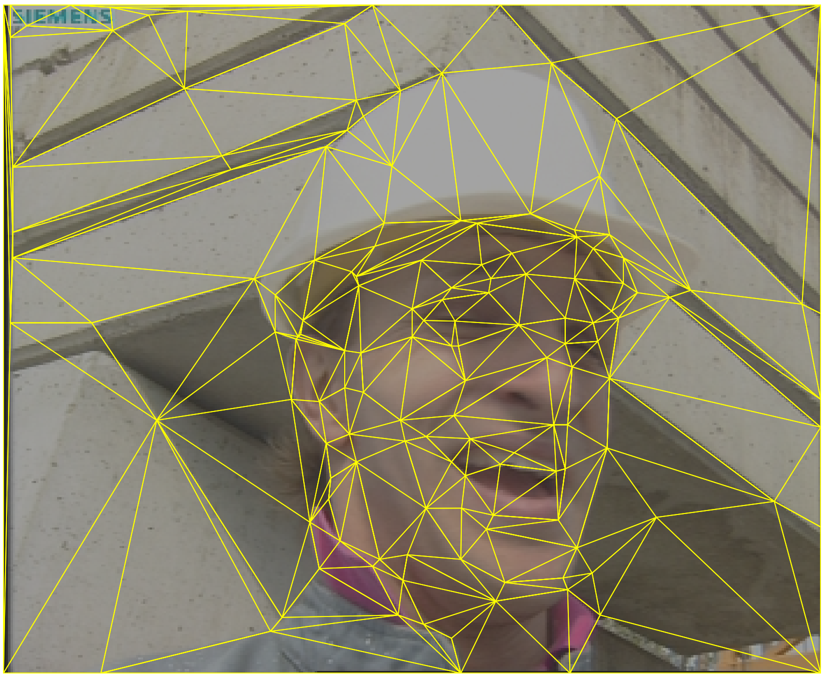
During the mesh generation process, the polygon boundaries are specified as segments within a *planar straight line graph*, and (where possible) triangles are created with all angles larger than 20°. This minimum angle rule-of-thumb helps to ensure that triangles are not too long and thin, which in turn enables more reliable motion estimation.

In addition, the number of interior nodes that can be added to a polygon during the triangulation process is limited to a maximum of four. This rule-of-thumb restriction on the number of interior nodes prevents a large number of very small triangles being created within each polygon. (Having too many triangles would allow more accurate motion compensation, but at the expense of coding their motion parameters.)

For a given frame (and reference frame), the number and shape of triangles in the mesh is dependent on the values of T_1, T_2 and d_{max}. Controlling these parameters in a reasonable way proved to be difficult across different sequences, so T_1 and T_2 were assigned fixed values, as described in Section 4.3.1. Thus for a particular segmentation map, the number and shape of triangles is dependent on d_{max}. As d_{max} increases, the number of vertices in the polygon-approximated regions decreases, resulting in a corresponding decrease in the number of triangles in the mesh.



(a) Polygon approximation of regions in Figure 4.1(f) using $d_{max} = 4$



(b) Delaunay triangulation of the polygons in (a)

Figure 4.2: Mesh generation using polygon approximation and triangulation

Figure 4.2(b) illustrates the results once triangulation has been applied to all the polygon-approximated regions in Figure 4.2(a).

4.4 Motion Estimation

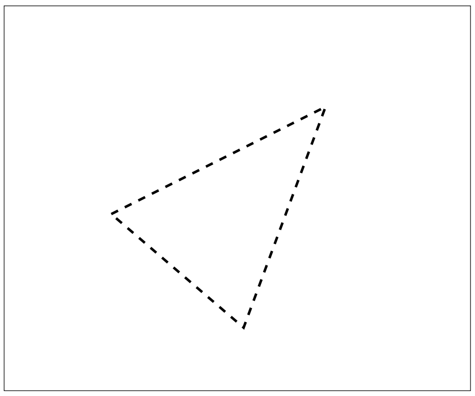
When estimating the motion of a triangle or its vertices, it is possible to use the dense motion field that was obtained as part of the segmentation process (see Section 4.3.1). However, this motion field was found to be generally unreliable near motion boundaries, and since many triangle vertices are located along such boundaries, this approach was considered unsatisfactory. In addition, for frames motion compensated from *two* reference frames, another motion vector field would need to be determined relative to the second reference frame.

It was therefore decided to use a two-stage "triangle-matching approach" for estimating motion. In the first stage, the translational motion of each triangle is estimated in a similar way to traditional block matching. Following this, the affine motion parameters are estimated for each triangle. The process is outlined below in more detail.

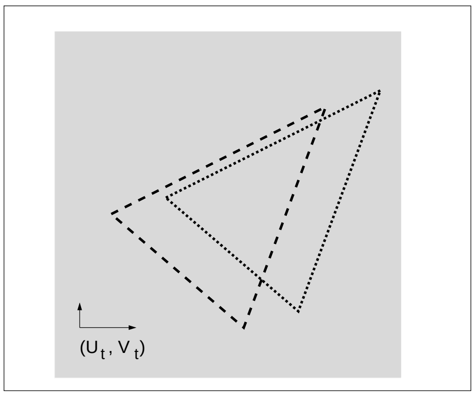
4.4.1 Translational Motion

The motion parameters for each triangle within the current frame are estimated independently of those for other triangles. This is possible because the mesh is non-connected, so that each triangle is motion compensated independently of its neighbours. The translation vector is determined as follows:

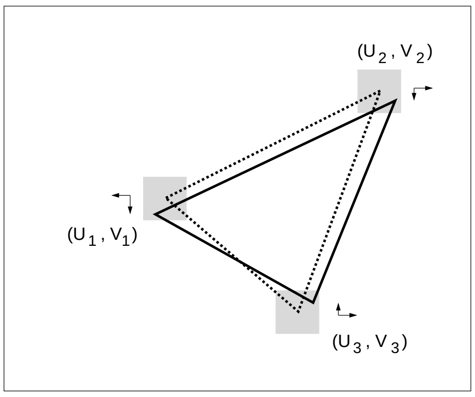
- Consider all the points within a given triangle in the current frame, such as the one depicted in Figure 4.3(a).
- For each possible translation, match these points to the corresponding set of translated points within the reference frame. Note that the translation is restricted to some specified search radius, as illustrated in Figure 4.3(b).



(a)



(b)



(c)

(a) The position of a triangle in the *current* frame.

(b) The dotted lines show the position of a triangle in the *reference* frame that is a translated version of the triangle in (a). Note that the initial matching is performed using translation only. The translation search region is shown in grey and the resultant motion vector, (u_t, v_t), is also depicted.

(c) The solid lines show the position of the triangle in the *reference* frame that is a translated and warped version of the triangle in (a). In this case, the initial matching has been performed using translation, and further refined with affine warping. The areas shaded grey show the search range for each vertex when estimating the affine motion. The three corresponding motion vectors for each vertex are also represented.

Figure 4.3: Translation and Affine Motion Estimation

- Matching is performed by calculating the sum of squared error (SSE) between the (colour or grey-scale) intensities of the points in the current frame and the intensities of the translated points in the reference frame. Alternatively, the sum of absolute error (SAE) or some other appropriate metric may be used.
- The shift which results in the smallest SSE is chosen as the translation vector, (u_t, v_t), for the triangle under consideration.
- Two reference frames are used - one before and one after the current frame. The one which gives rise to a smaller translational motion compensation error is selected as the sole reference frame for the affine motion estimation stage.
- Note that translation vectors are calculated to integer pixel accuracy. Sub-pixel accuracy can be achieved during the affine motion estimation stage. (Alternatively, if no affine parameters are to be computed, translational motion can be calculated to sub-pixel accuracy using bi-linear interpolation of pixel intensities.)

4.4.2 Affine Motion

Following the estimation of translational motion, each (translated) triangle in the reference frame is warped slightly to see if an even better match to the original triangle in the current frame can be obtained.[4] This warping is performed using the six-parameter affine model, as illustrated in Figure 4.3(c) and outlined below:

- Each of the three vertices of the (translated) triangle in the reference frame is moved within a small search area, while keeping the position of

[4] Note that it is possible to use the dense motion vector field to directly estimate a triangle's affine motion. However, this was found to produce inferior results to the matching approach described in this section. (This is because dense motion vector fields are often unreliable near motion boundaries.)

the other two vertices fixed.

- This results in a warped triangle, which is matched to the original in the current frame. Once again, matching is achieved by minimising the SSE for points inside the triangle (after affine motion compensation).

- For each of the three triangle vertices in the *current* frame, $\{(x'_i, y'_i) : i \in \{1,2,3\}\}$, the corresponding vertices in the *reference* frame, $\{(x_i, y_i)\}$ are related according to the equation:

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} x'_i \\ y'_i \end{bmatrix} + \begin{bmatrix} u_t \\ v_t \end{bmatrix} + \begin{bmatrix} u_i \\ v_i \end{bmatrix} \tag{4.5}$$

where (u_t, v_t) is the (optimal) translation motion vector calculated during the previous stage, and $\{(u_i, v_i) : i \in \{1,2,3\}\}$ are the *additional* displacements of the three triangle vertices in the reference frame.

- During the affine motion estimation stage, (u_i, v_i) is varied within a small search range for each of the three nodes. Note that consequently $\{(x_i, y_i)\}$ is known for each such variation. This is because $\{(x'_i, y'_i)\}$ and (u_t, v_t) are constant for each triangle, the latter having been determined during the translation estimation stage.

- In general, for a pixel (x', y') in a triangle in the current frame, the corresponding point (x, y) in the reference frame is given by the affine transform:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 + u_t \\ a_4 & a_5 & a_6 + v_t \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \tag{4.6}$$

where $\{a_1, \ldots, a_6\}$ are the six affine motion parameters. These can be determined by considering Equation 4.6 in the case of the three triangle vertices. This yields the system of equations:

$$\underbrace{\begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} x'_1 & y'_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x'_1 & y'_1 & 1 \\ x'_2 & y'_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x'_2 & y'_2 & 1 \\ x'_3 & y'_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x'_3 & y'_3 & 1 \end{bmatrix}}_{\mathbf{B}} \underbrace{\begin{bmatrix} a_1 \\ a_2 \\ a_3 + u_t \\ a_4 \\ a_5 \\ a_6 + v_t \end{bmatrix}}_{\mathbf{a}} \tag{4.7}$$

for which the positions of the vertices $\{(x'_i, y'_i)\}$ and $\{(x_i, y_i)\}$ are known. Solving for $\mathbf{a}$ yields:

$$\mathbf{a} = \mathbf{B}^{-1}\mathbf{x} \tag{4.8}$$

Motion compensation of any point in the current triangle can then be performed by substituting the affine motion parameters from **a** into Equation 4.6.

The above affine motion estimation process does not search the entire subspace of affine parameters. This is because each triangle vertex is moved individually within its search region (while the other two vertices are fixed). A full search involves moving the three vertices simultaneously across the range of possible positions (within the search radius). However, such a full search can be computationally expensive if a large affine search radius is used. In contrast, the sub-optimal search (i.e. moving one vertex at a time) is significantly faster, and was generally found to produce a good estimate of affine motion. A somewhat more robust sub-optimal affine motion estimation approach involves moving vertices one, two and three of the triangle, followed by a second perturbation of vertex one.

When applying motion compensation to a pixel position (which has integer coordinates), the resulting coordinates in the reference frame are (in general) non-integer real numbers. In this case, bi-linear interpolation (from the four neighbouring pixels) is used to estimate the intensity at the desired point in the reference frame. The use of bi-linear interpolation also enables affine motion estimation to be performed with sub-pixel accuracy, by allowing non-integer values for the motion vectors of the three triangle vertices, $\{(u_i, v_i) : i \in \{1, 2, 3\}\}$ in Equation 4.5.

The affine motion of a particular triangle is described in terms of six parameters. These can either be the $\{a_1, \ldots, a_6\}$ from Equation 4.6, or the motion vectors of the three vertices, namely the $\{(u_i, v_i) : i \in \{1, 2, 3\}\}$ from Equation 4.5. In a practical coding system it is often easier to use the latter, since the three motion vectors can be specified (or quantised) with equal precision.

Finally, it should be noted that the matrix $\mathbf{B}^{-1}$ in Equation 4.8 needs to be calculated only once for each triangle during the motion estimation stage. This is because the only variables $\mathbf{B}$ contains are the coordinates of the triangle in the *current* frame, which do not vary.

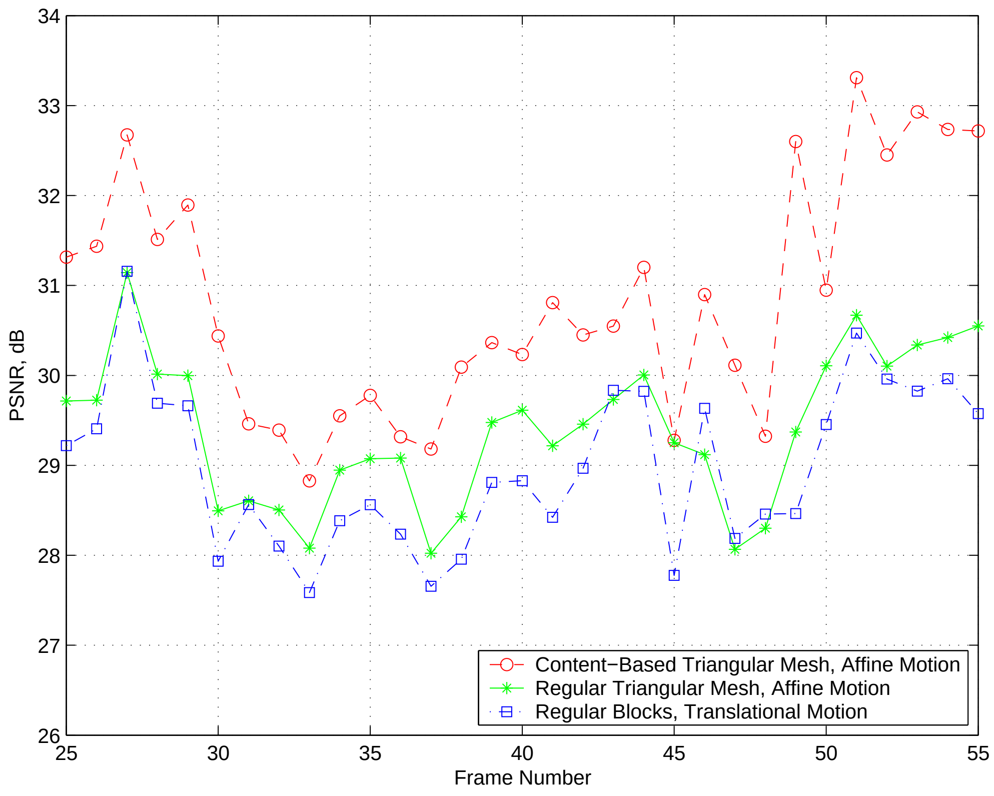
4.5 Results

The performance of mesh-based motion compensation was evaluated for seven[5] test sequences, using both regular and content-based meshes. This section presents results for the *Stefan* (CIF) and *Football* (SIF) sequences, both of which contain significant amounts of motion and texture information. (Note that the results shown here are representative of those obtained for the full set of seven sequences.)

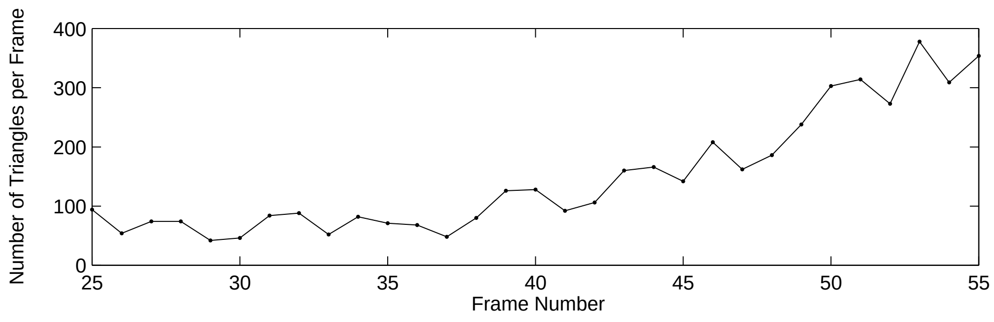
Figures 4.4(a) and 4.5(a) provide results for 31 frames from the *Stefan* and *Football* sequences respectively. Each compensated frame ($\tilde{I}_n$) has been motion compensated from two original frames (I_{n-2} and I_{n+2}) preceding and subsequent to it in time.[6] Tests were performed using a content-based mesh, a regular

[5] Namely *Flower Garden*, *Football*, *Foreman*, *Stefan*, *Crowd*, *Edinburgh* and *Tennis.* The first four are standard test sequences, while the latter three are proprietary BBC sequences.

[6] The distance between the current frame and the two reference frames was chosen as two frame intervals in either direction. This allows for a reasonable amount of motion between a frame and its two reference frames.

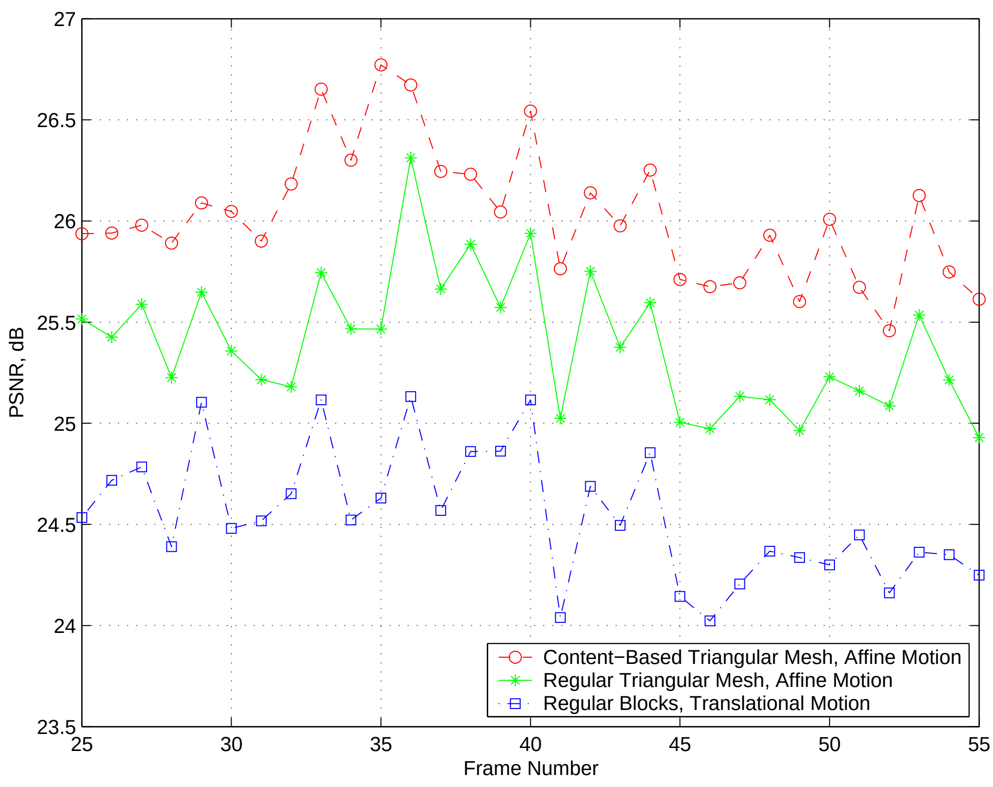


(a) Luma PSNR values of the motion compensated prediction error. Each (compensated) frame, $\tilde{I}_n$ has been motion compensated from the original frames I_{n-2} and I_{n+2}. A translation search radius of ± 31 pixels and an affine search radius of $\pm$ 1 pixel were used, both with half-pixel accuracy. Polygon approximation was applied to regions using $d_{max} = 4$.

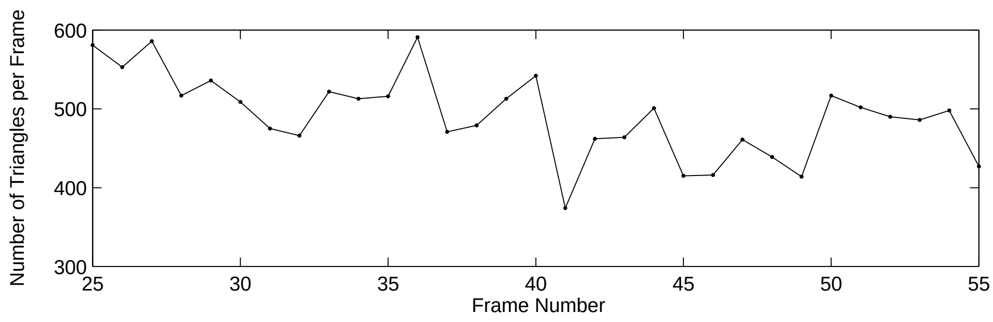


(b) The number of content-based triangles per frame. (For each frame, the number of regular triangles and blocks was set to be similar to the number of content-based triangles.)

Figure 4.4: Motion compensation of frames 25 to 55 of *Stefan*



(a) Luma PSNR values of the motion compensated prediction error. Each (compensated) frame, $\tilde{I}_n$ has been motion compensated from the original frames I_{n-2} and I_{n+2}. A translation search radius of ±63 pixels and an affine search radius of ± 3 pixel were used, both with half-pixel accuracy. Polygon approximation was applied to regions using $d_{max} = 4$.



(b) The number of content-based triangles per frame. (For each frame, the number of regular triangles and blocks was set to be similar to the number of content-based triangles.)

Figure 4.5: Motion compensation of frames 25 to 55 of *Football*

mesh with a similar number of triangles, and traditional block-matching.[7]

For both of the sequences, it is evident that more accurate motion compensation can be achieved using a content-based mesh, as opposed to a regular one with a similar number of triangles. In addition, a further gain is possible through the use of affine motion (compared to purely translational motion). The resulting improvement in quality depends largely on the type of motion present (e.g. translation, zoom, rotation) as well as on the accuracy of the object segmentation.

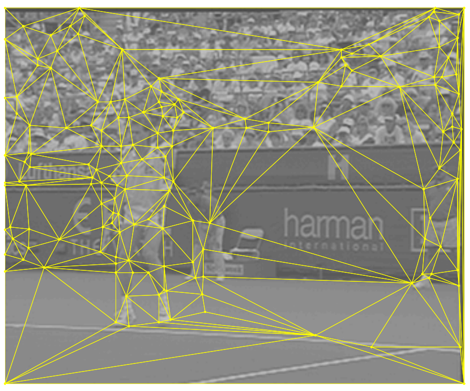
In the case of *Stefan*, the average improvement offered by affine motion compensation over purely translational block-matching is 0.4 dB. (Note that this is for an affine search radius of just ± 1 pixel at each triangle vertex.) An *additional* increase of more than 1.4 dB was obtained through the use of a content-based mesh (with affine motion compensation). This substantial gain is due to the tennis player in the foreground being motion compensated separately from the background region when using a content-based mesh.

The use of regular triangles (or blocks) results in erroneous motion vectors for those triangles (blocks) spanning both the foreground and background. Figure 4.6 illustrates this point for frame 51 of the sequence. In Figure 4.6(c), regular triangular shaped artifacts are particularly visible around the player's head. These are very similar to traditional blocking artifacts which occur when blocks straddle motion boundaries. This is less evident in Figure 4.6(d), where the content-based structure of the mesh has helped to preserve the player-background boundary.

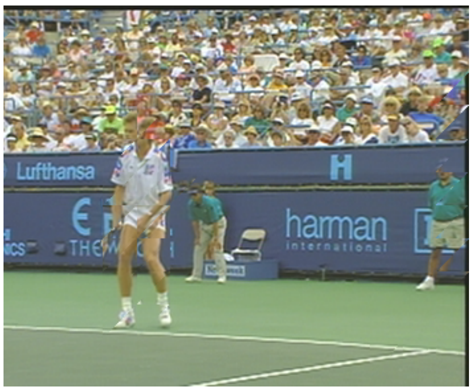
In the *Football* sequence, the advantage offered by affine motion compensation is more noticeable. A regular triangular mesh (with affine motion) provides an average gain of more than 0.8 dB over translational block-matching. This can

[7] Note that for each frame, the content-based mesh was created first, and following this a regular mesh with a similar number of triangles was generated. The number of triangles was also used to determine the (approximate) number of blocks used in block-matching. Motion vectors were calculated to half-pixel accuracy.

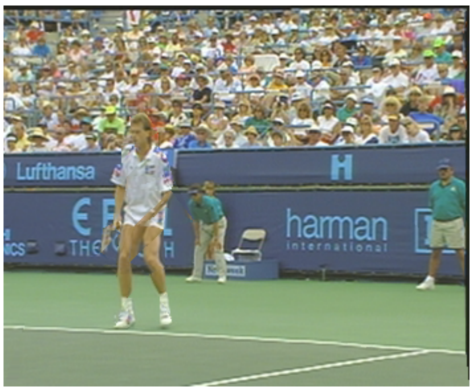
(a) Frame 51 (original) of *Stefan*



(b) A content-based mesh (314 triangles)



(c) Frame 51 motion compensated (from frames 49 and 53) using a regular triangular mesh with 308 triangles. PSNR$_Y$ = 30.67 dB

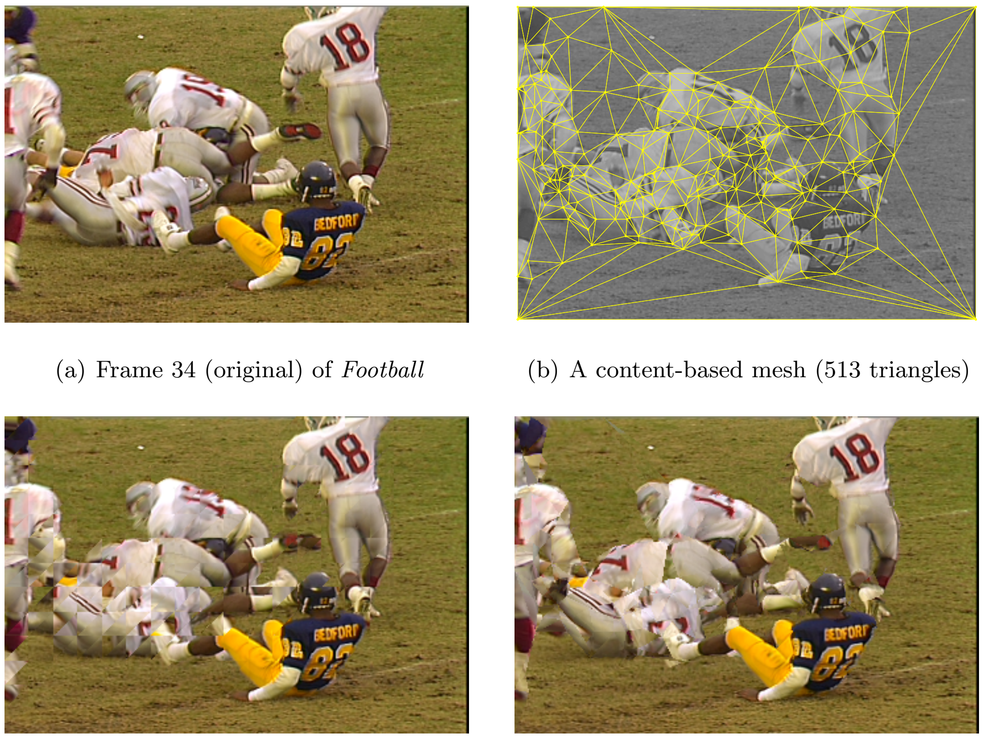


(d) Frame 51 motion compensated (from frames 49 and 53) using the content-based triangular mesh in (b). PSNR$_Y$ = 33.31 dB

Figure 4.6: Affine motion compensation of frame 51 of *Stefan*, using a regular and a content-based mesh.

be ascribed to the camera zoom and object rotation present in the scene, which can be modelled more accurately using an affine transform. An additional mean improvement of over 0.6 dB is achieved when using a content-based mesh.

A subjective improvement in quality is also evident in Figure 4.7. When using a regular triangular mesh, triangle-shaped artifacts are clearly visible, as shown in the bottom left quarter of Figure 4.7(c). It is evident from Figure 4.7(b) that the content-based approach results in triangles being clustered more densely in regions of complex motion, with larger triangles spanning the background. This allows for more accurate motion compensation, as illustrated in Figure 4.7(d).



(a) Frame 34 (original) of *Football*

(b) A content-based mesh (513 triangles)

(c) Frame 34 motion compensated (from frames 32 and 36) using a regular triangular mesh with 494 triangles. $\text{PSNR}_Y = 25.47$ dB

(d) Frame 34 motion compensated (from frames 32 and 36) using the content-based triangular mesh in (b). $\text{PSNR}_Y = 26.30$ dB

Figure 4.7: Affine motion compensation of frame 34 of *Football*, using a regular and a content-based mesh.

The use of a content-based mesh as well as an affine model allows for more accurate modelling of motion than regular block matching. However, these methods do result in an increased overhead. In a practical video coding system, the mesh design process proposed in this chapter would require the polygon boundaries to be transmitted to the decoder.[8] Likewise, specifying the affine motion parameters for each triangle can be costly. Thus for a fairer comparison, the rate-distortion behaviour of the various methods would need to be considered.

[8] For example, the average cost of encoding the shape information for the 31 frames of *Stefan* in Figure 4.4 is fairly low at 0.012 bpp (or 8.2 bits per triangle); In the case of the 31 frames of *Football* in Figure 4.5, the average cost of approximating the segmentation map is more substantial at 0.053 bpp (or 9.1 bits per triangle)

One problem with the proposed mesh-based approach is the triangle generation process within each polygon-shaped region. As described in Section 4.3.3, Delaunay triangulation is used to create triangles within each polygon. Thus the number of triangles within a polygon depends only on the *shape* of that polygon. This means that the number of triangles in a mesh can only be controlled by varying d_{max} (or thresholds T_1 and T_2, although these were set to fixed values, as described in Section 4.3.2). Two examples of the difficulty in controlling the number of triangles are shown in Figures 4.4(b) and 4.5(b). In the case of the *Stefan* sequence particularly, the number of triangles per frame varies substantially - between 42 and 378. (Note that each frame was obtained using polygon approximation with $d_{max} = 4$.)

4.6 Conclusion

Regular triangles (or blocks) can often cover an area occupied by two or more objects moving relative to one another. In this case, it is not possible to represent different types of motion accurately with just one set of (translation or affine) parameters. The method for designing a content-based mesh proposed in this chapter attempts to prevent this problem by positioning triangles in such a way that they do not cross motion boundaries. Such a mesh typically has many small triangles in regions of significant motion, with much larger triangles covering regions of little or no motion.

The use of a non-connected mesh enables the motion parameters for each triangle to be calculated independently of other triangles, and thus allows for easy comparison with a regular mesh or a block-based approach. However, one idea worth consideration is using a connected mesh within each polygon-shaped region. This may lead to improved subjective quality by ensuring continuous motion within each object, while reducing the cost of encoding the motion parameters. Additional gains are also likely if a mesh is allowed to evolve from one frame to the next (instead of being coded separately for each frame). This should allow a significant reduction in the number of bits required to encode a mesh's structure across a group of frames.

The results reported above demonstrate that the use of a content-based triangular mesh for affine motion compensation provides some advantages over traditional block-based methods. When using the same number of triangles/blocks per frame, the resulting improvement in image quality was demonstrated both objectively and subjectively for two test sequences with significant amounts of motion. However, it is important to consider the overhead involved in coding the shape information. Thus if the proposed mesh-based motion compensation method is to be considered for use in a video codec, its performance against regular block matching should be measured at the same overall rate - taking both shape and motion information into account.

One problem with the mesh-based motion compensation approach presented in this chapter is that it is difficult to control the number of triangles in the mesh. The number of triangles depends on the number and shape of regions in the segmentation map, as well as the polygon approximation parameter, d_{max}. The number and shape of regions are in turn dependent upon the scene content as well as the parameters T_1 and T_2 (which were set to fixed values). Allowing some variation in these two thresholds would enable greater scalability and control over the segmentation process, although their precise effect would need to be further investigated for a variety of different sequences. Using the d_{max} parameter for direct control of the number of triangles in the mesh is fairly crude, as was shown to be the case for the *Stefan* sequence (Figure 4.4(b)). The problem with a large variation in the number of triangles per frame is that there is a corresponding effect on the number of motion parameters to be coded. From a rate-distortion point of view, the density of triangles within a region should be proportional to the complexity of motion within the region, rather than being dependent on the object's shape.

The next chapter discusses a motion compensation method which attempts to address this issue. Rather than making the number of triangles dependent on an object's shape, a block-based scheme is used in which the number of blocks within a region is proportional to the motion compensation error for that region.

Chapter 5

Motion Compensation using Region-Based Block Matching

5.1 Introduction

Block matching methods for motion estimation are fast, easy-to-implement, and provide reasonable results across a wide range of sequences. They also have the advantage of combining well with block-based techniques for the coding of residuals, such as the Discrete Cosine Transform.

In general though, block boundaries do not coincide with motion boundaries, which means that a block may contain regions corresponding to more than one type of motion. This makes compensating for motion in such a block a difficult task. Another characteristic of traditional fixed-size block matching (FSBM) is that the block size is constant. However, it seems more intuitive to allow larger blocks in regions of uniform motion, and smaller blocks where there is more complex motion.

Variable-size block matching (VSBM) allows large blocks in areas of complex motion to be split into smaller blocks (quad-tree partitioning is normally used).

Some material in this chapter is based on the author's previously published work [101].

This chapter describes a *region-based* approach to block matching, in which the size of blocks varies from one region to the next.

A frame is first segmented into regions of approximately homogeneous motion (relative to one or two reference frames). Following this, a rectangular grid of blocks is generated within each region, with the size of the blocks in a region being inversely proportional to the mean square error (after motion compensation). Finally, motion estimation is performed in order to determine the translational and affine motion parameters for each block.

5.2 Related Work

In recent years, VSBM has gained in popularity over FSBM, and as a result of its superior rate-distortion performance, it has been incorporated into the H.264/AVC video coding standard. The use of variable-size blocks allows for greater adaptivity to local scene content. In a similar way, region-based methods enable objects in a scene to be coded independently of one another. This section provides a brief review of *region-based* block matching methods.[1]

- The MPEG-4 video coding standard [35] populates each Video Object Plane (VOP) with a regular grid of 16×16 macro-blocks. (When using the advanced prediction mode matching is performed using four 8×8 blocks per macro-block.) FSBM is used to determine the motion vector(s) for each block. Blocks along the boundary of a VOP usually contain some pixels outside the region. In this case, matching is performed using only those pixels inside the object mask. (Note that object segmentation and coding of any shape information is performed prior to the motion estimation stage.)

- Martin *et al* [63, 79] consider the use of variable-size blocks when estimating the motion within a region. They show that a quad-tree implementation of VSBM can lead to improved motion compensation, particularly

[1] A more general summary of FSBM and VSBM methods is provided in Section 6.2.

around the edges of objects. However, they also describe how VSBM provides little gain for certain objects undergoing complex motion.[2] They therefore propose a modified version which allows each set of four sibling nodes in a quad-tree to be merged in 14 possible ways (rather than being limited to just the one possible four-to-one merge in VSBM). This modified merging process enables the creation of non-square regions, which in turn helps to reduce the number of motion vectors required for motion compensation.

- Zhang *et al* [124, 125] describe how a quad-tree structure can be used to partition a frame into regions with different motion characteristics. Starting with the smallest possible blocks in the quad-tree, sibling blocks are merged in a way that optimises the rate-distortion performance. In addition to the standard four-to-one merging of four sibling blocks, an additional ten possible merging patterns are possible. A coding technique is used whereby regions with the same motion vector are combined, thus reducing the number of motion vectors to be coded. The proposed Region-Wise Motion Compensation (RWMC) method also incorporates the coding of motion vectors using temporal prediction from previously coded frames. Results demonstrate that RWMC out-performs VSBM on a variety of sequences.[3] RWMC achieves most of its gains by merging large background areas that can be motion compensated with a common motion vector.

[2] When a region undergoes complex motion, it is likely that small blocks will be used *throughout* the region. Consequently, a significant number of bits are then spent coding the quad-tree. This negates much of the advantage of VSBM, which is most effective when the optimal block size varies across the region/frame.

[3] This advantage is perhaps exaggerated by the fact that the implementation of VSBM used in testing does not use predictive coding of motion vectors.

5.3 Region Segmentation

From a motion compensation (and video coding) perspective, it is more advantageous to split regions along *motion* boundaries than spatial ones, since a scene can then be partitioned into regions of homogeneous motion. However, motion boundaries are often difficult to determine precisely, so spatial segmentation can be used to provide a good starting point, particularly since spatial and motion boundaries often coincide.

5.3.1 Initial Segmentation and Error Surfaces

An initial spatial segmentation is obtained using JSEG [16, 17]. This segmentation process is performed on a single frame and does not take any motion information into account. In order to consider the motion characteristics of each region, an error surface $\mathbf{E}_{r,f}(u, v)$ is calculated. This is defined as the sum squared error (SSE) when region number r is motion compensated by translating the corresponding region in reference frame number f a distance (u, v) within a search window. Note that the translation is performed at integer pixel accuracy.

Equation 5.1 shows the formula for calculating the SSE between the current frame, I, and a reference frame, I_f, for all points (x, y) in the region.

$$\mathbf{E}_{r,f}(u, v) = \sum_{(x,y) \in Region_r} [I(x, y) - I_f(x + u, y + v)]^2 \qquad (5.1)$$

For each region, the translational motion vector can be determined by finding the pair (u, v) that minimises $\mathbf{E}_{r,f}(u, v)$. Note that if two reference frames are used (e.g. one past and one future), then two error surfaces per region need to be calculated. Figure 5.1 shows a frame segmented into three regions, and the error surfaces for each region.

In the previous chapter, the method adopted for estimating the motion characteristics of each region was to calculate the optical flow for the current frame (relative to a reference frame). Using the optical flow, it was then possible to

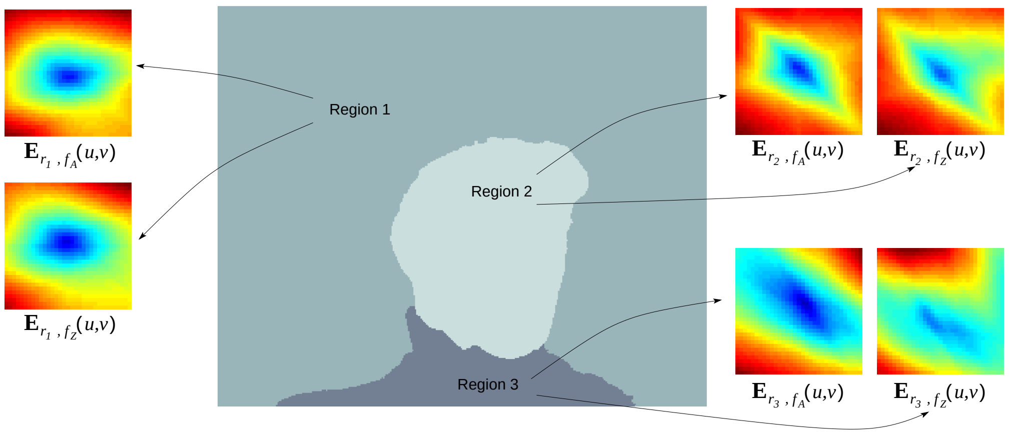

Region 1
Region 2
Region 3
$\mathbf{E}_{r_1,f_A}(u,v)$
$\mathbf{E}_{r_1,f_Z}(u,v)$
$\mathbf{E}_{r_2,f_A}(u,v)$
$\mathbf{E}_{r_2,f_Z}(u,v)$
$\mathbf{E}_{r_3,f_A}(u,v)$
$\mathbf{E}_{r_3,f_Z}(u,v)$

Figure 5.1: An example of a frame segmented into three regions. For each region, two translation error surfaces are shown. $\mathbf{E}_{r,f_A}(u,v)$ represents the error surfaces calculated relative to the preceding reference frame, while $\mathbf{E}_{r,f_Z}(u,v)$ shows the error surfaces corresponding to the subsequent reference frame. Areas of low error are shaded in blue, while higher errors are coloured red.

determine the mean and variance of motion within each region. In contrast, the error-surface approach outlined above reveals how well a region in the current frame matches translated versions of that region in the reference frame(s). One advantage of the latter method is that in addition to providing the optimal (pixel-accurate, translational) motion vector for each region, it also yields a measure (i.e. the SSE) of how effective this motion vector is in minimising the displaced frame difference (DFD) for a region.

5.3.2 Region Splitting

The error surface(s) calculated for each can be used to determine if the region should be segmented further. For a region r, the *minimum SSE*, $E_{min}(r)$, is defined as the smallest value of $\mathbf{E}_{r,f}(u,v)$ across the range of possible translations and reference frames. Assuming two reference frames, f_A and f_Z, the minimum SSE is calculated as

$$E_{min}(r) = min\left\{min(\mathbf{E}_{r,f_A}), min(\mathbf{E}_{r,f_Z})\right\} \tag{5.2}$$

If a region corresponds to an area of homogeneous (translational) motion, then its minimum SSE will be relatively low. Using this principle, the next stage in the segmentation process is as follows: Segment the region with the largest value of $E_{min}(r)$ further, and repeat the process until either all regions have an acceptable minimum SSE, or a maximum number of regions is reached. It was found experimentally that a reasonable threshold value for $E_{min}(r)$ varied significantly from one sequence to next. As a simple rule-of-thumb, it was therefore decided to perform region splitting a total of 50 times.[4]

It should be noted that large regions are more likely to have a large SSE value, since this metric is based on the total (and not the mean) square error. As a result, some very large regions may end up being segmented even if their motion is relatively uniform.[5] The reason that the SSE rather than the MSE is used as a criterion for splitting regions, is that using the latter measure can lead to small regions with complex motion being over-segmented.

Several methods were considered for splitting already-segmented regions: For example, it is possible to use the motion-compensation error as a guide to determining motion discontinuities. However, it was generally found to be as effective to use JSEG to further split a region.

5.3.3 Region Merging

Following this splitting process, regions are considered for merging by comparing their joint motion compensation error. Two neighbouring regions (r_i and r_j) are assumed to have similar motion (and can thus be merged) if:

$$\begin{aligned} \exists\ f_0 \text{ such that} \quad min(\mathbf{E}_{r_i,f_0}) &\leq min(\mathbf{E}_{r_i,f}) \\ \text{and} \quad min(\mathbf{E}_{r_j,f_0}) &\leq min(\mathbf{E}_{r_j,f}) \quad \forall \text{ ref. frames } f \end{aligned} \tag{5.3}$$

[4] It was found empirically that partitioning more than 50 times was unnecessary, since these extra regions would almost always be re-joined during the merging stage.

[5] However, as is explained in the next section, regions with similar motion characteristics can be re-merged.

and

$$min(\mathbf{E}_{r_i,f_0} + \mathbf{E}_{r_j,f_0}) = min(\mathbf{E}_{r_i,f_0}) + min(\mathbf{E}_{r_j,f_0}) \tag{5.4}$$

In order to satisfy Equation 5.3, the same reference frame (f_0) must result in the minimum motion compensation error for both of the neighbouring regions.[6] Equation 5.4 requires that the additional error resulting from merging these two regions must be zero. Together, these two equations imply that the (pixel-accurate) translational motion vectors of the joint region and the two individual regions will be identical.

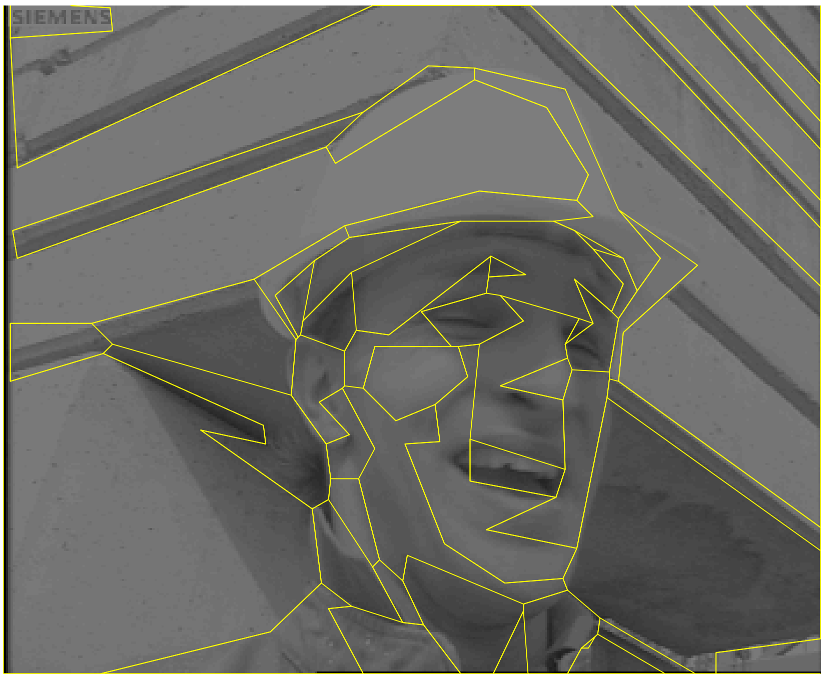
It is worth noting that the SSE of two merged regions is equal to the sum of their two individual SSE's, in other words $\mathbf{E}_{(r_i \cup r_j),f} = \mathbf{E}_{r_i,f} + \mathbf{E}_{r_j,f}$. This allows for a new error surface to be calculated in a simple and fast way from those of its constituent regions (as is the case on the left hand side of Equation 5.4).

The process of merging regions helps to ensure that there are no unnecessary boundaries between areas with similar motion characteristics. This decreases the number of regions in the segmentation map and consequently reduces the cost of coding the region boundary information.

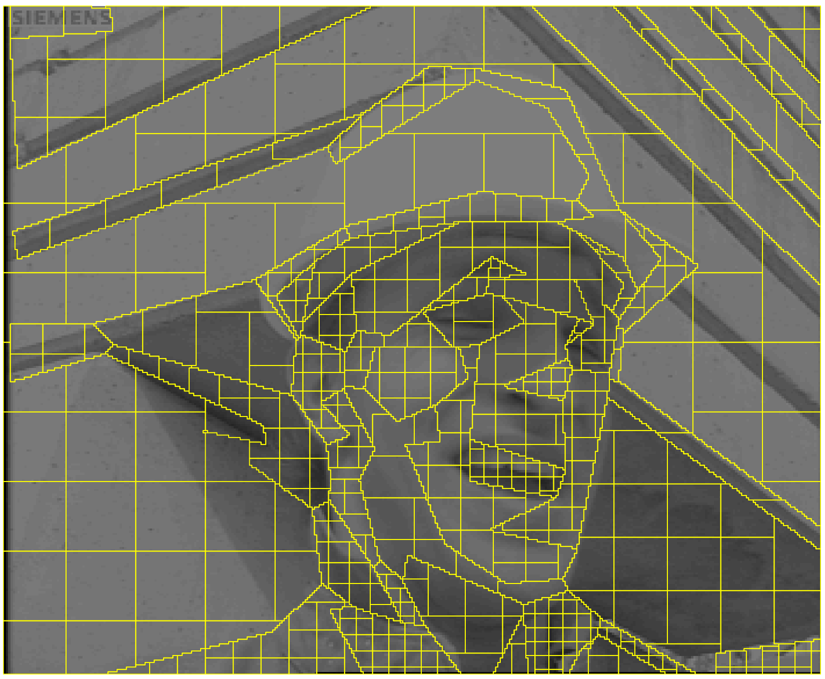
5.4 Variable-Size Block Generation

After splitting and merging, the segmented regions are approximated by polygons in order to allow the shape information to be encoded efficiently. The progressive polygon coding method described in Chapter 3 is used to achieve this. Figure 5.2(a) shows an example of regions approximated by polygons for a frame from the *Foreman* sequence. There is a tradeoff between the accuracy of the polygon approximation and the overhead required to encode the shape information. A value of $d_{max} = 8$ was found to provide a reasonable compromise for the (CIF resolution) sequences tested. This allows the polygon-approximated boundary of a region to vary by a distance of up to eight pixels from the actual boundary.

[6] If only one reference frame is available, then the f_0 in Equation 5.3 is this reference frame.



(a) A polygon approximation of segmented regions



(b) A grid of blocks is generated within each region

Figure 5.2: Generation of region-based variable-size blocks

Within each region, a square grid of blocks is generated. It seems reasonable that the size of blocks in different regions should vary according to the type of motion within each region. In this way, regions with fairly complex motion can be populated by relatively small blocks, while large blocks can span regions of more or less uniform motion. This process allows the allocation of more resources (i.e. motion vectors) to those parts of the picture where they are most needed.

Using the above principle, the number of blocks in a region is chosen to be proportional to the minimum SSE, $E_{min}(r)$, for that region. This ensures that regions with a relatively large *mean* square error (after translational motion compensation of the regions) are populated by smaller blocks, as illustrated in Figure 5.2(b). The size of blocks within a region, r, is set to $S_B(r) \times S_B(r)$, where $S_B(r)$ is given by:

$$S_B(r) = \sqrt{\frac{A(r)}{N_B \; E_{min}(r)} \sum_{r_i=1}^{N_R} E_{min}(r_i)} \tag{5.5}$$

where $A(r)$ is the size of region r, N_R the number of regions in the frame, and N_B the target number of blocks for the frame.

Upper and lower limits are placed on $S_B(r)$. A maximum block size of 64×64 is permitted, so as to restrict large-scale blocking artifacts. In addition, a minimum block size of 4×4 is allowed, in order to prevent spurious matching and to limit the relatively high motion-vector overhead associated with small blocks.

As can be seen in Figure 5.2(b), there are many blocks which lie along the boundary of a region, and as a result are non-square. Some of these non-square blocks can be very small in size (relative to other blocks in their region). Thus, in order to limit the number of small non-square blocks, some[7] of these blocks on the edge of a region are merged with a neighbouring block in the same region.

[7] In region r, edge blocks that have an area less than $\frac{1}{3}S_B(r) \times S_B(r)$ are combined with a neighbouring block in the same region.

In addition to coding the segmentation map, the size of blocks in each region needs to be transmitted to the decoder. This can be done very efficiently, since it requires less than six bits per region to encode the value of $S_B(r)$.

5.5 Motion Estimation

Motion estimation is performed in two stages for each block. First, a translation motion vector is estimated using traditional block matching,[8] as illustrated in Figure 5.3(a). This is followed by estimating the six affine motion parameters for each block. The advantage of affine motion compensation is that it allows rotation, zooming and shearing to be represented as well [97].

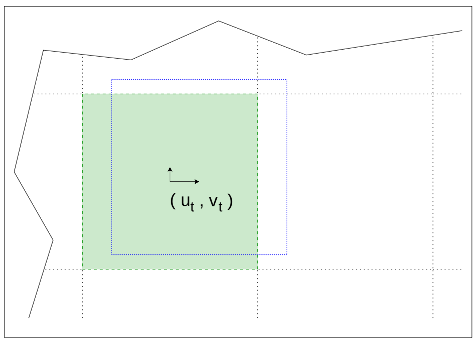
In order to estimate the affine motion of a block, an equilateral triangle is positioned around the centroid of the block in the current frame (and the centroid of the translated block in the reference frame). The distance from the centroid to each of the three vertices is $\frac{1}{2}S_B(r)$, where $S_B(r) \times S_B(r)$ is the area of blocks in that region. An affine search is performed using a similar method to that described in Section 4.4.2, except that matching is applied to the block as a whole (and not just the triangular area at its centre). The procedure for affine motion estimation is illustrated in Figure 5.3 and outlined below:

- For each of the three triangle vertices in the *current* frame, $\{(x'_i, y'_i) : i \in \{1, 2, 3\}\}$, the corresponding vertices in the *reference* frame, $\{(x_i, y_i)\}$ are related according to the equation:

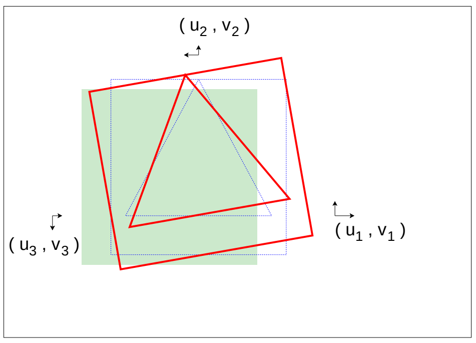
$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} x'_i \\ y'_i \end{bmatrix} + \begin{bmatrix} u_t \\ v_t \end{bmatrix} + \begin{bmatrix} u_i \\ v_i \end{bmatrix} \tag{5.6}$$

  where (u_t, v_t) is the (optimal) translation motion vector calculated during the previous stage, and $\{(u_i, v_i) : i \in \{1, 2, 3\}\}$ are the *additional* displacements of the three triangle vertices in the reference frame.

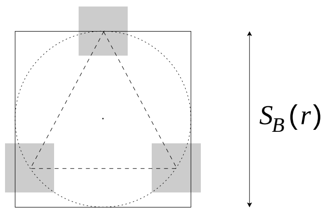
[8] The translation motion vector for the *region* containing a block provides a good starting point when estimating the motion of that block.



(a)



(c)



(b)

(a) The green (shaded) area shows a block within the *current* frame. The blue (unshaded) square represents the position of the best matching block in the *reference* frame, assuming translational motion only.

(b) For each block, affine motion is estimated by positioning an equilateral triangle around the centroid of the block. Each vertex is then perturbed within a small search area (shaded grey) in order to find the affine motion parameters which result in the smallest error.

(c) The red (thick) block shows the position of the best matching block found in the reference frame, determined using affine motion estimation. The three affine motion vectors are also indicated.

Figure 5.3: Translation and Affine Motion Estimation

- During the affine motion estimation stage, (u_i, v_i) is varied within a small search range for each of the three triangle nodes. Note that consequently $\{(x_i, y_i)\}$ is known for each such variation. This is because $\{(x'_i, y'_i)\}$ and (u_t, v_t) are constant for each triangle (block), the latter having been determined during the translation estimation stage.

- In general, for a pixel (x', y') in a block in the current frame, the corresponding point (x, y) in the reference frame is given by the affine transform:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 + u_t \\ a_4 & a_5 & a_6 + v_t \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \tag{5.7}$$

  where $\{a_1, \ldots, a_6\}$ are the six affine motion parameters. These can be determined by considering Equation 5.7 in the case of the three triangle vertices. This yields the system of equations:

$$\underbrace{\begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} x'_1 & y'_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x'_1 & y'_1 & 1 \\ x'_2 & y'_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x'_2 & y'_2 & 1 \\ x'_3 & y'_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x'_3 & y'_3 & 1 \end{bmatrix}}_{\mathbf{B}} \underbrace{\begin{bmatrix} a_1 \\ a_2 \\ a_3 + u_t \\ a_4 \\ a_5 \\ a_6 + v_t \end{bmatrix}}_{\mathbf{a}} \tag{5.8}$$

  for which the positions of the vertices $\{(x_i, y_i)\}$ and $\{(x'_i, y'_i)\}$ are known. Solving for $\mathbf{a}$ yields:

$$\mathbf{a} = \mathbf{B}^{-1}\mathbf{x} \tag{5.9}$$

In the case of a full search, each of the triangle vertices should be warped simultaneously. However this is computationally expensive. A faster, sub-optimal approach (which still provides good results) is to perturb each vertex in turn, while keeping the other two fixed. As in the case of mesh-based affine motion estimation, it was found that a good compromise can be achieved by warping each of the three vertices in turn, followed by a final refinement of the first vertex.

For each block, the translation motion estimation stage yields a motion vector pair plus a flag indicating which reference frame results in a smaller error. The affine motion estimation stage provides an additional three motion vector pairs - one per triangle vertex. In general, the translation search radius is much larger than the affine search radius.

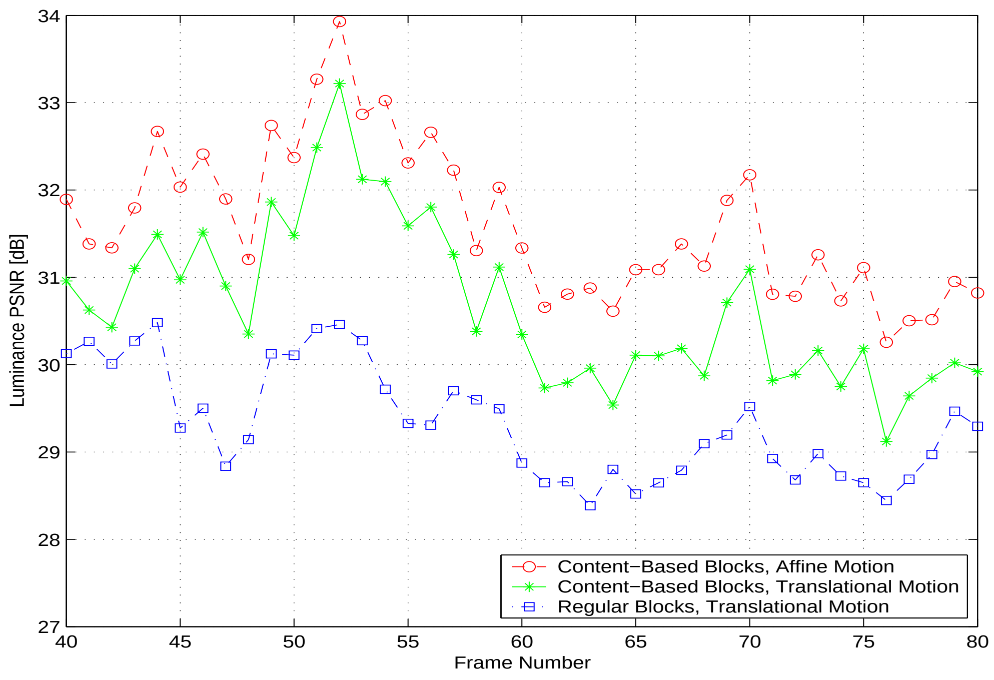
5.6 Results

The performance of the proposed region-based VSBM method was tested on seven[9] test sequences, using both regular and content-based blocks. This section presents results for the *Stefan* (CIF) and *Foreman* (CIF) sequences, both of which contain significant motion. (Note that the results shown here are representative of those obtained for the full set of seven sequences.)

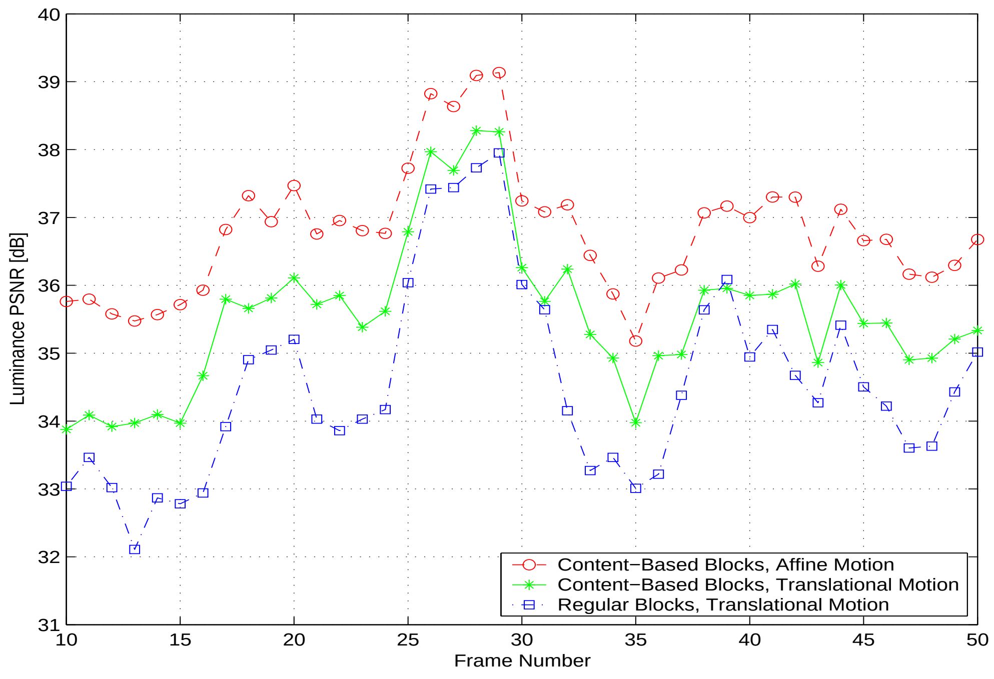
Regular and variable-size blocks were used, and both translation and affine motion compensation were performed. For both of the sequences, each (compensated) frame, $\tilde{I}_n$, was motion compensated from the original frames I_{n-2} and I_{n+2}. A translation search radius of ± 31 pixels, and an affine search radius of ± 3 pixels were used, both with half-pixel accuracy. The number of blocks per frame (N_B in Equation 5.5) was chosen as 396, which is equivalent to 16×16 blocks.

Figure 5.4(a) presents the results obtained from 40 frames of the *Stefan* sequence. It can be seen that the use of content-based blocks offers a mean improvement in quality of more than 1.3 dB, when compared to regular block matching with the same number of blocks. Using affine motion compensation for each block provides an additional average gain of almost 1 dB.

[9] Namely *Flower Garden*, *Football*, *Foreman*, *Stefan*, *Crowd*, *Edinburgh* and *Tennis*. The first four are standard test sequences, while the latter three are proprietary BBC sequences.

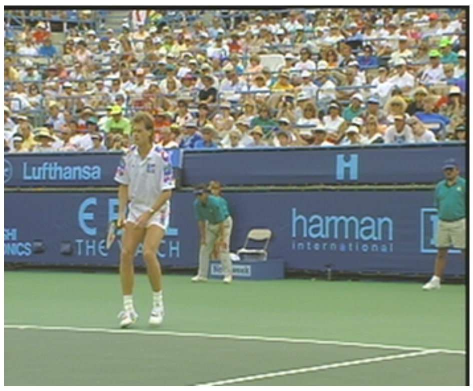


(a) Motion compensation of frames 40 to 80 of *Stefan* (CIF resolution)

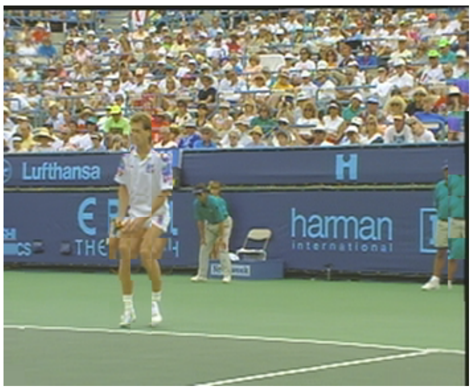


(b) Motion compensation of frames 10 to 50 of *Foreman* (CIF resolution)

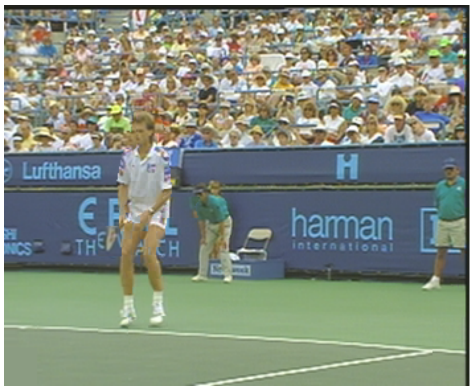
Figure 5.4: Mesh generation using polygon approximation and triangulation



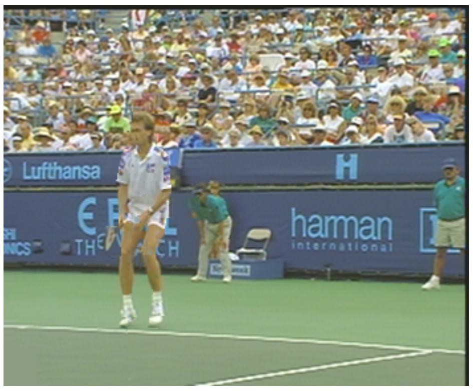
(a) Original frame.



(b) Translational motion compensation using regular blocks; PSNR$_Y$ = 30.46 dB



(c) Translational motion compensation using variable-size content-based blocks; PSNR$_Y$ = 33.22 dB

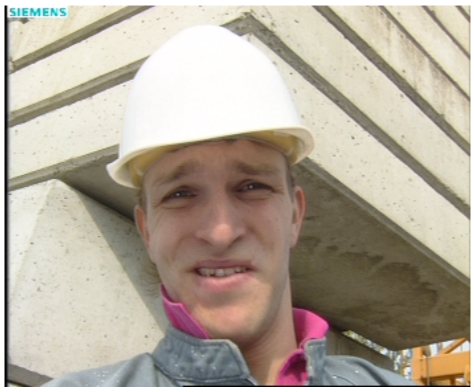


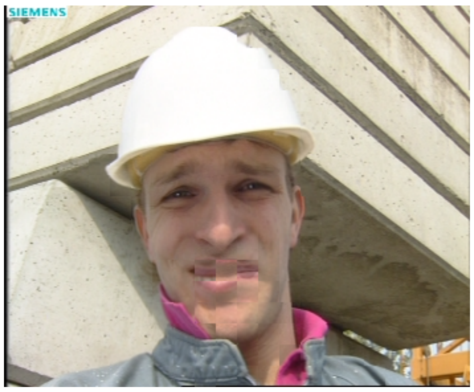
(d) Affine motion compensation using variable-size content-based blocks; PSNR$_Y$ = 33.93 dB

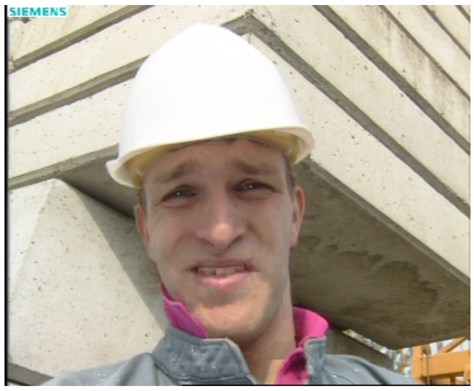
Figure 5.5: Frame 52 of the (CIF-resolution) *Stefan* sequence, motion compensated from frames 50 and 54 (using 396 blocks).

Similar results were obtained for the 40 frames of the *Foreman* sequence presented in figure 5.4(b). The graph shows that the use of region-based variable-size blocks provides an average increase of 1 dB (and affine motion an additional 1.2 dB) over fixed-size translational block matching.

As a guide to subjective performance, a frame from *Stefan* is shown in Figure 5.5 after having been motion compensated. Traditional blocking artifacts are clearly visible in Figure 5.5(b), particularly around the player's legs, as well as along the right-hand edge of the frame. When using region-based VSBM,


(a) Original frame.


(b) Translational motion compensation using regular blocks; PSNR$_Y$ = 35.35 dB


(c) Translational motion compensation using variable-size content-based blocks; PSNR$_Y$ = 35.87 dB

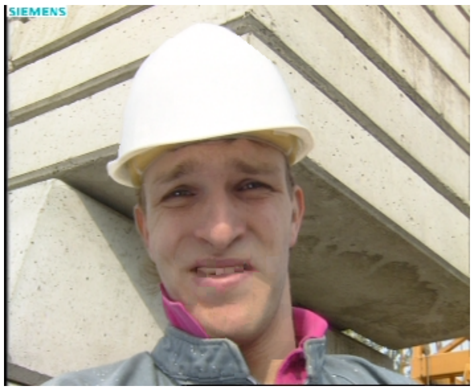

(d) Affine motion compensation using variable-size content-based blocks; PSNR$_Y$ = 37.30 dB

Figure 5.6: Frame 41 of the (CIF-resolution) *Foreman* sequence, motion compensated from frames 39 and 43 (using 396 blocks).

the boundary between the player and the background is preserved much more accurately, though there is some distortion present on the left of the player's head (see Figure 5.5(c)). The additional use of affine motion compensation helps to reduce this error somewhat, as can be seen in Figure 5.5(d).

Motion compensation results for a frame from *Foreman* are shown in Figure 5.6. The artifacts due to regular (fixed size) block matching are clearly evident around the mouth in Figure 5.6(b). Blocking artifacts are much less noticeable in Figure 5.6(c), which shows the results when region-based variable-size blocks are used. Upon closer examination, distortion is still visible at block boundaries, though on a much smaller scale. It can be seen (in Figure 5.6(d)) that the use of affine motion compensation results in a smoother reconstruction since actual motion can be represented more accurately. The slight discolouration around the bottom of the lower lip in 5.6(c) and 5.6(d) is probably due the fact that only the luma component was used when performing block matching.

From a video coding point of view, there is clearly an overhead associated with encoding the polygon shape and block size information. Likewise, there is a cost of specifying the affine motion parameters. For the forty frames of *Stefan* and *Foreman*, the overhead of coding the region shape was on average 0.019 and 0.020 bits per pixel, respectively. (This shape information was coded using the progressive polygon approximation method described in Chapter 3.)

5.7 Conclusion

The results presented in this chapter demonstrate the improvement in quality offered by the use of region-based variable-size blocks (over regular blocks) and affine motion (as opposed to purely translational motion) when performing motion compensation.

Region-based variable-size blocks offer two advantages: First, blocks do not straddle motion boundaries, since they are confined to regions of (approximately) homogeneous motion. This reduces blocking artifacts by allowing for more accurate motion compensation. Secondly, block size is varied from one region to another so that regions with complex motion tend to have smaller blocks. In this way, resources (motion vectors) are allocated to different regions according to their need, which helps to reduce the overall motion compensation error. In addition, the use of affine parameters allows for a more realistic modelling of motion. Nevertheless, it is important to consider the overhead involved in coding the region shape or the additional affine parameters. For a fair comparison, one should compare the performance of the region-based method to regular block matching at the same overall bit-rate.

In contrast to the mesh-based approach presented in Chapter 4, the region-based method described in this chapter enables the number of blocks (and motion vectors) per frame to be controlled precisely. This is a significant advantage when trying to achieve a rate-distortion optimisation of the motion compensation process (for a given segmentation map).

Despite this improvement, the spatio-temporal segmentation process allows little control over the number and shape of regions. This can be a problem when considering the use of region-based variable-size block matching in a rate-distortion optimised video codec. When used for video coding, region-based block matching needs to be incorporated within an optimal rate-distortion framework, to ensure that cost of specifying region shape and block size is offset by an improvement in quality.

If a segmentation map contains only a few regions, then it will not require many bits to code this information, but it is unlikely to provide much of an advantage over regular block matching. Conversely, if a frame is over-segmented, then it will enable good localisation of motion, but at the expense of spending a large number of bits on coding the segmentation map.

The next chapter considers an approach to overcoming this problem by combining the segmentation and block generation processes. By coupling these two stages, it becomes easier to incorporate rate-distortion optimisation into the process.

Chapter 6

Motion Compensation using a Binary Partition Tree of Blocks

6.1 Introduction

Motion compensation techniques are an important part of almost all video codecs since they provide an effective way of exploiting the temporal redundancy between frames in an image sequence.

Traditionally, Fixed-Size Block Matching (FSBM) has been used to determine the motion of each block in the current frame relative to the reference frame(s). In recent years, Variable-Size Block Matching (VSBM) has gained in popularity, since it enables the size and shape of blocks to adapt (to a limited degree) to scene content. Most VSBM implementations generate either a binary-tree or a quad-tree of blocks, with smaller blocks predominating in areas of complex motion.

This chapter describes a VSBM approach which allows the binary splitting of blocks. However, blocks are not necessarily split into two equal halves. Instead, blocks are split along the horizontal or vertical line that achieves the maximum reduction in motion compensation error. This allows for partitioning a scene

Some material in this chapter is based on the author's previously published work [104].

along motion boundaries, which enables effective motion compensation using relatively few blocks. As a result, significant gains in rate-distortion performance are possible.

6.2 Related Work

6.2.1 Fixed-Size Block Matching

Block matching using fixed-size blocks has been arguably the most popular motion estimation technique for more than two decades. It is an important component in several video coding standards, including MPEG-1 and MPEG-2, in which 16×16 macro-blocks are commonly used.[1]

- Jain and Jain [41] were the first to propose the use of blocks for motion estimation. Their method involves dividing a frame into a grid of (normally square) blocks. For a range of possible motion vectors (within some search radius), a block's best match is found by matching the pixels in the block with the corresponding translated pixels in a reference frame. The metric used when determining the best match is typically either the sum of absolute error (SAE) or the sum of squared error (SSE).

- Ever since block matching was first proposed, there has been a substantial research effort into speeding up the search process when performing motion estimation. Another popular focus of research is estimating and compensating for motion at *sub-pixel* accuracy. For more information on these topics, the reader is referred to review papers on motion estimation [19, 109].

- Seferidis and Ghanbari [97] discuss the use of generalised block matching. They allow blocks to undergo a projective transform in order to model

[1] Note that block matching is usually performed using the 16×16 luma component of each macro-block.

complex motion. As a result, matching is improved, although the information overhead increases due to the additional motion parameters. (A total of four motion vectors are required per block.) Despite the additional overhead, generalised block matching has been shown to provide superior rate-distortion performance when compared to standard translational block matching. However, there is an increased computational cost due to the process of searching for the additional motion parameters.

- Ribas-Corbera and Neuhoff [89] investigate the impact of block size on the effectiveness of block matching. Small blocks provide the advantage of allowing complex motion to be represented. However, this leads to an increase in the number of motion vectors - and consequently the motion vector bit-rate. This in turn reduces the number of bits available for coding the displaced frame difference (assuming a fixed bit budget). The optimal block size is shown to be dependent on several factors, in particular: motion vector precision, texture, inter-frame noise, and the type of motion present in the scene.

6.2.2 Variable-Size Block Matching

The use of VSBM allows for the size of blocks to adapt to scene content within small areas of an image. A variety of implementations of VSBM are possible, and several selected methods are outlined briefly below.

- Puri *et al* [86] describe a simple VSBM technique. Motion compensation is initially performed using a grid of 8×8 blocks. Those blocks that are deemed to have a poor match in the reference frame are partitioned into four 4×4 blocks, for which matching is performed. Even with only two possible block sizes, this method demonstrates advantages over FSBM.

- Chan *et al* [9] propose another top-down approach to VSBM. If a block's motion compensation error is large, then it split (horizontally or vertically) into two new rectangular blocks. These blocks are partitioned further if their motion compensation error remains high, and the process is repeated. This results in a binary tree of blocks, with smaller blocks being present in areas of complex motion. The leaf blocks of the binary tree are pruned in cases where splitting does not achieve any reduction in motion compensation error. This VSBM technique demonstrates significant gains over traditional block matching, at the price of increased encoder complexity.

- Sullivan and Baker [111, 112] describe how VSBM can be incorporated into a rate-distortion framework. Using a quad-tree block partitioning approach, they consider the reduction in error achieved by splitting a block, and also take into account the additional cost of coding the motion vector information for the new blocks.

- Seferidis and Ghanbari [98] extend generalised block matching to variable-size blocks. Using a quad-tree that allows block sizes from 32×32 down to 8×8, they demonstrate that the use of an eight-parameter projective motion model provides an improvement in rate-distortion performance. This gain was found to be most noticeable in the case of small blocks subject to complex (non-translational) motion.

- Kim and Lee [45, 46] discuss a hierarchical approach to VSBM. At each level of the quad-tree hierarchy, a block's motion vector is only coded if it differs significantly from that of its parent block. This hierarchical method allows for decisions on the coding of block motion vectors to be based on optimising rate-distortion performance.

- Zhang *et al* [126] describe the use of quad-tree VSBM in combination with a motion boundary segmentation method. The segmentation can be applied to each (variable size) block. It allows for each block to be split in two using a straight line of any orientation and position. Each half of the block can then be assigned its own motion vector. Using such a segmentation approach allows for motion boundaries to be approximated with straight lines, thus helping to ensure that each block represents an area of homogeneous motion. In terms of rate-distortion performance, the proposed method is superior to FSBM, but slightly inferior to regular VSBM. However, Zhang *et al* report that in terms of subjective picture quality, their method is superior to standard VSBM, when compared at the same rate.

- Chang *et al* [10] propose a variant of VSBM. Instead of allowing a block to be split into four equal-size child blocks (as in a quad-tree), all 14 possible combinations of these four blocks are possible. As a result, the partition tree can also comprise rectangular and L-shaped regions. This leads to a slightly increased overhead, since the type of partition at each node in the tree must be coded. However, because of the increased versatility of region shape, fewer motion vectors are required at each level in the partition tree. The proposed dynamic-shaped block matching method is demonstrated to provide superior rate-distortion performance to FSBM and both top-down and bottom-up implementations of VSBM, for "head and shoulder" type sequences.

- Rhee *et al* [88] consider two approaches to quad-tree VSBM. In particular, they focus on minimising the motion compensated prediction error for a given number of blocks (rather than for a given rate constraint). The first method (which is computationally expensive) uses dynamic programming in order to achieve the optimal quad-tree partitioning for the target number of blocks. The second technique determines a number of candidate motion vectors for each of the (fixed-size) smallest possible blocks. A bottom-up merging process is then used to repeatedly merge sets of four blocks that share a common parent and at least one common candidate motion vector. This approach is computationally much more efficient than the first (optimal) method, and suffers from only a relatively small loss in quality.

As demonstrated by the above methods, VSBM offers superior motion compensation performance over FSBM at the same overall rate, though at the price of greater complexity. As discussed in Section 2.7, the advantages offered by VSBM resulted in it being incorporated into a number of video coding standards. The H.263 and MPEG-4 coding standards were the first to allow variable size blocks (16×16 or 8×8). In addition, H.264/AVC provides several different macro-block partitioning modes, ranging from 16×16 to 4×4 blocks.

6.3 Motion Compensation Error Surfaces

In traditional block matching, the goal is to minimise the error between a block in the current frame and a displaced block in the reference frame. The distortion is usually measured in terms of either the sum of absolute error (SAE) or the sum of squared error (SSE). In effect, motion estimation amounts to finding the location of the minimum value on the error surface. Because they will prove useful later (in Sections 6.4 and 6.5), the process of generating motion compensation error surfaces is discussed below in more detail.

For each block, an error surface $\mathbf{E}_{b,f}(u,v)$ is calculated, i.e. the SSE when block number b is motion compensated by translating the corresponding block in reference frame number f a distance (u,v). For all points (x,y) in block b, the SSE between the current frame, I, and a reference frame, I_f, is calculated according to the equation:

$$\mathbf{E}_{b,f}(u,v) \;=\; \sum_{(x,y)\;\in\;\text{Block } b} [I(x,y) - I_f(x+u, y+v)]^2 \tag{6.1}$$

For a given block, the translational motion vector (measured to pixel accuracy) can be found by determining the vector (u,v) that minimises $\mathbf{E}_{b,f}(u,v)$. If required, sub-pixel motion can be estimated — either by performing matching (using interpolated pixel values at non-integer positions) or by interpolating the location of the minimum value on the error surface.

In general, it is possible to use multiple reference frames for block matching. For each block, it is necessary to calculate one error surface per reference frame. Figure 6.1 shows the position of a block in the current frame, as well as two

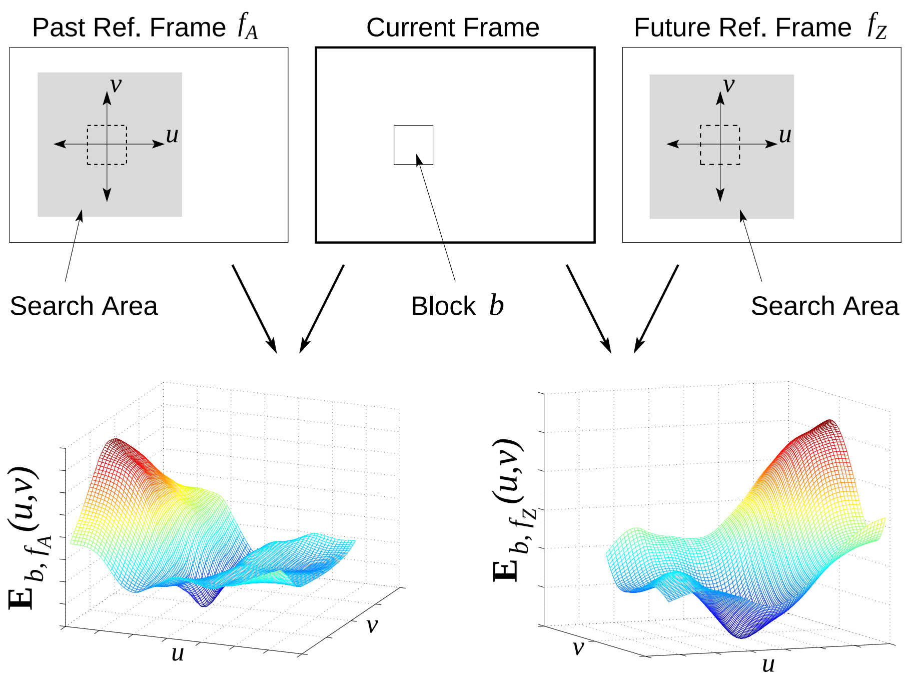


Figure 6.1: Generating motion compensation error surfaces using block matching (relative to the two reference frames)

typical error surfaces obtained when performing matching using two reference frames. In the remainder of the discussion in this chapter, it is assumed (for simplification purposes) that two reference frames are available, however the approach can easily be generalised to allow more reference frames.

6.4 Block Partitioning

When a block contains regions with different motion characteristics, it is difficult to represent the motion accurately using just one motion vector. It is therefore desirable to partition the block into child blocks in such a way that the child blocks contain regions of homogeneous motion. Nevertheless, deciding on how (and where) to split a block can be a very complex process, since searching the space of all possible partitions would be computationally expensive. It was therefore decided to adopt a simple approach to block partitioning, as discussed below.

Binary partitioning is used, which allows for a block to be split into two children. This is not a significant constraint, since these children can themselves be divided into two at a later stage. In order to simplify the partitioning process, straight horizontal or vertical lines are used to divide a block into two. While other forms of splitting (e.g. using diagonal or curved lines) may sometimes be more appropriate (in terms of fitting motion boundaries), the use of straight lines for block partitioning makes the process manageable. (If there are fewer options available for splitting, the range of options can be searched more quickly and also coded more efficiently.) One further simplification is to only allow partitioning of the *larger* dimension of a block. Thus if a block has $width > height$, it is split by a vertical line, otherwise it is partitioned using a horizontal line.

Given the above constraints on permissible block partitions, the goal is to select the partition which allows for the maximum reduction in motion compensation error. The process for determining the optimal partition for a block of height h pixels and width w pixels is outlined below and accompanied by an illustration in Figure 6.2.

- Divide the block into long thin (non-overlapping) strips that are one pixel thick. If $w > h$ then there are w strips of size $1 \times h$ orientated vertically, and if $h \geq w$ then there are h strips of size $w \times 1$ orientated horizontally. Let $N = max(w, h)$ be the number of strips.

- For each strip, s_i, calculate its two error surfaces as shown in Section 6.3: $\mathbf{E}_{s_i,f_A}(u, v)$ and $\mathbf{E}_{s_i,f_Z}(u, v)$, where f_A and f_Z are the two reference frames.

- Consider a partition between strips s_n and s_{n+1} that divides the block into two sets of strips. Then the fist half comprises the set $C_1(n) = \{s_1, ..., s_n\}$, and the second half consists of the set of strips $C_2(n) = \{s_{n+1}, ..., s_N\}$.

- Error surfaces can then be calculated for each of the two halves by simply

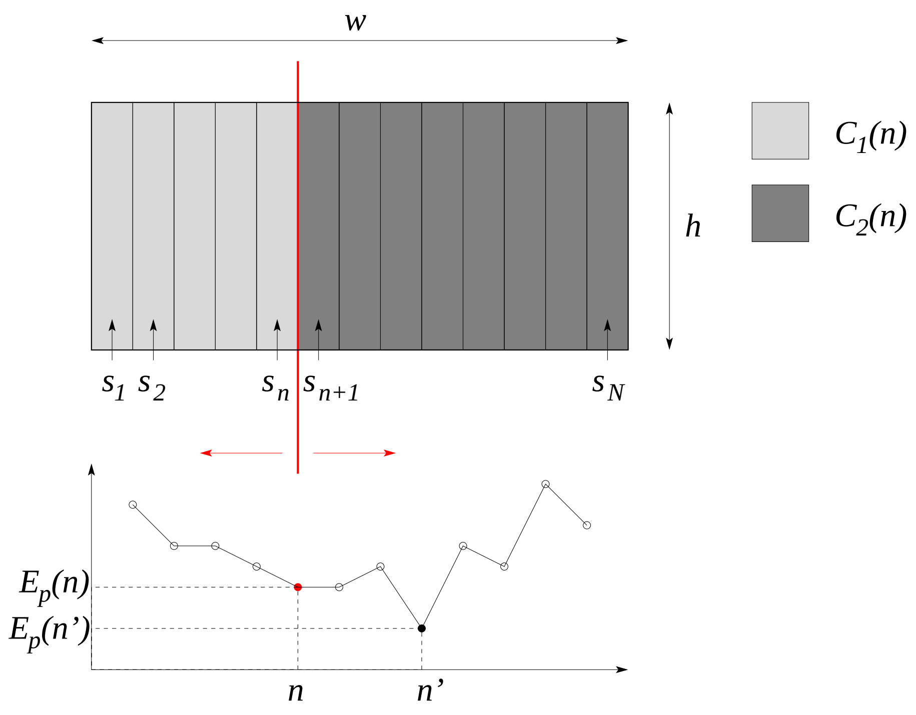


Figure 6.2: Partitioning blocks using error minimisation

adding the error surfaces of their constituent strips:

$$\begin{aligned}\mathbf{E}_{C_1(n),f}(u,v) &= \sum_{s_i=1}^{n} \mathbf{E}_{s_i,f}(u,v) \qquad \text{and} \\ \mathbf{E}_{C_2(n),f}(u,v) &= \sum_{s_i=n+1}^{N} \mathbf{E}_{s_i,f}(u,v) \qquad (6.2)\end{aligned}$$

- Then the total minimum SSE for the block when partitioning it between strips s_n and s_{n+1} is given by:

$$\begin{aligned}E_p(n) &= \min\left\{\min(\mathbf{E}_{C_1(n),f_A}), \min(\mathbf{E}_{C_1(n),f_Z})\right\} + \\ &\quad \min\left\{\min(\mathbf{E}_{C_2(n),f_A}), \min(\mathbf{E}_{C_2(n),f_Z})\right\} \qquad (6.3)\end{aligned}$$

The optimal partition point is given by n', the value of n that minimises $E_p(n)$ (with $1 \leq n < N$). If $E_p(n)$ is constant for all n, then n' is assigned a value of $\lfloor N/2 \rfloor$. In the worst case scenario, $E_p(n')$ will equal the original minimum SSE of the block (in which case the block is simply split down the middle). However, if the motion in the block is non-uniform, $E_p(n')$ will typically have a value less than the original minimum SSE. This means that splitting the block into two child blocks $C_1(n')$ and $C_2(n')$ will generally yield a reduction in motion compensation error.

6.5 Generating a Binary Partition Tree

The previous section described the process for splitting individual blocks based on minimising motion compensation error. This can be applied iteratively to an entire image in order to generate a block structure that allows for effective motion compensation using relatively few blocks.

Block partitions are represented using a *binary partition tree*, that indicates which blocks are split and the location of these partitions within each block. The strategy used is a split and merge one, where more blocks are split than required, followed by subsequent re-merging of certain blocks. The process for growing, pruning and coding the binary partition tree is described below in more detail.

6.5.1 Growing the Tree

To generate a binary partition tree, start with one block that is the size of the frame. This block is the root of the binary tree, and initially it is also its only entry. The process of growing the tree by repeatedly partitioning blocks then operates as follows:

- Find the block in the tree which has the largest minimum SSE - i.e. the block with the maximum value of

  $$E_{min}(b_i) = min\{min(\mathbf{E}_{b_i,f_A}), min(\mathbf{E}_{b_i,f_Z})\} \tag{6.4}$$

  where b_i is the block index.

- Partition this block using the method described in Section 6.4 to create two new blocks. These two blocks are added to the tree as children of the partitioned block.

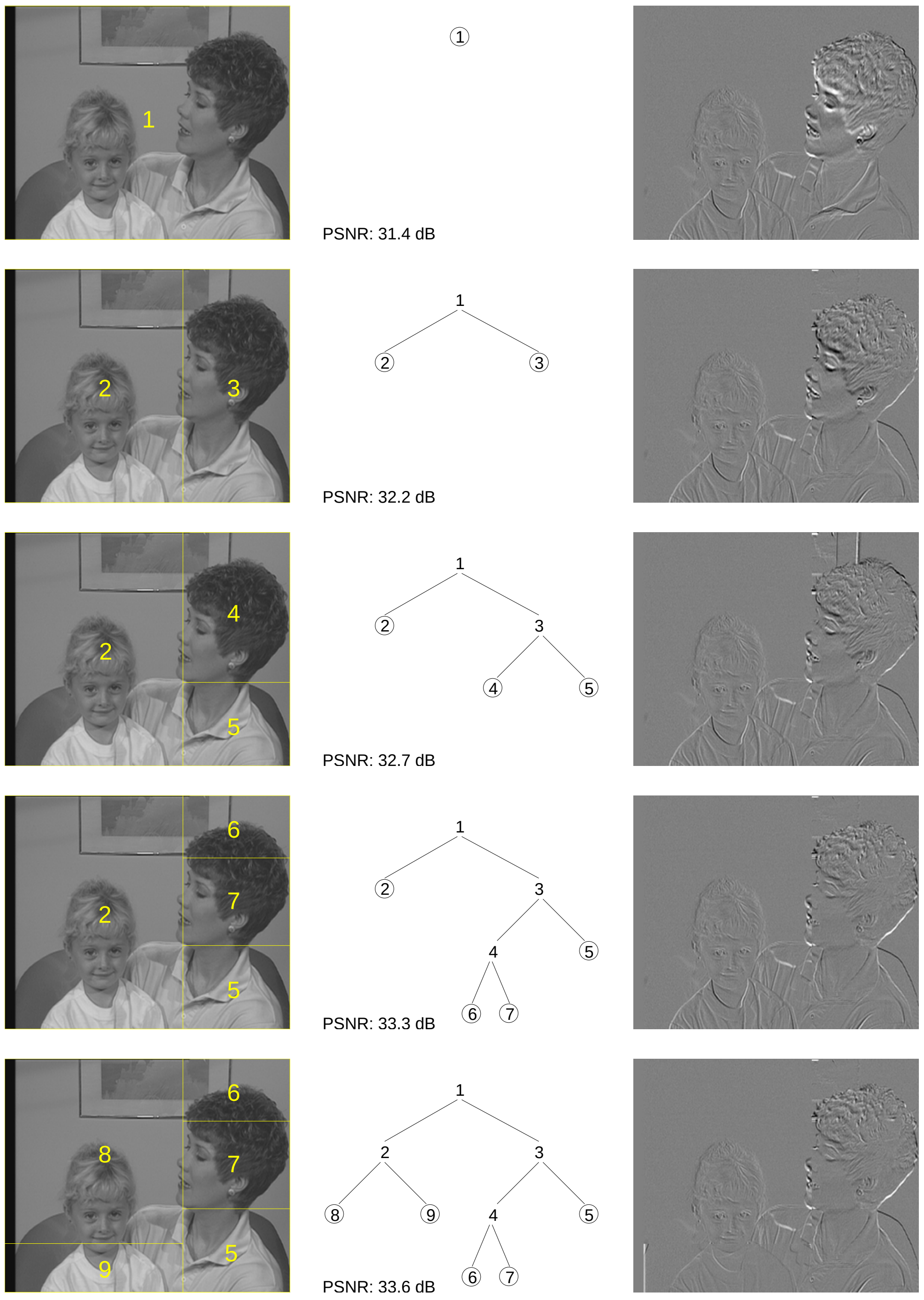


Figure 6.3: An example of the growth of a Binary Partition Tree for frame 50 of *Mother & Daughter*. The blocks are overlaid on the original image at each stage. The corresponding binary tree structure and the DFD are also shown.

- The previous steps are repeated while the number of *leaf* nodes in the tree (i.e. blocks without children) is less than $1.25N_B$, where N_B is the target number of blocks.

Figure 6.3 provides an example of the block partitioning and tree growing procedure for frame 50 of *Mother & Daughter* (relative to reference frames 48 and 52). Starting with one block spanning the entire frame, each row shows one stage in the partitioning process. The example illustrates the first four binary partitioning steps, resulting in the creation of a tree with five leaf nodes (blocks).

The first column in Figure 6.3 illustrates the leaf blocks in the tree overlaid on the original image, with each block assigned a unique label. The growth of the binary tree is represented in the second column, with leaf blocks depicted as circled nodes. The third column shows the Displaced Frame Difference (DFD) after motion compensation using the relevant block structure. At each stage, the corresponding (pixel-accurate) luma PSNR is also given. As can be seen, the partitions generally correspond to motion boundaries. For example, the first partition provides a rough separation of the mother's head (which is moving) from the stationary picture background on the left. Likewise, the second partition divides the mother's head from her neck and upper body (which move much less than the head).

When using just one motion vector to motion compensate the entire frame, a PSNR of 31.4 dB is obtained. Splitting the frame into two blocks (each with their own motion vector) achieves a gain of 0.8 dB, while subsequent partitioning leads to slightly smaller (but still significant) improvements in quality.

6.5.2 Pruning the Tree

Recall that when growing the binary tree, the (leaf) block chosen for partitioning at each stage is the one with the largest minimum SSE (i.e. the maximum value of $E_{min}(b_i)$). However, this top-down approach does not necessarily yield the best choice of block for further splitting. It may be that splitting some other block in the tree yields a greater reduction in error.

As a result, it was decided to incorporate a bottom-up block merging process when generating the binary tree. This requires that more than the target number (N_B) of blocks be generated. As described above, $1.25N_B$ blocks are created. This enables a more optimal bottom-up approach to be used for pruning the excess blocks.

The advantage of bottom-up pruning is that one can consider all pairs of *leaf* child blocks in the tree, and re-merge the pair whose splitting provided the smallest error reduction. The process is outlined below in more detail.

- Consider a pair of child blocks (b_{c_1} and b_{c_2}), both of which are leaf nodes and share a common parent, b_p. The increase in error that results from pruning these two siblings (i.e. performing motion compensation using the parent block instead of the two child blocks) is given by:

$$E_{Prune}(b_{c_1}, b_{c_2}) = E_{min}(b_p) - E_{min}(b_{c_1}) - E_{min}(b_{c_2}).$$

- Calculate E_{Prune} for each pair of children that are also both leaf nodes. Merge the pair which yields the smallest value of E_{Prune}. The two leaf nodes are thus pruned from the partition tree, and their parent becomes a leaf node.

- The previous step is repeated until the number of *leaf* nodes in the tree equals N_B. The end result is a frame consisting of these N_B variable-sized blocks.

An example of binary tree pruning is illustrated in Figure 6.4 for frame 50 of *Mother & Daughter* (motion compensated relative to reference frames 48 and 52). The first and second rows show the use of a binary partition tree grown to 40 and 50 blocks (leaf nodes) respectively. The use of these ten additional blocks results in a gain of 0.7 dB in luma PSNR. When the 50-block tree is pruned to 40 nodes, it yields the binary partition tree depicted in the bottom row of the figure. The advantage of using pruning is clearly visible: The *pruned* 40-block tree results in an increase in quality of 0.5 dB over the *un-pruned* 40-block tree.

The pruning method described above increases the number of blocks by 25% during the tree growing stage. This was found empirically to provide a sufficiently large pool of blocks from which to prune those partitions that provide little advantage. Increasing this percentage may lead to slightly improved performance, but would also increase the computation involved in both growing and pruning the tree.

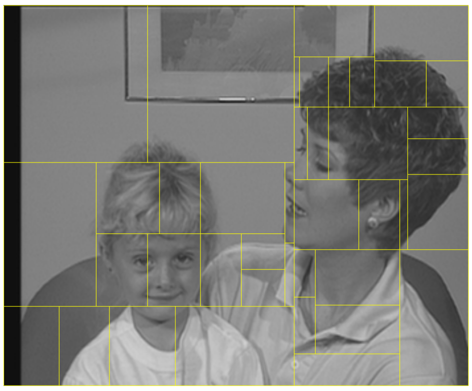
6.5.3 Coding the Tree

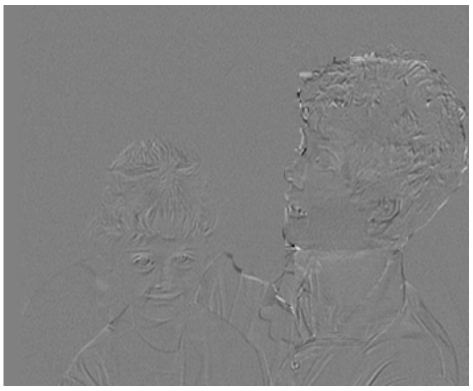
Coding the binary partition tree is fairly straightforward, and comprises two components. The first is the shape of the tree, where each node has two possibilities: either it is a leaf (in which case it has no children), or it is not a leaf (in which case it has exactly two children). For a tree with N_B leaf blocks, $2N_B - 1$ bits are required to code this information. The second component that is required relates to *where* blocks with children are to be partitioned. (Coding a block partition requires specifying the position of the horizontal or vertical line that divides the block in two.)

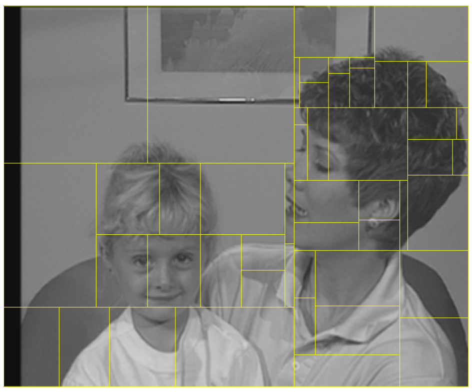
The algorithms for the encoding and decoding of a binary partition tree are presented below, using pseudo-code. Note that the *leaf* blocks are the ones that are ultimately used for motion compensation.

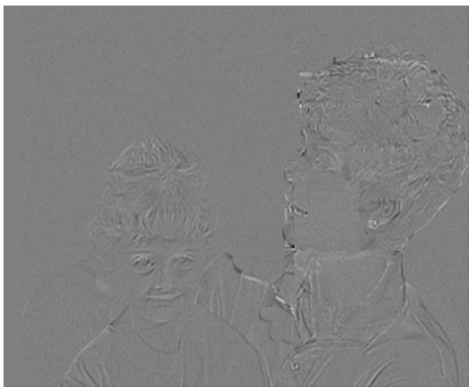
Algorithm for Encoding a Binary Partition Tree:

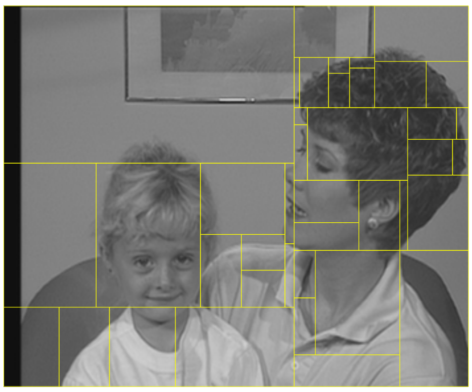
```
CurrentNode = RootNode; // i.e. entire frame
NodeList = {}; // empty list to start with
while CurrentNode $\neq$ {} do
  if CurrentNode.HasChildren() then
    Encode(1);
    Encode(CurrentNode.SplitPosition); // where current block is split
```


(a) 40 blocks


(b) DFD for (a); Luma PSNR: 37.5 dB


(c) 50 blocks


(d) DFD for (c); Luma PSNR: 38.2 dB


(e) 40 blocks (pruned from 50)

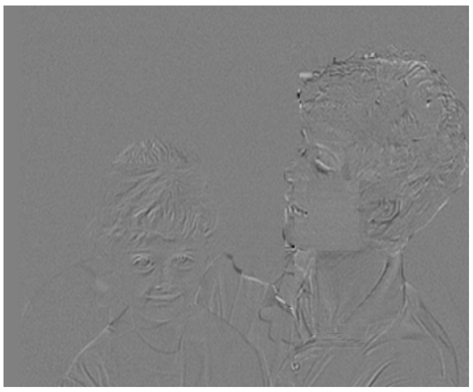

(f) DFD for (e); Luma PSNR: 38.0 dB

Figure 6.4: An example of pruning a Binary Partition Tree for frame 50 of *Mother & Daughter*. The column on the left shows the blocks overlaid on the original image. The column on the right illustrates the corresponding DFD after motion compensation (from frames 48 and 52).

```
      NodeList.Append(CurrentNode.ChildNode1); // add child block 1 to list
      NodeList.Append(CurrentNode.ChildNode2); // add child block 2 to list
    **else**
      Encode(0);
    **end if**
    CurrentNode = NodeList.ExtractFirstNode(); // extract first block
  **end while**
```

Algorithm for Decoding a Binary Partition Tree:

```
  CurrentNode = RootNode; // i.e. entire frame
  NodeList = {}; // empty list to start with
  LeafNodeList = {}; // empty list to start with
  **while** CurrentNode $\neq$ {} **do**
    HasChildren = DecodeBit();
    **if** (HasChildren = 1) **then**
      CurrentNode.SplitPosition = DecodePartition(); // where block is split
      CurrentNode.CreateChildren(); // create child blocks (based on split)
      NodeList.Append(CurrentNode.ChildNode1); // add child block 1 to list
      NodeList.Append(CurrentNode.ChildNode2); // add child block 2 to list
    **else**
      LeafNodeList.Append(CurrentNode); // add block to list of leaf blocks
    **end if**
    CurrentNode = NodeList.ExtractFirstNode(); // extract first block
  **end while**
```

The above algorithms provide a good general description of the binary tree coding process, however they do not describe the details of coding each block's partition location. (Recall that blocks are partitioned using a straight line that splits their longer dimension.) This information is encoded using arithmetic coding, with a probability model that assumes that splitting a block near the middle is more likely than towards the edges.

For a block with a longer dimension of length l, a (modified) Laplacian distribution is used to model the split position. Let $p(n)$ represent the probability of a partition between positions n and $n+1$. Thus valid values for n are integers in the range $1 \leq n < l$. The Laplacian distribution is centred at the midpoint of the line (i.e. at $n = \lfloor l/2 \rfloor$) and is given by the equation:

$$p(n) = \frac{\alpha}{q} \left(\left\lceil q \left[\log_2 (l-1) \right]^{\left(\frac{|n - l/2|}{1 - l/2} \right)} - 1/2 \right\rceil + \beta(n) \right) \tag{6.5}$$

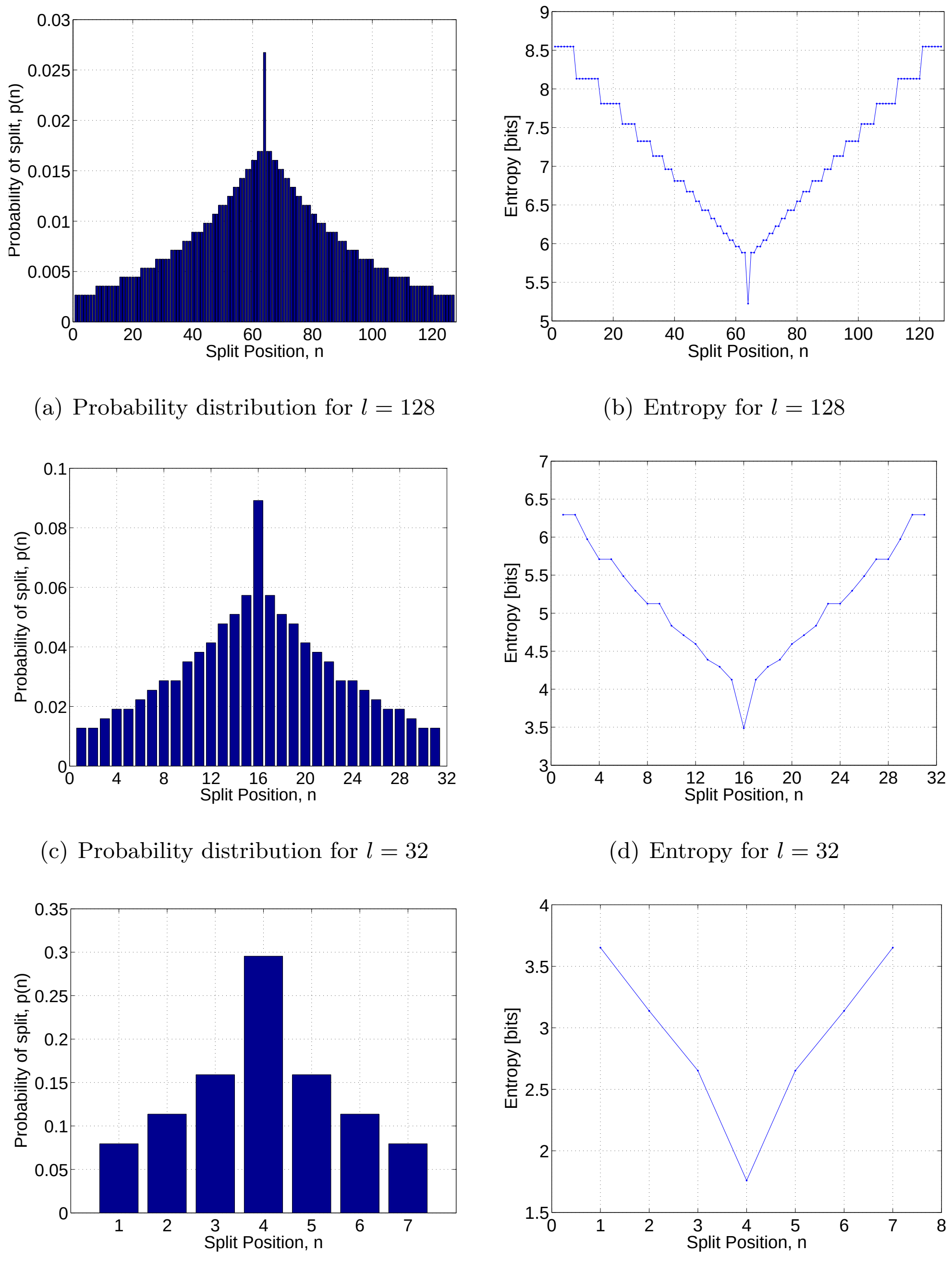
$$\text{where } \beta(n) = \begin{cases} 0 & n \neq \lfloor l/2 \rfloor \\ \log_2 \left(\frac{1}{2} q l \right) & n = \lfloor l/2 \rfloor \end{cases}$$

$$\text{and } \alpha \text{ has a value such that } \sum_{i=1}^{l-1} P(i) = 1$$

The parameter, q, is used to quantise the probability distribution, thus enabling it to be represented as a histogram.[2] The effect of the term $\beta(n)$ is to increase the probability of partitioning a block down its centre. (Recall that a block is split down its centre when none of the possible partitions results in a reduction in motion compensation error.)

Figure 6.5 provides examples of probability distributions for splitting blocks of three different sizes. The associated entropy values are also shown. These indicate the number of bits required to encode the position of a split. As can clearly be seen, the most probable position (and the least costly to code) is at the centre, while splitting becomes less likely (and more expensive to code) towards the edges each the block.

[2] The arithmetic coder used for entropy coding requires a histogram of integer values, rather than a (real-valued) probability distribution. The use of the quantisation parameter, q, enables a one-to-one relationship between the Laplacian distribution in Equation 6.5 and its corresponding histogram. Note that q is limited to a *minimum* value of $\frac{1}{2}\log_2(l-1)$. A fixed value of $q = 20$ was used, since it introduces limited quantisation noise in the distribution.



(a) Probability distribution for $l = 128$

(b) Entropy for $l = 128$

(c) Probability distribution for $l = 32$

(d) Entropy for $l = 32$

(e) Probability distribution for $l = 8$

(f) Entropy for $l = 8$

Figure 6.5: An example of some typical (modified) Laplacian probability distributions used for modelling where a block of length l is likely to be partitioned. The associated entropy is also shown. (The entropy graphs indicate the number of bits required to specify the location of a partition.)

6.6 Motion Estimation

Motion estimation plays a critical part in the block partitioning stage, since it is used when calculating the error surfaces for each block. Because each block's error surfaces are known, its motion vector (and choice of reference frame) can easily be determined by finding the location of the minimum error for that block. This provides the block's motion vector to pixel accuracy. In order to refine the accuracy to a sub-pixel level, a search is performed in the neighbourhood of the (pixel-accurate) motion vector. Bi-linear interpolation is used to estimate pixel values at sub-pixel locations.

Motion vectors and reference frame selection information are then coded for each block as follows: First, (leaf) blocks are sorted in raster scan order according to the position of their top left corners. This allows for a fixed ordering of blocks, which is reproducible when decoding. For each block, the motion vector is entropy coded - relative to the motion vectors of those neighbouring blocks already coded. Similarly, the reference frame used for each block is also predicted from those of neighbouring blocks. Note that arithmetic coding is used for entropy coding both the motion and reference frame information.[3]

6.7 Results

The performance of the proposed variable-size block matching approach was tested on nine[4] test sequences, using both regular and content-based blocks. This section presents results for the *Foreman* (CIF) and *Stefan* (CIF) sequences, both of which contain significant motion. (Note that the results shown here are representative of those obtained for the full set of nine sequences.)

[3] More details on the coding of motion information are provided in Section 7.4.3

[4] Namely *City*, *Flower Garden*, *Football*, *Foreman*, *Mobile & Calendar*, *Stefan*, *Crowd*, *Edinburgh* and *Tennis*. The first six are standard test sequences, while the latter three are proprietary BBC sequences.

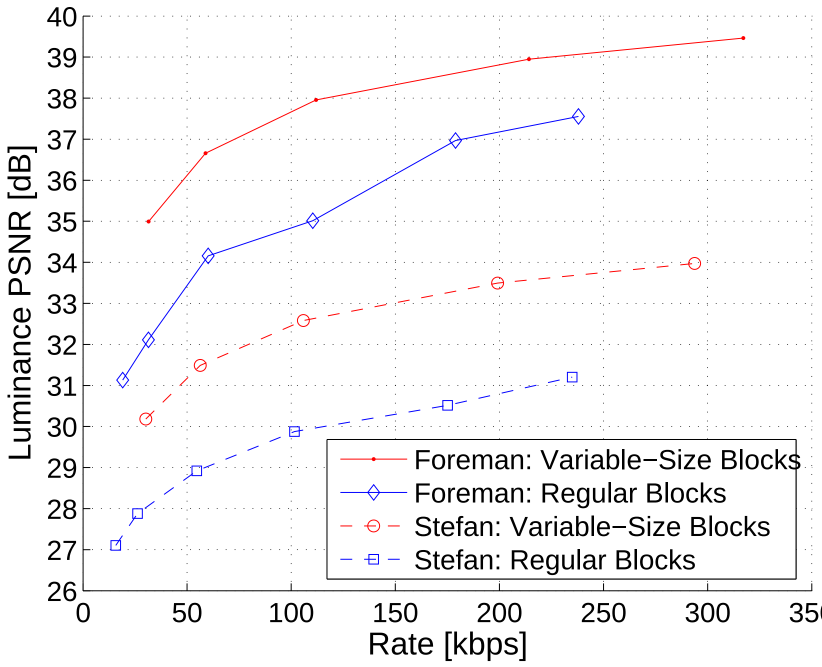


Figure 6.6: Rate-Distortion plots for *Foreman* and *Stefan*. Results have been averaged over frames 10 to 40 for both sequences. The points plotted show the results when setting the target number of blocks to: {50, 99, 198, 396, 594, 792} (regular blocks) and {50, 99, 198, 396, 594} (variable-size).

The performance of the proposed variable-size block matching approach was tested on frames 10 to 40 of both *Foreman* and *Stefan*. Given a target number of blocks, each frame (I'_t) was motion compensated from two *original* frames (I_{t-2} and I_{t+2}), a distance of two frames away in time (before and after). For each frame, the tree structure was encoded, along with the reference frame selection information and the motion vectors (at quarter-pixel accuracy).

For comparative purposes, the same tests were performed using square, fixed-size blocks. In this case, only the motion information was encoded, since the block structure is regular for a given number of blocks. In order to test at a variety of different rates, the number of blocks per frame, N_B, was varied. When using both regular and variable-size blocks, N_B was chosen from the values: 50, 99, 198, 396 and 594. In addition, 792 regular-size blocks were used.

The rate-distortion performance of the two methods is illustrated for both sequences in figure Figure 6.6. It can be seen that the proposed method provides substantial gains in quality of between 1.5 and 3.0 dB over regular block matching at the same bit-rate.[5] This is despite the overhead required to represent the structure of the partition tree.

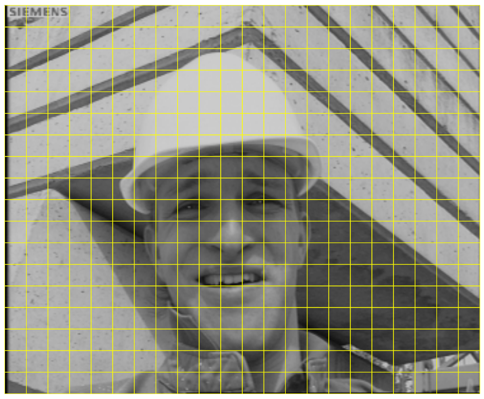
Figure 6.7 shows frame 32 of *Foreman*, motion compensated from *original* frames 30 and 34 in the sequence. When using 396 regular blocks, there are some blocking artifacts present, particularly at the corner of the man's mouth and around his chin (see Figures 6.7 (c) and (e)). This type of distortion occurs when blocks straddle motion boundaries, and are thus unable to compensate for multiple motion.

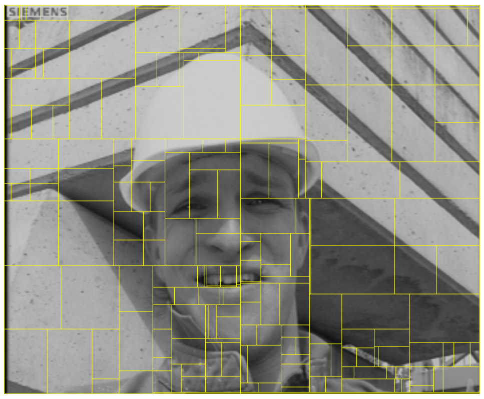
The use of a binary partition tree allows blocks to vary in size and shape, based on scene content. Using 198 variable-size blocks requires roughly the same bit-rate as twice as many regular blocks, although blocking artifacts are much less evident. For the content-based method, there is some distortion visible around the man's chin (see Figures 6.7 (d) and (f)). However, this can in part be ascribed to the use of only the luma component when performing block matching.[6]

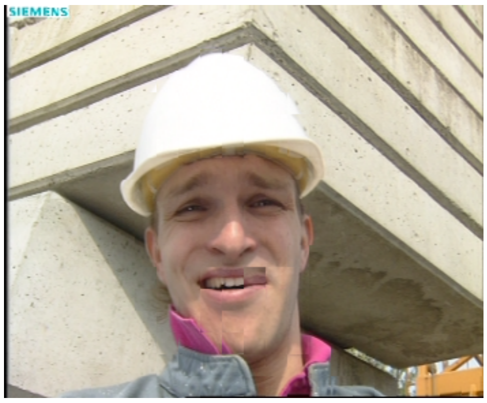
Figure 6.8 shows frame 13 of *Stefan*, motion compensated from *original* frames 11 and 15 in the sequence. When using 396 regular blocks, there are significant blocking artifacts present, particularly along the top and right edges of the frame (see Figures 6.8 (c) and (e)). This is due to the blocks along the perimeter struggling to represent two types of motion. In all images in the sequence, there are thin black strips along the top and right borders. These are effectively regions of zero motion. However, the picture background is subject to a zooming and panning camera motion (as well as movement of people in the crowd). Consequently, blocks covering one of the black strips *and* the picture background

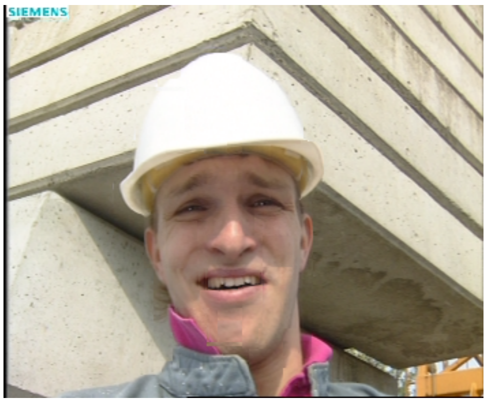
[5] The bit-rate is given in kilobits per second (kbps) for CIF-resolution frames at 30 fps.

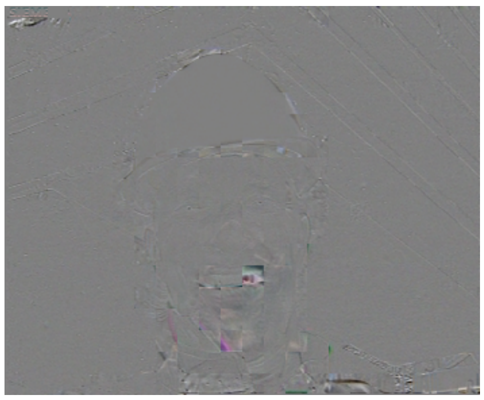
[6] Matching using the luma component only can sometimes result in an incorrect motion vector. This is because the the luma components of two blocks can be similar, while their chroma components may differ substantially.


(a) 396 regular 16 × 16 blocks


(b) 198 variable-size blocks


(c) After motion compensation using the blocks in (a). Total rate: 103 kbps


(d) After motion compensation using the blocks in (b). Total rate: 109 kbps


(e) DFD for (c); Luma PSNR: 34.6 dB

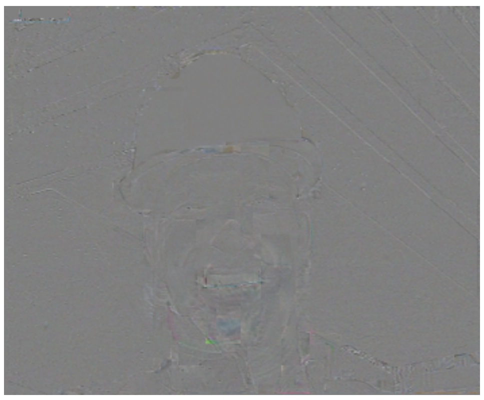
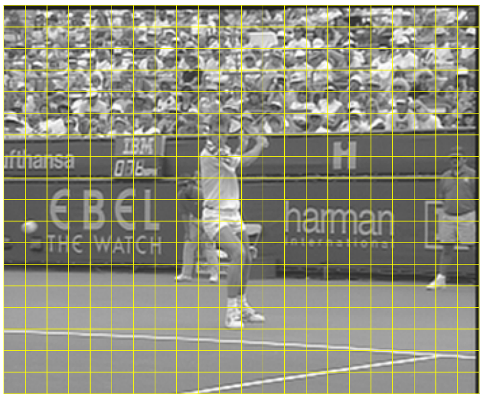
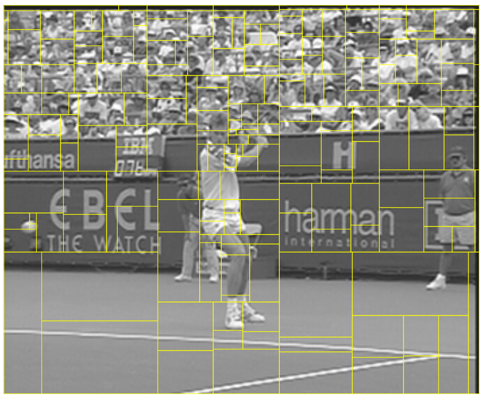

(f) DFD for (d); Luma PSNR: 38.5 dB

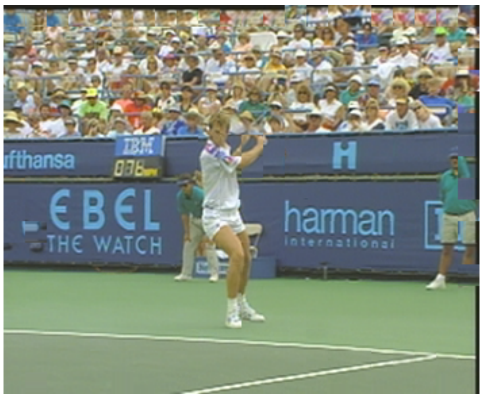
Figure 6.7: Frame 32 of *Foreman* motion compensated from original frames 30 and 34. ($\frac{1}{4}$-pel motion vector accuracy.)



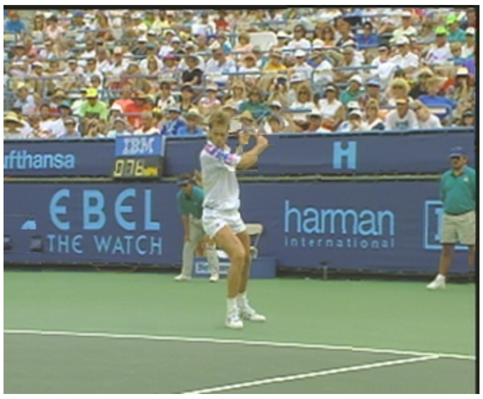
(a) 396 regular 16 × 16 blocks



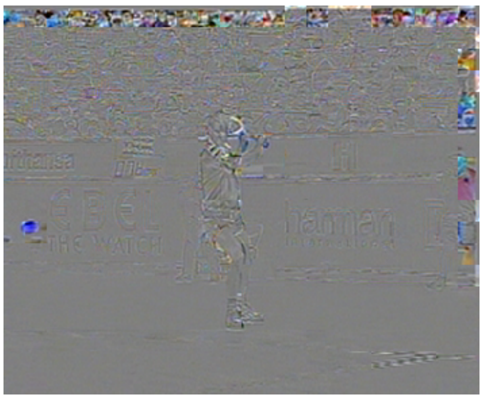
(b) 198 variable-size blocks



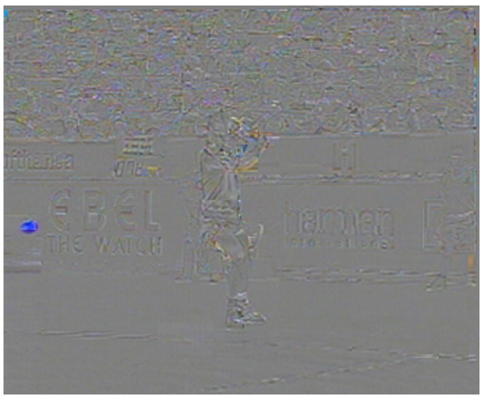
(c) After motion compensation using the blocks in (a). Total rate: 154 kbps



(d) After motion compensation using the blocks in (b). Total rate: 119 kbps



(e) DFD for (c); Luma PSNR: 27.33 dB



(f) DFD for (d); Luma PSNR: 31.3 dB

Figure 6.8: Frame 13 of *Stefan* motion compensated from original frames 11 and 15. ($\frac{1}{4}$-pel motion vector accuracy.)

are unable to represent both the moving background and the stationary black border.

The proposed binary partition tree is well suited to handling this problem, since it partitions the black perimeter area from the picture background (see Figure 6.8 (b)). As a result the error in the DFD is significantly reduced, although there is still some distortion present around the player's upper body, and particularly the head of his racquet (see Figures 6.8 (d) and (f)). Note that both regular and variable-size blocks fail to track the motion of the tennis ball on the left of the picture. This is probably because of the ball's fast motion, which results in it lying outside the motion vector search areas in the two reference frames.

6.8 Conclusion

The binary partitioning method proposed in this chapter provides an effective compromise between region-based and block-based methods of motion compensation. It does this by incorporating the segmentation process into the block-generation stage, although the approach is made manageable by allowing only straight-line rectangular partitioning.

To start with, a top-down block splitting technique is used in order to select *which* blocks should be split. This is followed by a bottom-up merging process that helps to remove those partitions that provide the least advantage. The process of selecting *where* a block is split is based on minimising the motion compensation error (subject to the constraints of straight-line partitioning). As was demonstrated, this method results in partitioning along motion boundaries, as well as the creation of small blocks in regions of complex motion, with larger blocks spanning background areas.

There is a significant overhead associated with coding a binary partition tree. However, it was shown that the technique requires relatively few blocks (compared to regular block matching) in order to provide reasonably good quality motion compensation. Consequently, fewer bits are required to encode the motion vector information.

The next chapter goes on to discuss how the proposed binary partition tree approach to block matching can be incorporated into a video codec. It also provides a comparison with the H.264/AVC implementation of variable-size block matching.

Chapter 7

Integration into a Hybrid Video Codec

7.1 Introduction

Most video coding methods rely heavily on motion compensation to help reduce the temporal redundancy between frames. The three preceding chapters discussed content-based methods of motion estimation and compensation. As was shown in the Chapter 6, a Binary Partition Tree approach to block-based motion compensation allows for significant advantages over regular block matching.

This chapter describes the design of a hybrid video codec that allows the user to select one of four block-based motion compensation methods. Two of these methods are fixed-size block matching (using either 8×8 or 16×16 blocks). The third choice available is the H.264/AVC implementation of variable-size block matching (VSBM). The fourth option allows for the use of Binary Partition Tree VSBM, based on the technique outlined in the previous chapter.

Each of the components of the video codec is described in some detail. The primary purpose of the codec is not to try and out-perform all other coding methods (although this is certainly desirable). Rather, the focus is on providing a common framework with which to compare the four block-based motion estimation and compensation techniques highlighted above.

With some effort, it would be possible to integrate the proposed Binary Partition Tree VSBM method into a codec such as H.264/AVC. However, H.264/AVC has already been optimised for its own VSBM approach, and would thus bias rate-distortion performance in favour of this method of motion compensation.[1] Consequently, it was decided to develop a "neutral" video codec, combining many state-of-the-art components.

7.2 An Overview of the Codec Structure

The design of a video codec can be fairly complex. This section provides an overview of several building blocks used in the proposed system. Having established the foundations, the next two sections focus in more detail on both inter and intra frame coding. Block diagrams showing the structure of the encoder and decoder are provided in Figures 7.1 and 7.2. They show the relationship between the various codec components that are discussed in more detail throughout the remainder of this chapter.

7.2.1 Picture Types and Grouping

In common with standards such as MPEG-1 and MPEG-2, the proposed video codec uses three frame types. Intra (I) frames are coded directly using transform coding techniques, without reference to any other frames. Predicted (P) frames are motion compensated from the preceding reference frame. (Both I and P frames can act as reference frames.) Bi-directionally predicted (B) frames are motion compensated from two frames — the immediately preceding and subsequent reference frames. This is shown graphically in Figure 7.3(a), where the arrows indicate the relationships between the three frame types.

[1] For example, the de-blocking filters used in H.264/AVC are designed for a specific range of block sizes and shapes. As a result, they are unlikely to perform as well on blocks of arbitrary size and shape.

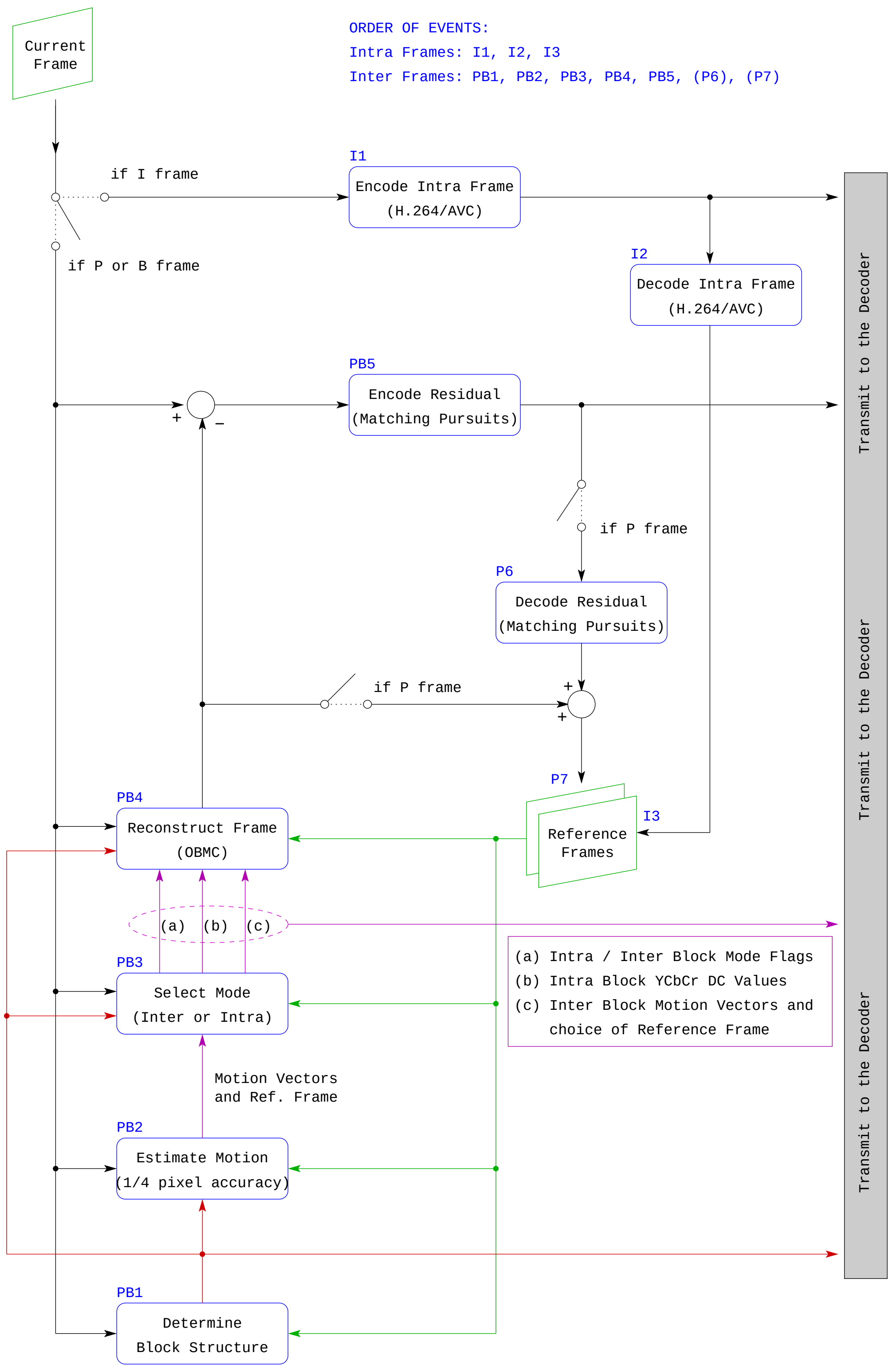

Current Frame
ORDER OF EVENTS:
Intra Frames: I1, I2, I3
Inter Frames: PB1, PB2, PB3, PB4, PB5, (P6), (P7)
if I frame
if P or B frame
I1
Encode Intra Frame (H.264/AVC)
I2
Decode Intra Frame (H.264/AVC)
PB5
Encode Residual (Matching Pursuits)
+
−
if P frame
P6
Decode Residual (Matching Pursuits)
if P frame
+
+
P7
I3
Reference Frames
PB4
Reconstruct Frame (OBMC)
(a) (b) (c)
PB3
Select Mode (Inter or Intra)
(a) Intra / Inter Block Mode Flags
(b) Intra Block YCbCr DC Values
(c) Inter Block Motion Vectors and choice of Reference Frame
Motion Vectors and Ref. Frame
PB2
Estimate Motion (1/4 pixel accuracy)
PB1
Determine Block Structure
Transmit to the Decoder
Transmit to the Decoder
Transmit to the Decoder

Figure 7.1: Block diagram of the Encoder

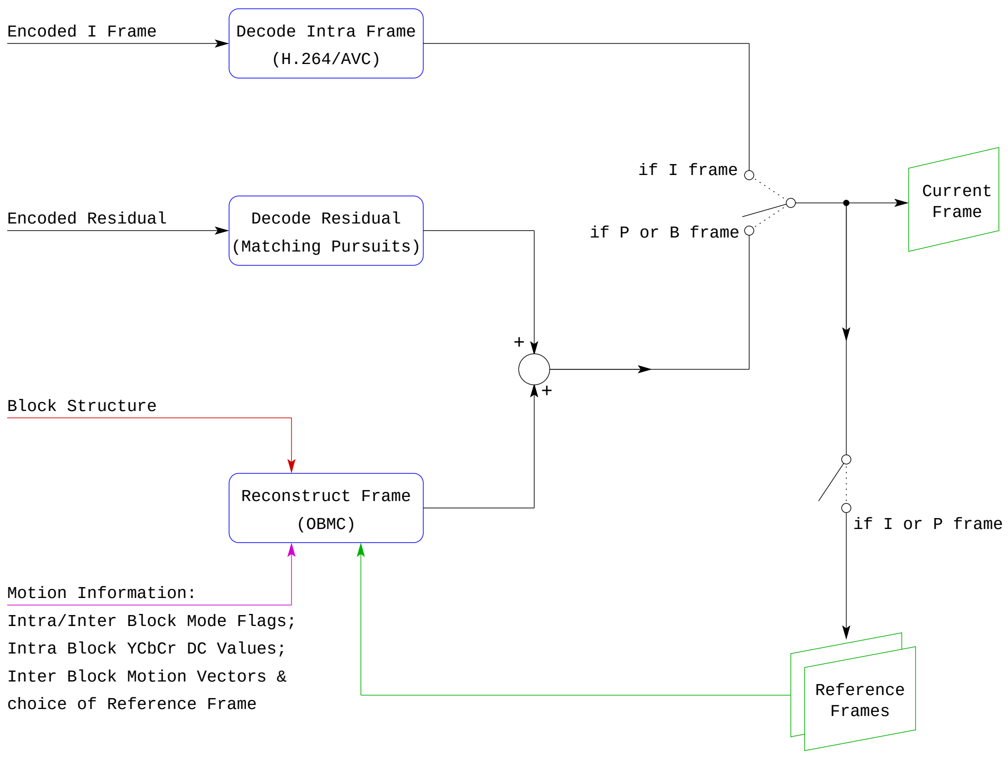
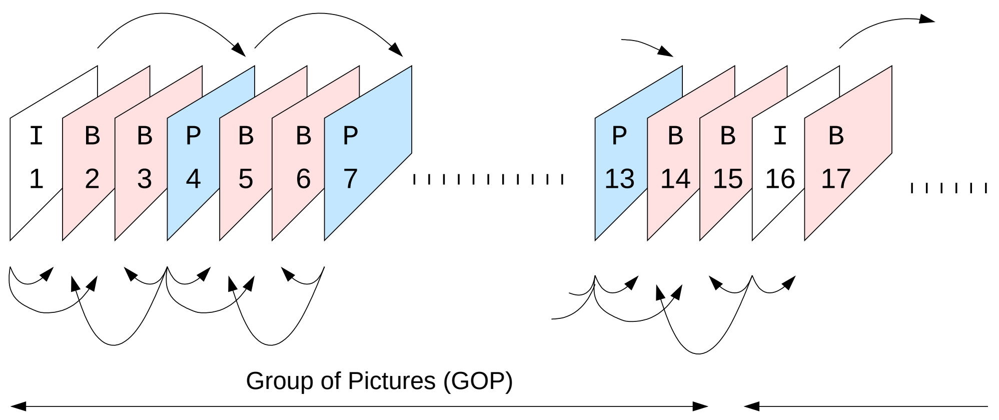

Encoded I Frame
Decode Intra Frame (H.264/AVC)
Encoded Residual
Decode Residual (Matching Pursuits)
Block Structure
Reconstruct Frame (OBMC)
Motion Information:
Intra/Inter Block Mode Flags;
Intra Block YCbCr DC Values;
Inter Block Motion Vectors &
choice of Reference Frame
if I frame
if P or B frame
Current Frame
if I or P frame
Reference Frames
+
+

Figure 7.2: Block diagram of the Decoder

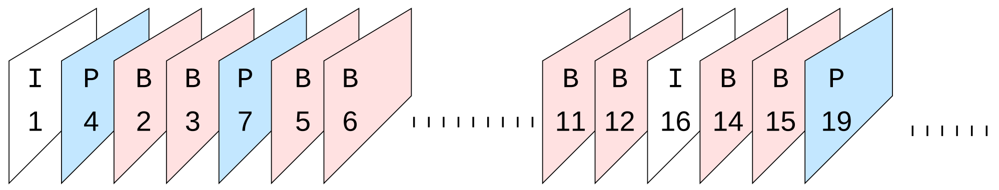
Figure 7.3(a) also shows how an image sequence is arranged into sets of 15 frames, known as a Group of Pictures (GOP). As mentioned above, B frames are predicted from future (and past) reference frames. Using a future reference frame is made possible by coding the frames in a different order to that in which they are displayed. This coding order (shown in Figure 7.3(b)) introduces a small delay into the system and also requires a limited amount of frame buffering, however the use of both past and future reference frames (for B pictures) allows for more effective motion compensation. At the decoder, frames are simply rearranged into their original display order.

7.2.2 Codec Input Parameters and File Formats

The encoder accepts input files in YC_bC_r 4:2:0 input format, with each sample represented using 8-bit precision. There is one file per frame, with the filename indicating the index of the frame within the sequence. The standard sequences



(a) Display order (with arrows indicating the direction of frame prediction).



(b) Coding order

Figure 7.3: The frame structure of a Group of Pictures (GOP)

used in testing were obtained in this raw, one-file-per-frame format. The decoder produces an output video with the same specifications, except that frames are concatenated together into one file. The use of a single file allows for the sequence to be played back using an off-the-shelf YC_bC_r 4:2:0 video player.

Each frame comprises one luma and two (half-resolution) chroma components. The number of frames (N_F) for encoding and decoding is required to be such that $N_F = 3n + 1$, for n a non-negative integer. This ensures that the sequence always ends with either an I or a P frame. Terminating on a B frame is not possible, since such a frame would have no future reference frame.

The encoder requires the following input parameters from the user:

- The name of the YC_bC_r video file to be compressed;
- The number of rows and columns of the luma component (Both of these are required to be integer multiples of 16 and less than 4096);

- The indices of the frames in the sequence at which to start and stop coding (Recall that the number of frames to be coded should be one plus an integer multiple of three.);
- The search radius used during motion estimation (This should have a value less than 256.);
- The choice of block partitioning method (Four options are available, namely: regular 16×16 blocks, regular 8×8 blocks, H.264/AVC variable size blocks, and the Binary Partition Tree block structure.); and
- A quantisation parameter, QP, used in the coding of I frames, and indirectly affecting all other frames. (This allows for control over the quality and bit-rate of the compressed sequence.)

The encoder produces a compressed output file, which is used as the input to the decoder. This coded file consists of arithmetic-coded data, preceded by a seven-byte header. The header stores the video format specifications (such as resolution) and other options required by the decoder (such as the block partition choice and the value of QP).

The decoder only requires one user-specified input, and that is the name of the compressed file to be decoded. All other necessary information can be obtained from the seven-byte header.

7.2.3 Rate and Quality Control

The components of the codec use Lagrangian optimisation in order to ensure that all parts of the system operate at the same trade-off gradient between rate and distortion. This is done by supplying a value of λ to each of the building blocks of the system.

As a result of the motion compensation process there are inter-dependencies between I, P and B frames. For example, motion compensated prediction is applied to an I-frame in order to generate the next P-frame. Thus, the quality of an I-frame determines (in part) the quality of the next P-frame. Because of these inter-dependencies, it was decided to use different values of λ for each frame type (i.e. λ_P and λ_B). These λ values are controlled using a simple picture quality heuristic that is described below.

When coding intra (I) frames, the proposed codec uses the user-specified quantisation parameter (QP) in order to code the frame with H.264/AVC's intra-coding method. This allows some control over the quality of the coded image. It was decided to use this QP value as a means of controlling the quality of *all* frames in the sequence. This method of controlling the encoder based on quality (as opposed to bit-rate) operates according to the following two assumptions:

- The target luma PSNR of a P-frame is 0.7 dB less that the PSNR of its preceding I-frame; and
- The target luma PSNR of a B-frame is 0.3 dB less than the PSNR of its closest P-frame.

These assumptions are based on general observations of picture quality for a variety of traditional hybrid codecs. They were suggested as a simple way of regulating quality (and thus bit-rate) in a reasonable manner [119]. For P and B frames, λ is varied in such a way as to try and maintain the PSNR differences mentioned above.

A. Initialising Lambda

When encoding each sequence, λ_P and λ_B need to be initialised. For λ_P, this is done by using the first and fourth frames in the sequence as I and P frames respectively. Starting from a value of $\lambda_P = 100$, the P frame is encoded, and the resulting quality checked. Depending on how much this differs from the target PSNR, λ_P is updated accordingly.[2] The process is repeated a maximum

[2] Note that at each stage, λ_P and λ_B are scaled by at most $\pm 35\%$. This prevents very rapid changes in λ, and was found to make the process more robust.

of ten times, or until a λ_P that yields an the target PSNR is found.[3] The steps are outlined below using pseudo-code:

Algorithm for Initialising $\boldsymbol{\lambda_P}$:

```
$I'$ = EncodeIntraFrame($I$, QP); // encode I frame
$\lambda_P = 100$; // initial estimate
counter = 0;
**repeat**
  $P'$ = EncodeInterFrame($P, I', \lambda_P$); // encode P frame
  $\Delta$PSNR = PSNR($P, P'$) – (PSNR($I, I'$) – 0.70); // difference in quality
  $\lambda_P = max\{min\{1 + \Delta\text{PSNR}|\Delta\text{PSNR}|, 1.35\}, 0.65\}\lambda_P$; // update $\lambda_P$
  counter = counter + 1;
**until** ($|\Delta\text{PSNR} - 0.70dB| < 0.25$) **or** (counter > 10)
```

In order to initialise λ_B, a similar approach is adopted. The third frame in the sequence is used as a B frame — predicted from both the I (first) and the P (fourth) frames. Using a target PSNR of 0.3 dB less than that of the P-frame (with a tolerance window of ±0.1 dB), the process seeks to find an appropriate value for λ_B, as outlined below:

Algorithm for Initialising $\boldsymbol{\lambda_B}$:

```
$\lambda_B = 2\lambda_P$; // initial estimate
counter = 0;
**repeat**
  $B'$ = EncodeInterFrame($B$, $I'$, $P'$, $\lambda_B$); // encode B frame
  $\Delta$PSNR = PSNR($B$, $B'$) - (PSNR($P$, $P'$) - 0.30); // difference in quality
  $\lambda_B = max\{min\{1 + \Delta\text{PSNR}|\Delta\text{PSNR}|, 1.35\}, 0.65\}\lambda_B$; // update $\lambda_B$
  counter = counter + 1;
**until** ($|\Delta\text{PSNR} - 0.30dB| < 0.10$) **and** (counter > 10)
```

During the λ initialisation stage, frames are repeatedly encoded while the estimates of λ_P and λ_B are updated. One of the most computationally expensive

[3] The target PSNR for the P-frame is 0.70 dB less than that of the I-frame, with a tolerance window of ±0.25 dB.

parts of the coding process is the block partitioning and motion estimation stage. This is particularly true in the case of H.264/AVC VSBM as well as the Binary Partition Tree block matching method. It was therefore decided to replace these two partitioning methods with 16×16 regular block matching, but only during the λ *initialisation* phase.

This means that the two non-regular block matching approaches are initialised with slightly inaccurate λ values. However, the consequences of this were found to be negligible, since λ_P and λ_B are updated frequently when the codec operates on subsequent frames in the sequence (using the user-specified motion compensation method).

B. Updating Lambda

The initial λ values provide some control over the relative quality of I, P and B frames in the sequence. Because the scene characteristics change over time, it necessary to update λ_P and λ_B throughout the sequence, so as to maintain the target quality relationships between frames.

The updating of λ_P and λ_B occurs after the coding of each P-frame. If the three most recent frames are B_1, B_2 and P, and the previous intra frame is I (with coded versions B_1', B_2', P' and I'), then the process of updating λ is as follows:

Algorithm for Updating $\boldsymbol{\lambda_P}$ and $\boldsymbol{\lambda_B}$:

```
    $\Delta\text{PSNR} = \frac{1}{2}(\text{PSNR}(B_1, B_1') + \text{PSNR}(B_2, B_2')) - (\text{PSNR}(P, P') - 0.30)$;
    $\alpha = max\{min\{1 + \Delta\text{PSNR}|\Delta\text{PSNR}|, 1.35\}, 0.65\}$; // for updating $\lambda_B$
    $\Delta\text{PSNR} = \text{PSNR}(P, P') - (\text{PSNR}(I, I') - 0.70)$;
    $\gamma = max\{min\{1 + \Delta\text{PSNR}|\Delta\text{PSNR}|, 1.35\}, 0.65\}$; // for updating $\lambda_P$
    $\lambda_P = \gamma\, \lambda_P$;
    $\lambda_B = \alpha\, \gamma\, \lambda_B$;
```

This helps to maintain the quality of P and B frames close to their respective targets of 0.7 and 1.0 dB less than the PSNR of the previous I-frame. Note that as in the initialisation phase, λ is restricted to vary by at most 35% at each update. Preventing very large changes in λ helps to ensure that the picture quality does not vary suddenly from one frame to the next. (However, once initialised, large changes in λ seldom occur, unless there is a sudden change such as a scene cut.)

7.2.4 Entropy Coding

The proposed hybrid video codec uses a *Range Coder* [62] to achieve entropy coding. This is an integer-based implementation of arithmetic coding. It is virtually identical to an arithmetic coder, but instead of requiring a probability model when coding, it uses an (integer-valued) histogram. This allows for faster operation, with virtually no loss in coding performance.[4]

With a range coder, binary arithmetic coding is also possible using a two-bin histogram. This is particularly useful when binarising a value (such as a motion vector component) prior to encoding it. Most components of the proposed hybrid video coding system use the range coder for compressing data. They supply it with a histogram (probability model) that is also used by the corresponding component at the decoder.

7.3 Intra-Frame Coding

Intra coding is used to code frames without reference to any other frames. As explained earlier, the proposed hybrid video codec uses an intra frame at the start of each 15-frame GOP. The main advantages of using regular intra frames are allowing the decoder frequent access points in the bit-stream, and also helping to prevent the propagation of errors.

The primary focus of the research described in this dissertation is content-based motion estimation. It was therefore decided to use an off-the-shelf method of

[4] A general-purpose MATLAB Range Coder can be downloaded from [99].

image coding to achieve compression of intra frames. Both JPEG-2000 [34, 115] and H.264/AVC Intra [39, 43] were considered for incorporation into the codec. The latter was chosen because of its slightly superior performance at low and medium spatial resolutions (such as the CIF-format sequences used in testing) [60].

When performing intra coding, the video codec simply codes the image using H.264/AVC (version JM 9.5), and appends the compressed output bytes to the bit-stream. Note that the user-specified QP value is used when performing this operation.

7.4 Inter-Frame Coding

In general, inter frames can be coded much more efficiently than intra frames, since they are predicted from reference frames, rather than having to be coded from scratch. This section describes how blocks are generated (using one of four block partitioning approaches), and how these blocks are used for motion compensation. The coding of the resulting Displaced Frame Difference (DFD) is also discussed.

7.4.1 Block Partitioning and Motion Estimation

The proposed video codec was designed with the intention of allowing four block partitioning methods. (These are: fixed-size 16×16 blocks, fixed-size 8×8 blocks, the H.264/AVC method of VSBM, and the Binary Partition Tree approach to VSBM.) The remainder of the codec is intended to operate in a constant way — independently of the partitioning choice specified by the user.

A. Using Fixed-Size Blocks

Two of the block partitioning options available to the user are fixed-size 8×8 and 16×16 blocks. These two cases result in the creation of a regular grid

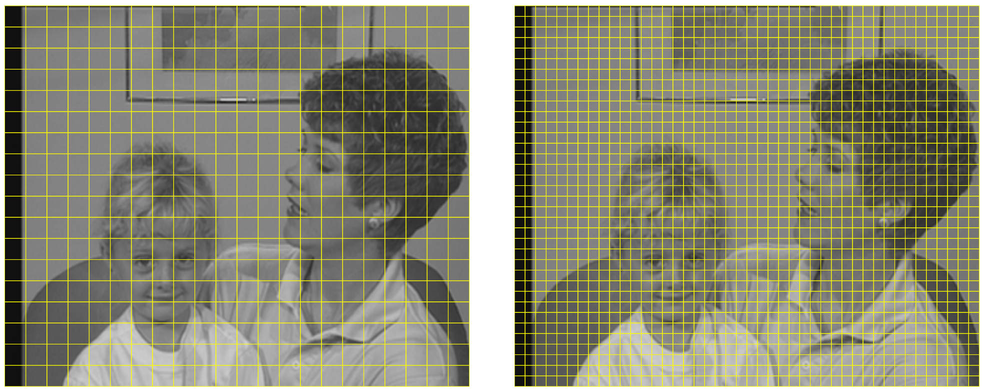

(a) 396 16×16 blocks (b) 1584 8×8 blocks

Figure 7.4: An example of using fixed-size blocks for a frame of size 352×288

of square blocks that span each inter frame. Recall that the number of rows and columns in the input video are required to be integer multiples of 16. As a result of this constraint, there are always a whole number of 8×8 or 16×16 blocks present in each frame. Figure 7.4 provides an example of two grids of fixed-size blocks, overlaid on a frame of the CIF-resolution *Mother & Daughter* sequence.

Pixel-accurate motion estimation is performed using full-search block matching, relative to the reference frame(s). In the case of B-frames (which have two reference frames), the reference frame which yields the smaller motion compensation error for a particular block is chosen as the reference frame for that block. Following this, each motion vector is refined to sub-pixel accuracy, as is described later in this section.

B. Using H.264/AVC Variable-Size Blocks

As described in Section 2.7.8, one of the significant features of H.264/AVC is its use of VSBM. This allows the size and shape of blocks to adapt (in a limited way) to the characteristics of a scene. Each 16×16 macro-block is partitioned in one of the following four ways (as illustrated in Figure 7.5):

- no further partitioning;

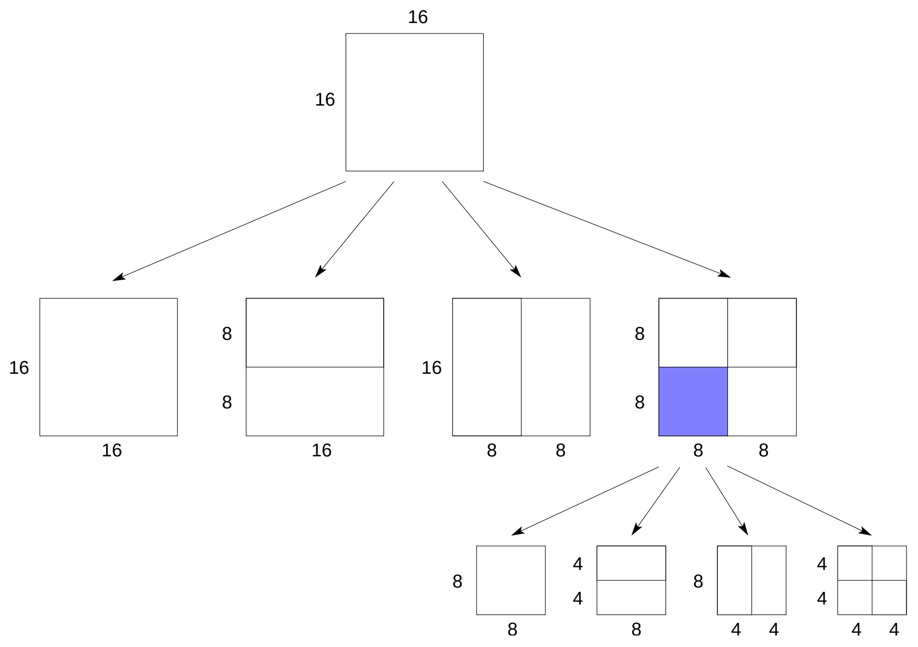


Figure 7.5: H.264/AVC block splitting modes

- divide into two equal rectangular halves using a horizontal partition;
- divide into two equal rectangular halves using a vertical partition; and
- divide into four equal square quarters.

If the four-block partitioning mode is chosen, then each of these blocks is considered in turn for further splitting using the same method outlined above. Thus if a macro-block is split into four blocks, which are themselves all partitioned into four blocks, one ends up with 16 4×4 blocks in the macro-block.

At each stage, the decision on which type of partition to choose is based on optimising the rate-distortion performance. Thus splitting a block into four children may lead to a reduction in distortion (i.e. motion compensation error), but it is also likely to result in an increase in rate due to there being four motion vectors to code (instead of just one). Choosing between one or four blocks in this scenario depends on the relative importance of rate and distortion.

In the proposed video codec, Lagrangian rate-distortion optimisation is used to help solve this problem. For each macro-block, the encoder considers the four top-level partitioning methods (none, horizontal, vertical, and four-way splitting).

In each of these four cases, the resulting rate (R) and distortion (D) are calculated. The rate comprises the cost of coding the following: the block partitioning mode, the choice of reference frame (in the case of B-frames), and the motion vectors (to quarter-pixel accuracy).[5] Distortion is measured using the sum of squared error (SSE) after motion estimation at quarter-pixel accuracy. Using the current frame's value of λ, it is then possible to find the best of the four partition options by determining which yields the smallest value of $J = D + \lambda R$.

If the four-way splitting mode out-performs the other three choices, further splitting of the resulting four blocks needs to be considered. This is done by re-applying the process outlined in the previous paragraph to each of the four 8×8 blocks. Using Lagrangian optimisation, the best partition is then found for each of these four blocks.

The end result is the optimal splitting of each macro-block, subject to the constraints of the H.264/AVC partitioning method. Figure 7.6 provides an example of H.264/AVC block partitioning. The stationary background areas are dominated by un-partitioned blocks, while smaller blocks populate regions of more complex motion, particularly along the edges and textured areas of moving objects.

Motion estimation is a critical part of the block splitting process and is performed in parallel with it. For each block (or potential block), the (sub-pixel) motion estimation process provides a motion vector and the associated motion compensation error. These are used directly when determining the rate and distortion properties of a block.

[5] Motion vectors and reference frame information are coded predictively, based on previously-coded neighbouring blocks. For further details, refer to Section 7.4.3.

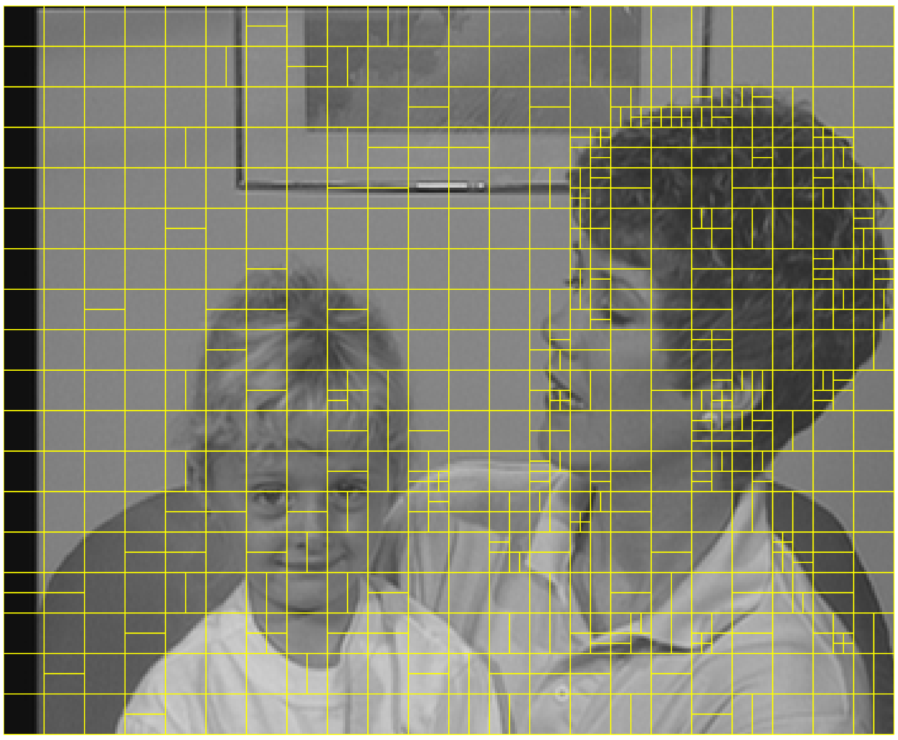
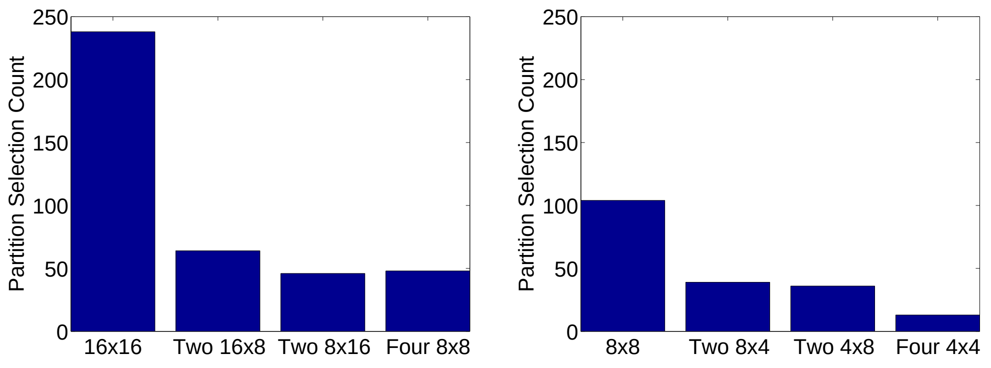


Figure 7.6: An example of H.264/AVC block partitioning applied to frame 50 of *Mother & Daughter*. Using $\lambda_B = 20$, and motion compensating from frames 48 and 52 yields a block structure with 764 blocks.

Coding the H.264/AVC Block Partition Structure:

The ways in which the partition choice, reference frame selection, and motion information are coded have some bearing on the macro-block partitioning process, since they influence the bit-rate (and thus the value of J). The coding of motion vectors and reference frame information is addressed later in the chapter (see Section 7.4.3), but the modelling and coding of the partition information are discussed here in more detail.

Macro-block partition information is modelled using two four-bin histograms. The first histogram represents the probability of partitioning a 16×16 macro-block using each of the four possible partitioning methods. Similarly, the second histogram is used to indicate the probability of splitting an 8×8 sub-block, using the four permissible block division methods. In both histograms, the four



(a) First-level (16 × 16) partitioning

(b) Second-level (8 × 8) partitioning

Figure 7.7: Histograms of block partition choices for the block structure shown in Figure 7.6. At each stage, four H.264/AVC splitting options are possible.

bins represent: no splitting, horizontal partitioning, vertical partitioning, and four-way splitting.[6]

To start with, all splitting options are assigned equal probability, but the model is updated as more and more macro-blocks are processed. This allows for both the encoder and decoder to adapt to statistics in the same way. Entropy coding the partition information using this two-histogram model allows for the variable-size block structure to be coded efficiently.

As an example, the two histograms in Figure 7.7 demonstrate the distribution of probabilities for the block partition structure shown in Figure 7.6. Note that the sum of all bins in the second histogram is four times the value of the fourth bin in the first histogram. The total cost of encoding the partition information in this case is 968 bits, which is equivalent to just 1.27 bits per block.

C. Using a Binary Partition Tree

The use of a Binary Partition Tree approach to variable size block matching was discussed in detail in Chapter 6. Starting with one block that covers the

[6] Note that the second histogram is only necessary when the first level of partitioning corresponds to four-way splitting.

entire frame, this method operates by repeatedly splitting blocks in a way that maximises the reduction in error (subject to the constraints of straight-line partitioning).

Recall that binary tree partitioning requires the user to specify the target number of blocks. An extra 25% of blocks are created when growing the binary tree, and those partitions that provide the least gain are then pruned from the tree.

However, when used in a video codec, it should not be left up to the user to select a target number of blocks for each frame. Also, it is not necessarily optimal for the codec to use a fixed number of blocks across all frames. Rather, the system itself should choose a reasonable (and hopefully optimal) number of blocks with which to perform motion compensation.

It was therefore decided to adopt the following approach when growing and pruning the Binary Partition Tree for each frame:

- Grow the tree to a fixed number of blocks, N_b. This number is a function of the picture size and the frame type (P or B). It is given by the equation $N_b = \alpha N_R N_C / 256$, where N_R and N_C are the number of rows and columns respectively, and α has a value of 2 for P-frames and 1 for B-frames.[7]

- The Binary Partition Tree is then pruned several times, each time resulting in a different number of blocks. At each stage, the encoder simulates the remainder of the encoding and decoding process for that frame, in order to determine the resulting rate and distortion. Thus if n is the number of blocks after pruning, the encoder calculates $J(n) = D(n) + \lambda R(n)$ for a range of different n, in order to find the value of n that yields the smallest $J(n)$.

[7] Thus for B-frames, the number of blocks is equal to those in fixed-size 16×16 block frames, while for P-frames the number is twice as many. Note that this is only true during the initial phase, since some of these blocks are subsequently pruned from the tree.

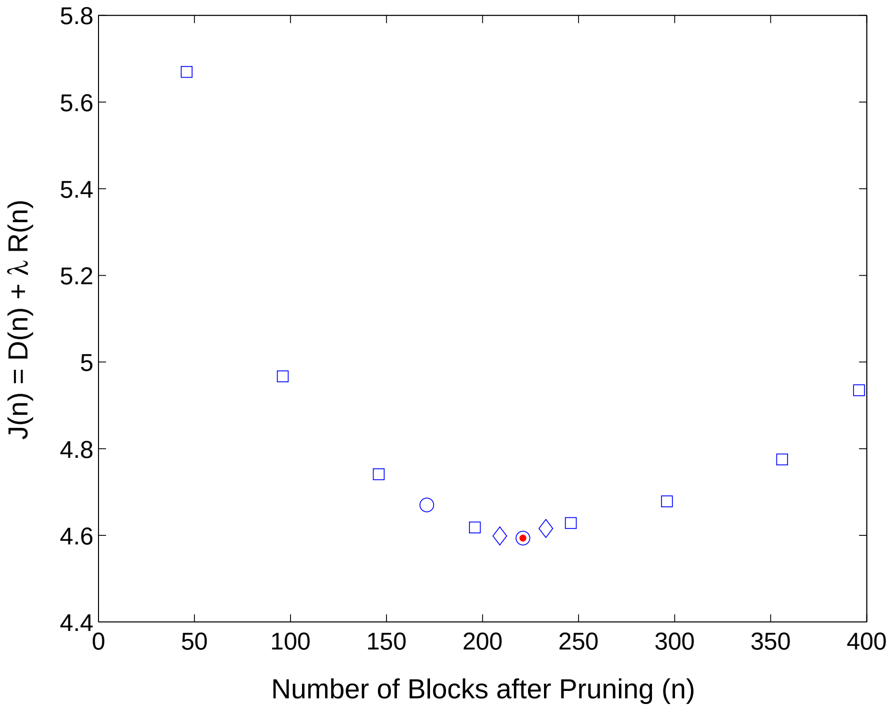


Figure 7.8: An example of determining the optimal number of blocks when using the Binary Partition Tree method of motion compensation. The values of n tested during the first, second and third phases are represented as squares, circles and diamonds, respectively. The red dot shows n_3, the optimal number of blocks. The results shown here are for frame 50 of the *Mother & Daughter* sequence, motion compensated from frames 48 and 52, and using $\lambda_B = 20$.

- A total of twelve different values of n are tested. During the first phase, the encoder determines $J(n)$ for eight of these values of n. It does this by pruning the tree from the original N_b leaf nodes down to N_b, $\frac{7}{8}N_b$, $\frac{6}{8}N_b$, $\ldots$, $\frac{2}{8}N_b$ and $\frac{1}{8}N_b$ leaf nodes (i.e. blocks). Figure 7.8 provides an example of the process for a tree pruned from 396 blocks. The points marked with blue squares represent the eight values of n tested in the first phase. Of these points, the smallest value of $J(n)$ is obtained when $n = 198$.

- The second phase then proceeds as follows: Using the best value of n from the first phase (call this n_1), the encoder tests two values of n on either side of n_1. The two values tested are $n_1 - \frac{1}{16}N_b$ and $n_1 + \frac{1}{16}N_b$.[8] Of the ten points tested thus far, the one yielding the minimum value of $J(n)$ is labelled as n_2. Continuing with the example in Figure 7.8, one can see that the two points tested during the second phase (represented with blue circles) yield a value of $n_2 = 222$.

- The third and final phase continues in a similar way. The encoder tests two values of n on either side of n_2. The two values tested are $n_2 - \frac{1}{32}N_b$ and $n_2 + \frac{1}{32}N_b$.[9] Of the twelve points tested, the one yielding the minimum value of $J(n)$ is labelled as n_3. Completing the example in Figure 7.8, one can see that the two points tested during the third phase (represented with blue diamonds) provide no further reduction in $J(n)$. Therefore, in this example, $n_3 = n_2 = 222$.

Having obtained a value of n_3 blocks that minimises $J(n)$ over the twelve points tested, the tree is then pruned from N_b down to n_3 leaf nodes (blocks), and the structure is encoded in the bit-stream. The resulting n_3 blocks are then used to motion compensate the current frame.

D. Block Sorting

Once the blocks for a frame have been generated, it is necessary to sort them. This serves two purposes. First, it allows the encoder and decoder to process blocks in the same order, which is essential if the codec is to function correctly. Secondly, blocks tend to have similar characteristics (such as motion vectors) to nearby blocks. Thus, it is advantageous to predict the properties of blocks from those of their previously-coded neighbours. This can help to significantly reduce the bit-rate.

The approach adopted by the proposed video codec is to sort blocks spatially in raster-scan order, based on the location of their top-left corner. This provides

[8] Note that If $n_1 = N_b$ then only the lesser of the two values is tested.

[9] Note that If $n_2 = N_b$ then only the lesser of the two values is tested.

a simple and fast way of sorting blocks, and one that works for each of the four block generation methods used by the codec. As part of the sorting process, each block is assigned a set of references to those of its (spatial) neighbours that precede it in the sorted list.

E. Sub-Pixel Motion Estimation

When using fixed-size blocks, the shape and size of blocks is independent of the motion estimation process. However, for the two methods using variable-size blocks, the motion estimation process is an integral part of determining the structure of blocks in the frame.

The Binary Partition Tree method performs block splitting based on pixel-accurate motion vectors only. Therefore, motion estimation to sub-pixel accuracy is not required during this stage.

However, in the case of H.264/AVC VSBM, quarter-pixel-accurate motion vectors (and their corresponding motion compensation error) are used when determining how to partition a block. It was found that using only pixel-accurate motion vectors at this stage resulted in a less effective rate-distortion optimisation process.

With all four of the methods used,[10] motion vectors are eventually estimated to quarter-pixel accuracy. (For methods other than H.264/AVC VSBM, this happens *after* the block generation phase.) In the case of B-frames, the choice of reference frame is performed after pixel-accurate motion estimation. Quarter-pixel-accurate motion estimation is then carried out relative to the selected reference frame only.

For both P and B frames, full-search block matching is used when estimating a block's motion vector to pixel accuracy. Motion estimation is then performed to quarter-pixel accuracy by searching within the immediate neighbourhood

[10] Fixed-size 16×16 blocks, fixed-size 8×8 blocks, the H.264/AVC method of VSBM, and the Binary Partition Tree approach to VSBM.

of the pixel-accurate motion vector.[11] During this process, the encoder stores the SSE values for the best pixel, half-pixel and quarter-pixel accurate motion vectors. These are used at a later stage to optimise (in a rate-distortion sense) the coding of the sub-pixel motion vector information.

7.4.2 Block Mode Selection

Some areas in a scene can be difficult to motion compensate. This can be for several reasons, two of the most common being occlusion and changes in illumination. If a block covers an area affected in this way then the motion estimation process will yield a poor motion vector with a very high motion compensation error.

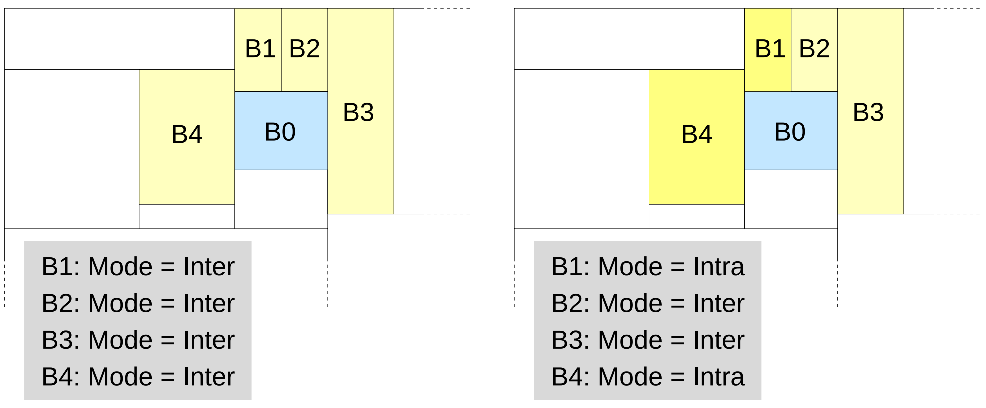
The proposed video codec therefore allows the option of *intra blocks*, which are not motion compensated from a reference frame, but instead are coded directly. The information conveyed in an intra block is fairly crude. Only the DC (average) values of the luma and two chroma components are coded, and no texture information is conveyed. Any significant texture is left to be coded as part of the residual image.

The mode of each block is indicated by an intra/inter flag, which is determined by the encoder using Lagrangian rate-distortion optimisation:

- In the case of an inter block, the distortion is measured using the motion compensation error (more specifically, the SSE). The rate is determined by the cost of coding the following: the inter flag, the choice of reference frame (for B-pictures), and the motion vector.
- For an intra block, the distortion is measured as the energy of its AC component (i.e. the SSE between the block and its DC value). The rate comprises the cost of coding the intra flag plus the 24 bits required to represent the luma and chroma DC values.[12]

[11] Bi-linear interpolation is used to estimate pixel values at non-integer positions.

[12] The luma and chroma DC values are each represented using eight bits (at integer precision).



(a) With no intra neighbours, block B0 is assumed to have a $^1/_{32}$ chance of being intra.

(b) With two intra neighbours, block B0 is assumed to have a 50% chance of being intra.

Figure 7.9: Two examples of predicting the (intra or inter) mode of a block, B0, based on the modes of its previously-coded neighbours (B1 to B4).

Because there is a high degree of redundancy between neighbouring frames, there are generally very few intra blocks per frame. This difference in probability between intra and inter modes is modelled when entropy-coding the type of block.

If a block has no previously-coded intra neighbours, then the probability of it being intra is assumed to be $^1/_{32}$. However, if it has n neighbouring intra blocks (with $n > 0$), then the chance of it also being intra is estimated at $\frac{n}{n+2}$. Note that this model assumes that intra blocks are more likely to be clustered within the same regions, which was generally found to be the case. Figure 7.9 provides examples for each of the two scenarios.

The use of intra blocks allows the video codec to handle another potential problem with block matching. As with most other codecs, the one proposed in this chapter performs motion estimation using only the luma component of each frame. This allows for faster computation than considering all three components, however it can result in erroneous matching.[13]

[13] For example, two areas may match closely in luma characteristics, but have significantly different chroma components. In this case, motion estimation would return a good match (low error), though this would clearly not be the case for a human observer.

One way around this problem is to consider the SSE contributions of all three colour components when determining the mode of a block. By adding the chroma SSE into the rate-distortion calculations,[14] blocks with low luma error but high chroma distortion are more likely to be flagged as intra blocks. This approach requires minimal extra computational cost, while helping to remove some of the artifacts caused by luma-only matching.

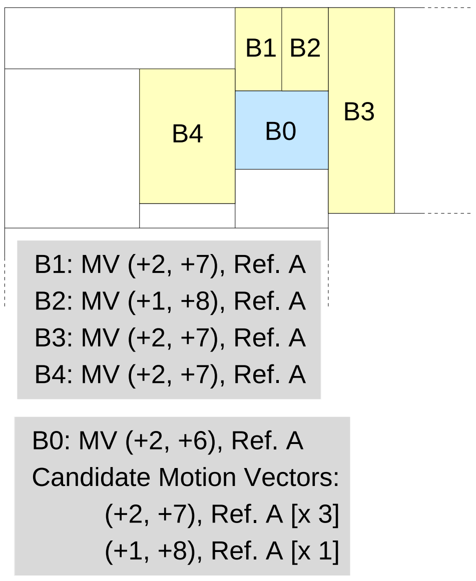
7.4.3 Coding Block Motion Information

With most codecs, the coding of motion vectors typically consumes a large portion of the bit-stream for P and (especially) B frames. The proposed video codec operates by trying to predict three aspects of motion information for each block: the choice of reference frame (for B-pictures); the (pixel-accurate) motion vector; and the sub-pixel refinement of the motion vector. The process is described in detail below, and accompanied by two examples in Figure 7.10.

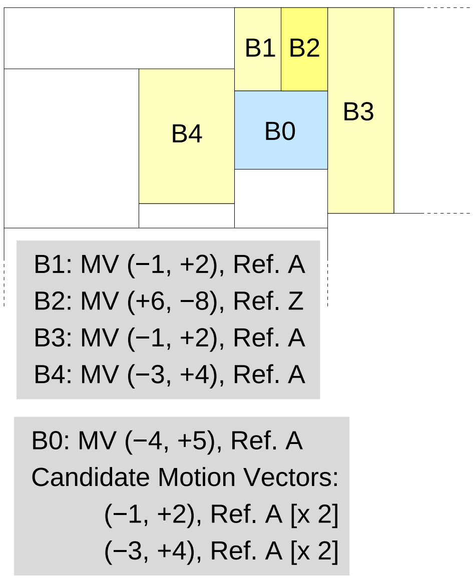
- For a given inter block, find its previously-coded neighbouring inter blocks (i.e. ignore any neighbouring intra blocks). The two examples in Figure 7.10 assume that B0 is the current block, and its previously-coded neighbouring inter blocks are labelled B1 to B4.

- In the case of B frames, the choice of reference frame needs to be coded. The model for this is based on all previously coded inter blocks in the frame (and not just neighbouring ones). Let the two reference frames be referred to as A and Z. Furthermore, assume that the number of previously-coded blocks having used the two reference frames is N_a and N_z, respectively. Then the model assumes that the probability of the current block using frame A as its reference is $(N_a+1)/(N_a+N_z+2)$. Similarly, the probability of frame Z being the reference frame is assumed to be $(N_z+1)/(N_a+N_z+2)$.

[14] Note that because the chroma components are half the (horizontal and vertical) resolution of the luma component, they effectively each have a weighting of $1/4$ of the luma contribution.



(a) Using one reference frame (A)



(b) Using two reference frames (A and Z)

Figure 7.10: Two examples of predicting the motion properties of a block, B0, based on the characteristics of its previously-coded neighbours (B1 to B4).

- The next step is to determine a list of *candidate* motion vectors that are used to predict the motion vector of the current block. Candidate motion vectors are derived from the motion vectors of previously-coded neighbouring blocks. For blocks with the same reference frame as the current block, their motion vector is added to the list of candidate motion vectors.

  Figure 7.10(a) provides an example for a P-frame, in which all blocks use the same reference frame (A). In this case, three of block B0's neighbours (B1, B3 and B4) use the same motion vector, $(+2, +7)$. This is added to to the candidate list, and its count is set to 3. Block B2 contributes one other motion vector, $(+1, +8)$, to the list, and its count value is set to 1.

- The process is similar, but slightly more complicated in the case of B-frames. Once again, blocks with the same reference frame as the current block have their motion vector added to the list of candidate motion vectors. Neighbouring blocks using a different reference frame to the current block can still be used to predict its motion vector if a constant velocity assumption is made. These neighbouring blocks have their motion vectors reversed and scaled before being added to the candidate motion vector list. The reversal is because the two reference frames lie on either side of the current frame (i.e. one in the past and one in the future). The scaling is because the reference frames are not necessarily an equal distance on either side of the current frame.

This is perhaps best illustrated using the example in Figure 7.10(b). There are two reference frames (A and Z) and the current block, B0, uses frame A as its reference. Consequently, all motion vectors added to the candidate list need to be expressed relative to reference frame A. In this example, two of block B0's neighbours (B1 and B3) use the same motion vector, $(-1, +2)$. This is added to to the candidate list, and its count is set to 2. Block B4's motion vector, $(-3, +4)$, is also added to the list, but with a count of one.

The remaining neighbouring block, B2, uses reference frame Z. As described above, its motion vector, $(+6, -8)$, therefore needs to be reversed and scaled. For this example, it is assumed that reference frame A occurs *one* frame *before* the current frame, while reference frame Z is positioned *two* frames *after* the current frame. Thus, in order to express the motion vector relative to reference frame A, it needs to be multiplied by $-1/2$. This yields the normalised motion vector $(-3, +4)$, which is already an entry in the candidate motion vector list. The count value of this entry in the list is thus incremented by one.

- Once the list of candidate motion vectors has been calculated,[15] it can be used to code the motion vector of the current block. The encoder performs this step by finding the candidate motion vector that is closest to the current block's motion vector. It then codes the index of the chosen

[15] Note that the process of calculating the list of candidates is identical at both the encoder and decoder.

candidate and the difference between it and the current block's motion vector. (The horizontal and vertical difference components are binarised and encoded using binary arithmetic coding.)

The process of coding the index of the chosen candidate motion vector is based on the count values of each entry in the list. Thus, the more frequently-occurring candidates are assigned a higher probability and can consequently be coded more efficiently. If there is only one candidate motion vector present, its index does not need to be coded. Also, if there are no candidate motion vectors for a particular block, the coder uses a dummy vector of $(0, 0)$.

Returning to the example in Figure 7.10(a), it is evident that the closest candidate vector to block B0's motion vector of $(+2, +6)$ is $(+2, +7)$. This is the first candidate vector in the list, and so an index of 1 is coded. (Note that this entry has a probability of 75%.) The horizontal and vertical components of the motion vector difference, $(0, -1)$, are then coded using binarisation and arithmetic coding.

For the example in Figure 7.10(b), it can be seen that the closest candidate vector to block B0's motion vector of $(-4, +5)$ is $(-3, +4)$. This is the second candidate vector in the list, and so an index of 2 is coded. (Note that this entry has a probability of 50%.) The horizontal and vertical components of the motion vector difference, $(-1, +1)$, are then coded using binarisation and arithmetic coding.

- The final step is the coding of each motion vector to sub-pixel accuracy. Recall that during the motion estimation stage, SSE values are retained for the best pixel, half-pixel and quarter-pixel accurate motion vectors of each block. Using Lagrangian rate-distortion optimisation, a block is only coded to sub-pixel accuracy if the reduction in SSE is significant enough to warrant it. If not, the encoder forces the sub-pixel component of the motion vector to zero prior to coding.

7.4.4 Overlapped Block Motion Compensation

The goal of the motion estimation process is to choose the optimal motion vector for each block. However, as described in Section 2.5.3, it is possible to improve the reconstruction quality of a motion compensated picture by allowing blocks to overlap.

Overlapped Block Motion Compensation (OBMC) helps to significantly reduce blocking artifacts that occur along the edges of blocks (and which are common in many block based codecs). No extra processing is required during the motion estimation stage, and no additional information needs to be encoded in the bit-stream. Instead, OBMC is performed by the decoder during the motion compensation stage (and in the encoder's reconstruction loop).

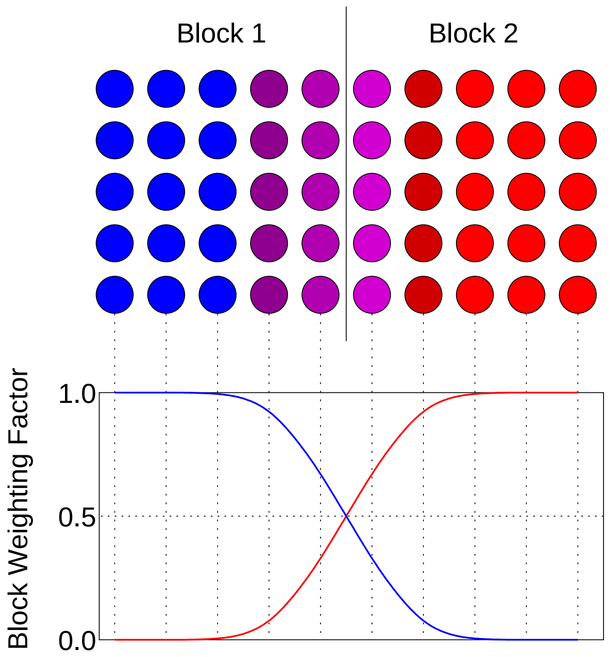


Figure 7.11: An example of two overlapping blocks used for OBMC. The red and blue circles represent pixels in two blocks, and the two curves below show the raised-cosine weighting factors for each block.

The video codec described in this chapter uses OBMC with an overlap of two pixels on either side of each block boundary. A raised-cosine mask in each overlap zone determines the weighting factors of each block. At each point in the image, the weights are normalised so that they sum to unity. Figure 7.11 provides an example of two blocks that overlap along their common boundary.

The codec allows intra blocks to overlap into neighbouring inter and intra blocks. In this case, the (weighted) DC components of an intra block contribute to the values of surrounding pixels. Similarly, inter blocks are allowed to overlap into neighbouring intra blocks.

7.4.5 Residual Coding

The process of motion compensation produces a reconstructed frame that is similar to the original. However, it is still likely to contain a number of significant artifacts that are not accurately modelled by block-based prediction or intra coding.

Transform coding is typically used to code the Displaced Frame Difference (DFD), which is the residual image after motion compensation. Most codecs tend to code residuals using the same transform employed for the coding of intra frames. However, the video codec proposed in this chapter uses a method based on *Matching Pursuit* to code the residual. (An overview of Matching Pursuit is provided in Section 2.4.3.)

This decision was made for two reasons. First, the software implementation of H.264/AVC does not accept 9-bit difference images for intra coding. Secondly, the Matching Pursuit technique has been reported to be a highly effective way of coding *difference* frames [68]. It was therefore decided to use an off-the-shelf Matching Pursuit coder. The MATLAB code of Jost *et al's* Tree-Based Pursuit algorithm [42, 78] was downloaded and integrated into the video codec.

Matching-Pursuit performs quite slowly on large or even medium sized images. Therefore the codec first breaks down each difference frame into a number of tiles, before coding each tile using Matching Pursuit. A residual image of size $N_C \times N_R$ is split into tiles of size $\left(2^{\lceil 6-\log_2(N_C)\rceil}\right) N_C \times \left(2^{\lceil 6-\log_2(N_R)\rceil}\right) N_R$ prior to being coded. This limits the height and width of tiles to less than 128 pixels.

Note that the coding of each tile is performed using the current frame's λ value. This ensures that tiles are coded at the same rate-distortion tradeoff point. The chroma components of each residual frame are also split into tiles, and coded using Matching Pursuit.

The λ parameter also helps to control the number of atoms used to approximate each tile.[16] In order to reduce computation, tiles in P and B frames are restricted to using a maximum of 100 and 50 atoms respectively. Note that tests were performed at low bit-rates (using a high value of QP), so that more atoms were found to be unnecessary.

7.5 Results and Analysis

The video codec proposed in this chapter was implemented in MATLAB, with certain functions written in C to help speed up the encoding process. (Source code, and a MATLAB demo are available on the project web-site [99].) The codec was tested on standard video sequences in order to compare the rate-distortion performances of the four different block matching methods.

Nine sequences[17] were used in testing. This section presents results for the *Foreman*, *City*, *Mobile & Calendar* and *Stefan* sequences (which are representative of the results obtained for the full set of nine sequences). These four sequences all depict scenes with fairly complex motion, and the latter three also contain significant spatial detail. The *Foreman* and *City* sequences have a spatial resolution of 352×288 (CIF), while the resolution of *Mobile & Calendar* and *Stefan* is 352×240 (SIF). All four sequences play at a frame rate of 30 fps and use YC_bC_r 4:2:0 chroma sub-sampling.

For each sequence, encoding and decoding were performed using all four block

[16] An atom is a Matching Pursuit basis function together with its position and weighting.

[17] Namely *City*, *Flower Garden*, *Football*, *Foreman*, *Mobile & Calendar*, *Stefan*, *Crowd*, *Edinburgh* and *Tennis*. The first six are standard test sequences, while the latter three are proprietary BBC sequences.

generation methods, at a variety of quantisation (QP) settings.[18] Note that all of the sequences referred to in this chapter (as well as the uncompressed originals) are included on the accompanying DVD (Appendix D).

7.5.1 Rate-Distortion Performance

Rate-distortion results were obtained by averaging over the first 100 frames of each sequence, and are plotted in Figures 7.12 to 7.15. It can be seen that in each case the Binary Partition Tree VSBM method out-performs H.264/AVC VSBM, which in turn is superior to the two FSBM methods. However, the amount of gain varies from sequence to sequence and is also dependent on bit-rate.

Regular 8×8 block matching performs poorly overall, particularly at low bit-rates. This is because of the substantial overhead associated with coding the motion vectors of many small blocks. Fixed-size 16×16 block matching uses only a quarter of the number of blocks. Thus, although the motion compensation is more crude, there are more bits left over to either refine the motion vectors to sub-pixel accuracy, or to spend coding the residual.

The one exception to the superior performance of 16×16 over 8×8 block matching is in the coding of the *Stefan* sequence at higher bit-rates (see Figure 7.15). This is partly due to the presence of non-translational motion in highly-textured background areas. Another factor is the effect of the thin black lines along the top and right borders of the picture. As discussed in Chapter 6, 16×16 blocks are unable to resolve the motion in this area, leading to significant distortion. However, the use of smaller blocks allows for motion to be modelled more accurately, thus reducing the error along these frame boundary areas by a substantial amount.

The H.264/AVC implementation of VSBM out-performs both FSBM methods

[18] The following quantisation parameter (QP) settings were used for each of the sequences: *Foreman*: 28, 30, 32, 34, 36; *Stefan*: 36, 37, 38, 39, 40; *City*: 30, 31, 32, 34, 36; and *Mobile & Calendar*: 38, 39, 40, 41, 42.

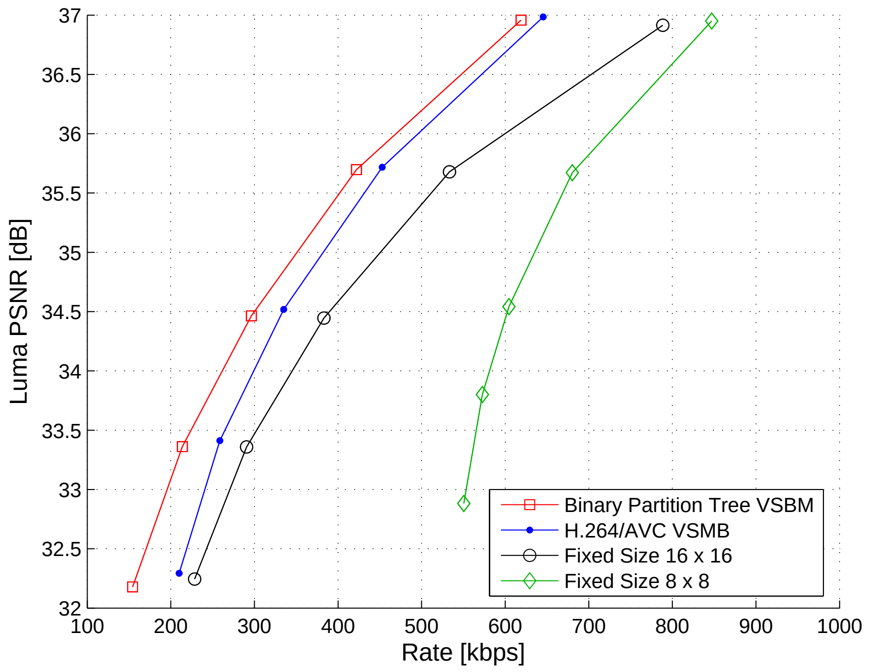


Figure 7.12: Rate-Distortion results for the first 100 frames of *Foreman*

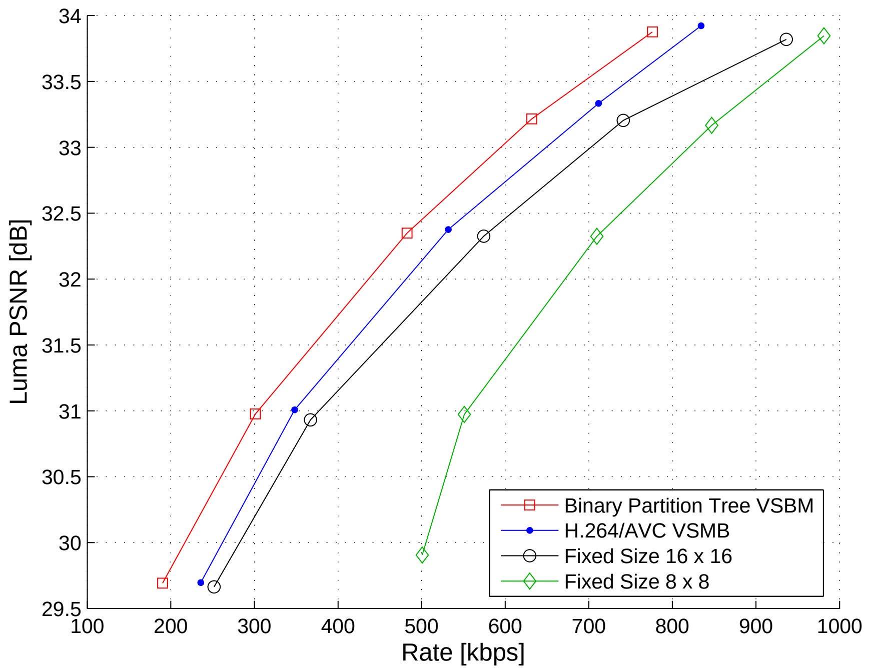


Figure 7.13: Rate-Distortion results for the first 100 frames of *City*

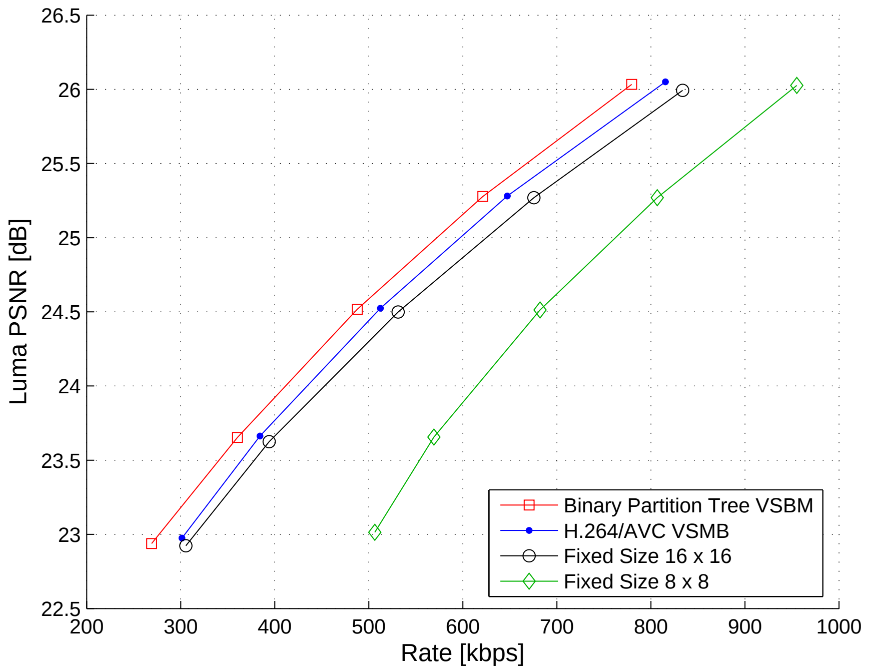


Figure 7.14: Rate-Distortion results for 100 frames of *Mobile & Calendar*

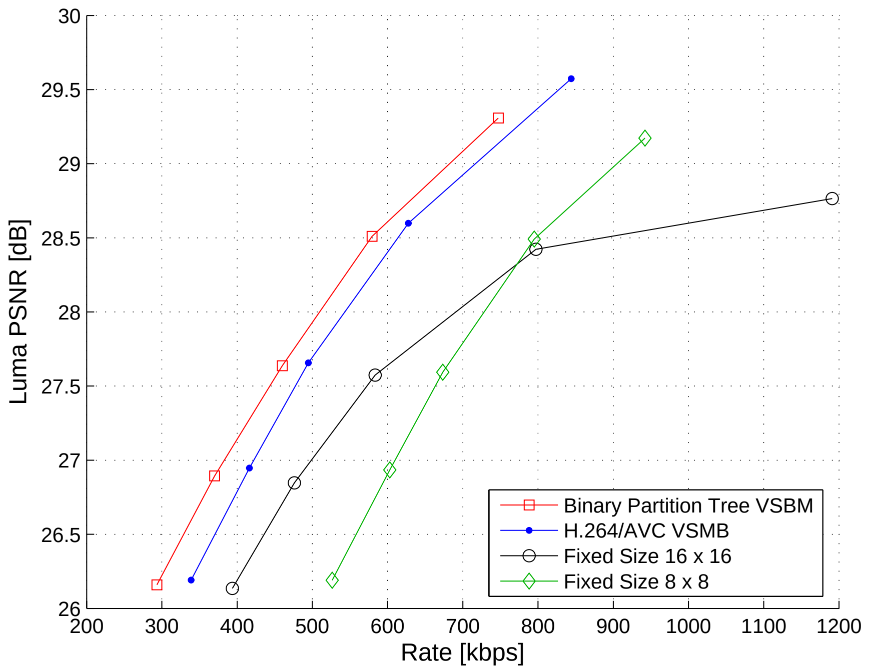


Figure 7.15: Rate-Distortion results for the first 100 frames of *Stefan*

for all values of QP. For all four sequences, the gain over 16×16 FSBM is more pronounced at high bit-rates than at lower ones. This can be understood as follows: When operating at low rates, H.264/AVC VSBM has few bits available to spend on specifying block partitions and the resulting additional motion vectors. Consequently, many block are not split and as a result the performance is closer to that of FSBM. However, at higher rates, the encoder is able to spend more bits on improving the motion compensation performance in areas of complex motion — something which FSBM is unable to do.

The Binary Partition Tree approach to VSBM in turn out-performs H.264/AVC VSBM across all sequences, for all the values of QP which were tested. Its advantage is most noticeable at low bit-rates, which is when relatively few blocks are used for motion compensation. This seems reasonable when one considers that the initial partitions in the tree tend to provide the largest reductions in error. Also, H.264/AVC does not have the option of using any blocks larger than 16×16, which could potentially be a disadvantage at low bit-rates.

Tables 7.1 and 7.2 provide the minimum and maximum gains of the Binary Partition Tree approach to block matching over its rival methods, for each of the four sequences.[19] It can be seen that the most significant advantage over H.264/AVC VSBM was found to be an increase of 1.0 dB in quality (at the same rate), or alternatively a 24% reduction in bit-rate (for the same average PSNR).

Table 7.1: Improvement in PSNR provided by Binary Partition Tree VSBM

Sequence	**relative to H.264/AVC VSBM**	**relative to FSBM**
Foreman	0.11 dB to 1.00 dB	0.76 dB to 1.35 dB
City	0.23 dB to 0.58 dB	0.52 dB to 0.80 dB
Mobile & Calendar	0.13 dB to 0.23 dB	0.27 dB to 0.31 dB
Stefan	0.11 dB to 0.42 dB	0.93 dB to 1.03 dB

[19] based on the results plotted in Figures 7.12 to 7.15

Table 7.2: Reduction in bit-rate provided by Binary Partition Tree VSBM

Sequence	**relative to H.264/AVC VSBM**	**relative to FSBM**
Foreman	3.4% to 24.4%	21.3% to 31.4%
City	5.9% to 19.2%	14.9% to 24.9%
Mobile & Calendar	3.8% to 9.2%	7.6% to 12.4%
Stefan	3.5% to 12.6%	18.1% to 27.9%

7.5.2 Frame-by-Frame Results

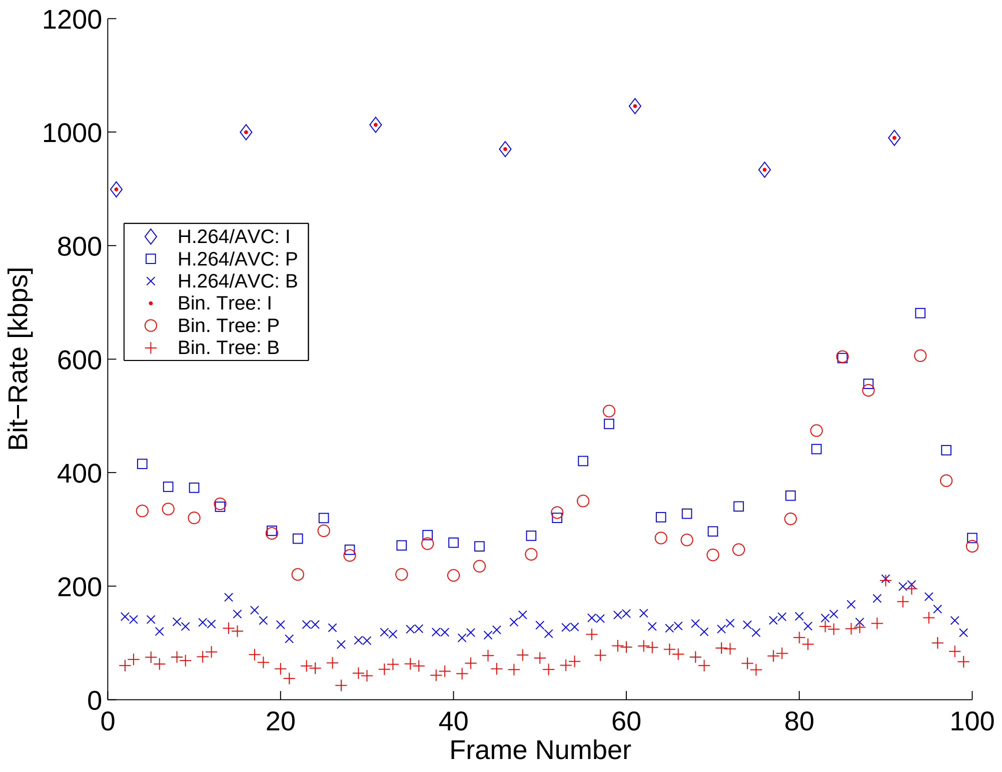
The rate-distortion results in Figures 7.12 to 7.15 are based on the average performance over 100 frames. It can also be revealing to examine the rate and distortion behaviour of individual frames. Figures 7.16 to 7.19 show the bit-rate and quality (PSNR) of each frame, using the two VSBM methods.[20]

The most noticeable characteristic of the bit-rate graphs is that the I, P and B frames fall into three distinct bands: I frames require the largest number of bits since they are coded independently of any other pictures; P frames need fewer bits since they are predicted from previous reference pictures; and B frames are bi-directionally predicted and thus require still fewer bits. In addition, the encoder operates by coding P and B frames with a target PSNR of 0.7 and 1.0 dB less than that of the previous I frame.

The frame-by-frame results allow for a comparison of H.264/AVC VSBM (plotted in blue) and Binary Partition Tree VSBM (displayed in red). Both methods demonstrate a very similar PSNR performance, which is not surprising when one considers that the encoder uses the same value of QP (and the same quality control mechanism) for both motion compensation techniques.

For the same average quality, Binary Partition Tree VSBM generally requires fewer bits to code inter frames than H.264/AVC VSBM. For example, when coding *Foreman* with QP set to 34 (as shown in Figure 7.16), the average cost of

[20] For each sequence, the frame-by-frame results are only shown for one selected value of QP.



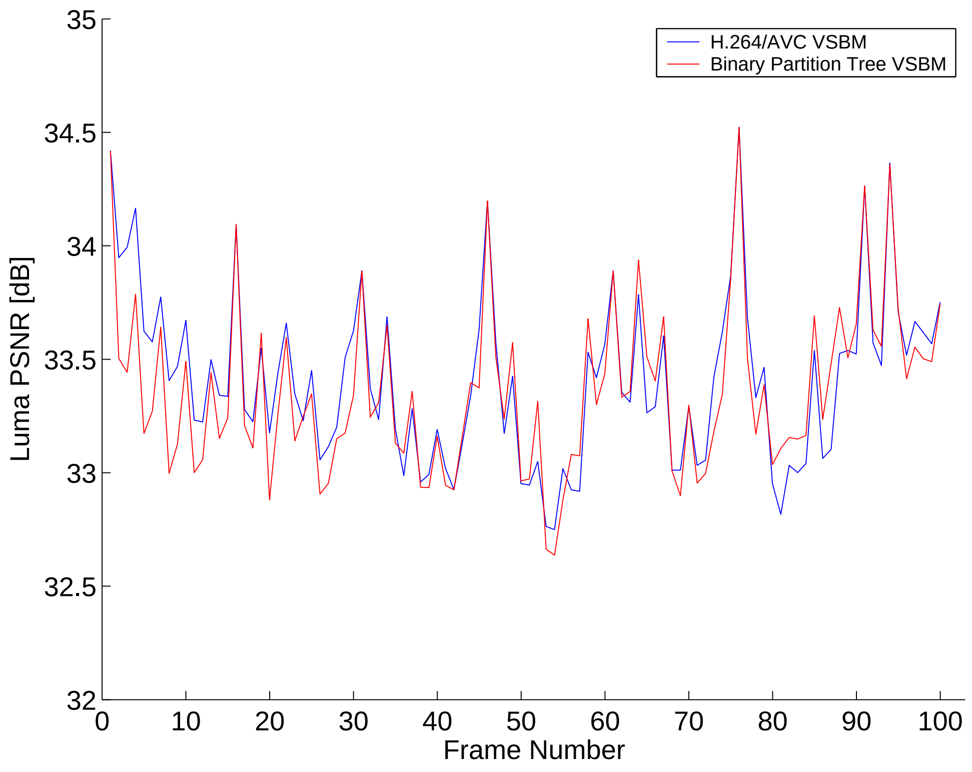


Figure 7.16: Frame-by-frame results for *Foreman* (QP = 34)

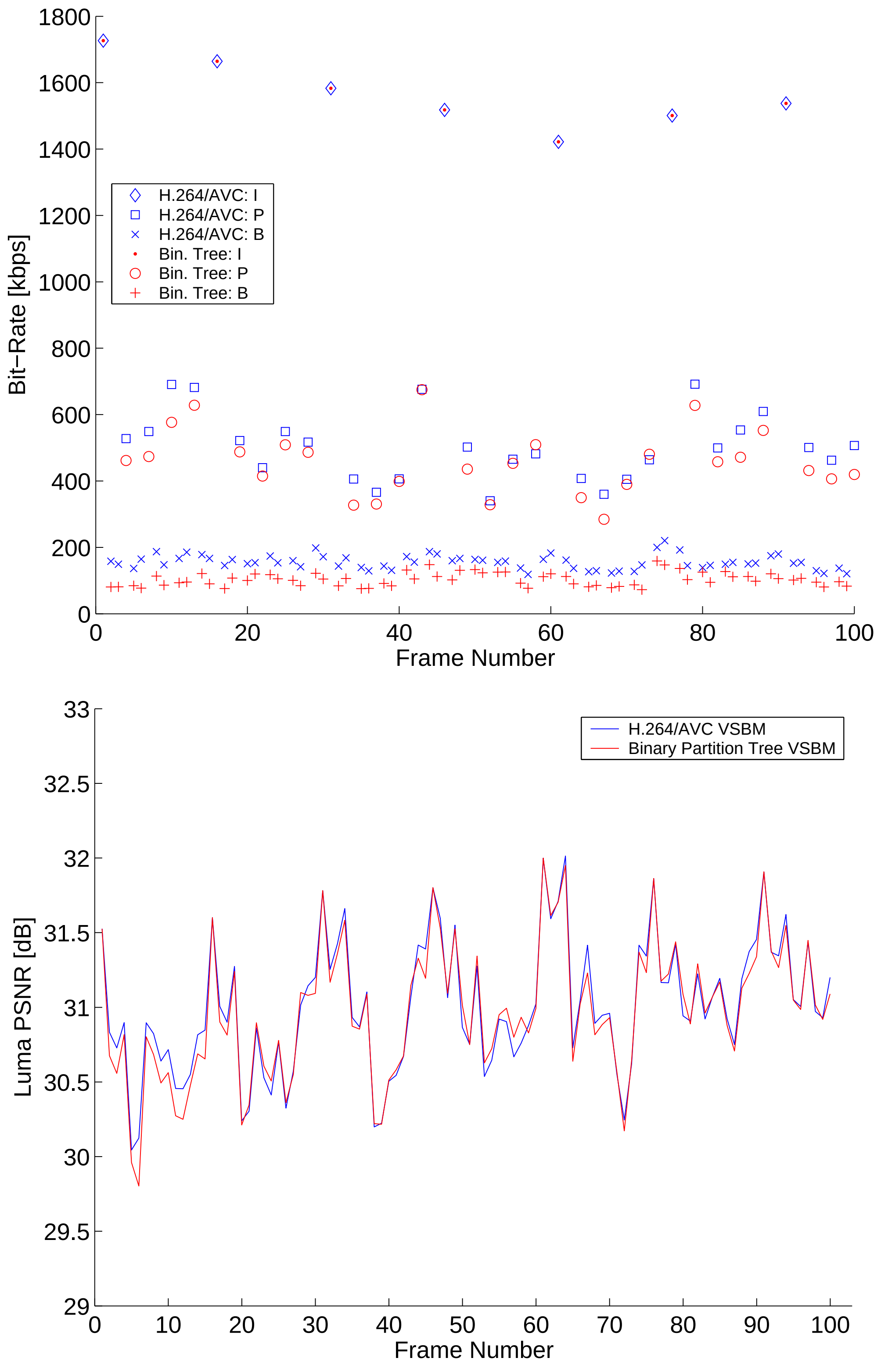
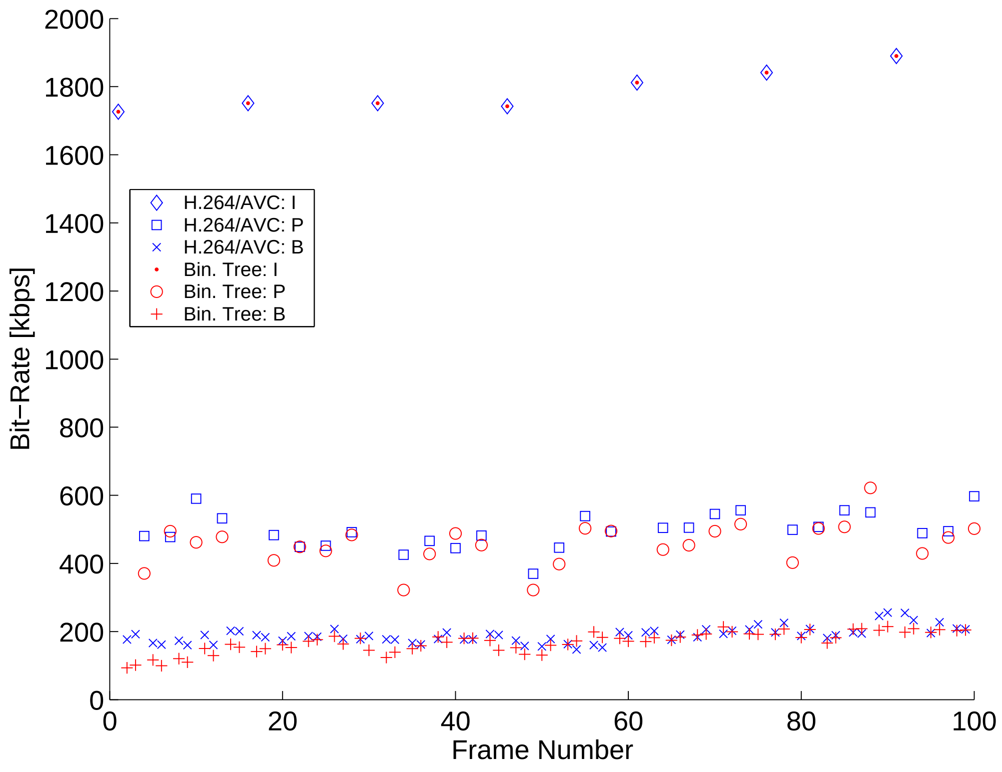


Figure 7.17: Frame-by-frame results for *City* (QP = 34)



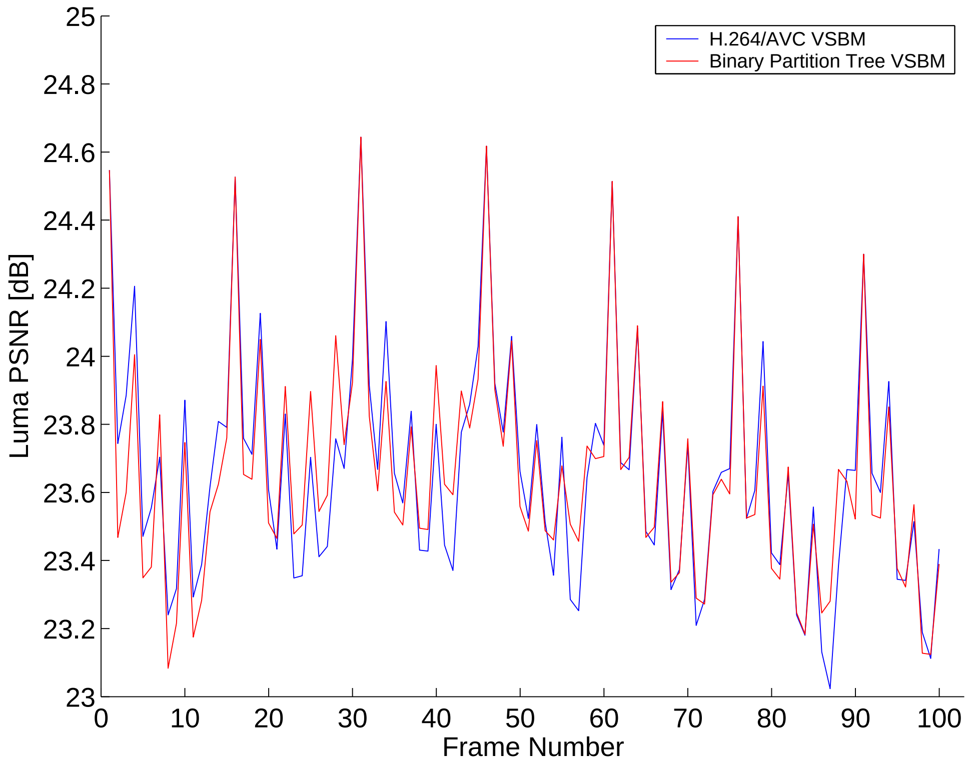
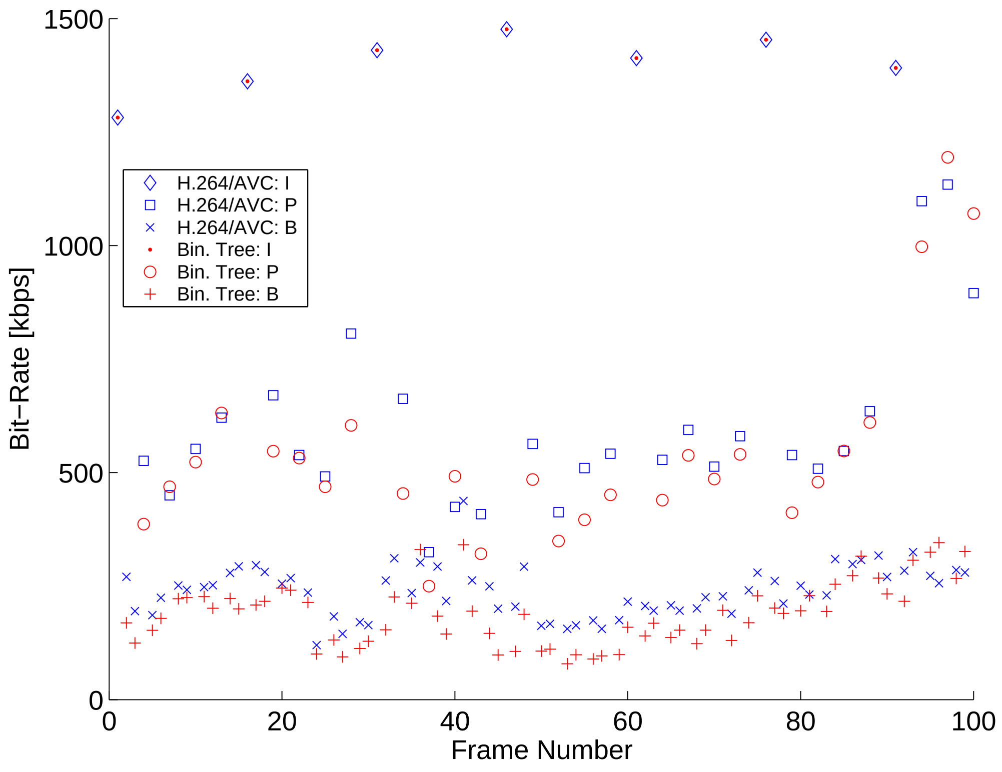


Figure 7.18: Frame-by-frame results for *Mobile & Calendar* (QP = 41)



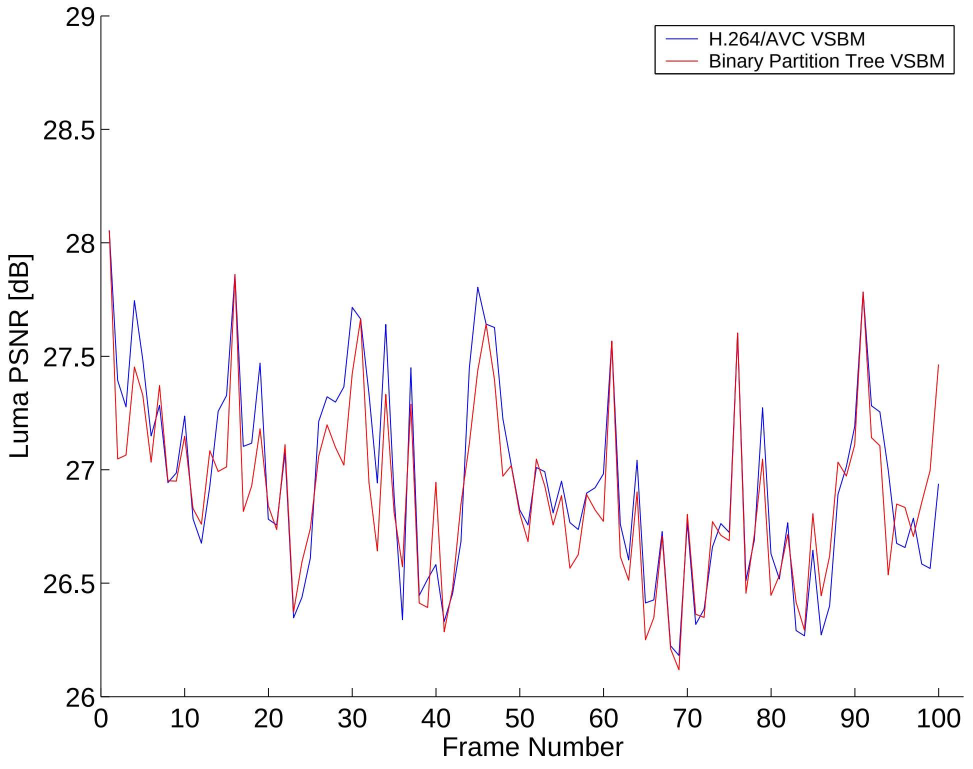


Figure 7.19: Frame-by-frame results for *Stefan* (QP = 39)

Table 7.3: The reduction in average bit-rate when using Binary Partition Tree VSBM, relative to H.264/AVC VSBM. Sequences compared at the same quality.

Sequence	**P frames**	**B frames**
Foreman ($QP = 34$)	9%	40%
City ($QP = 34$)	9%	34%
Mobile & Calendar ($QP = 41$)	8%	10%
Stefan ($QP = 39$)	9%	20%

coding a B frame using H.264/AVC VSBM is 137 kbps, while a Binary Partition Tree approach requires just 83 kbps.[21] This amounts to a 40% reduction in the B picture bit-rate. For the same example, the average reduction in bit-rate for P frames is 9%. The other three sequences also demonstrate savings in bit-rate for P and especially B frames. The results for inter frames are summarised in Table 7.3.

7.5.3 Bit-Stream Analysis

In order to further investigate the savings in bit-rate provided by Binary Partition Tree VSBM, it is instructive to compare the bit-stream characteristics when using the four different block-based motion compensation methods. Figures 7.20 to 7.23 describe the compressed data according to its three main components, which are:

- the information required to represent the block structure for each inter frame (Note that for the two FSBM methods, the block structure is implicit and thus does not need to be coded.);
- the motion information for each block (The includes a block's motion vector, choice of reference frame, and mode decision, i.e. the intra/inter

[21] Note that the mean PSNR for B frames using the two block matching methods is 33.29 dB for H.264/AVC VSBM and 33.22 dB for Binary Partition Tree VSBM (*Foreman*, $QP = 34$).

flag for each block. All of these values are coded predictively relative to neighbouring blocks.);

- an approximation of the residual after motion compensation. (Recall that the Matching Pursuit technique is used to represent and code the residual.)

The bar graphs presented in Figures 7.20 to 7.23 provide a break-down of the bit-stream into these three components (block structure, motion, and residual). They provide results for three different QP settings, for both P and B frames. Each bar graph also shows the average number of blocks per frame generated when performing motion compensation. (Clearly, the number of blocks only varies in the case of VSBM.) The characteristics of each of the four block-matching methods are discussed below in more detail:

- 8×8 FSBM (shown in black) uses relatively small blocks, and thus requires many of them to span the frame. As a result, a large number of motion vectors need to be coded. Although this helps to achieve accurate prediction, it leaves few bits for coding the residual. As the results show, the majority of the bit-stream is generally allocated to the motion component. This trend is particularly noticeable in case of B pictures, when the residual component is often very small.

- 16×16 FSBM (displayed in green) uses only a quarter as many blocks. Consequently, the proportion of motion information in the bit-stream is significantly reduced. This leaves more bits to code the residual, and as a result, the coded residual generally dominates the bit-stream. (It is worth noting that the use of 16×16 blocks can lead to more significant motion compensation artifacts, thus creating a DFD that needs more bits to encode.)

- The H.264/AVC implementation of VSBM (shown in blue) provides something of a compromise between the above two methods. By partitioning

16×16 blocks only where it is advantageous to do so, it enables trading-off the bit-rate allocation between motion and residual information. As the results show, fewer blocks are generated for B frames than for P frames. This can be understood by considering that *bi-directional* prediction allows for more accurate motion compensation of a block.

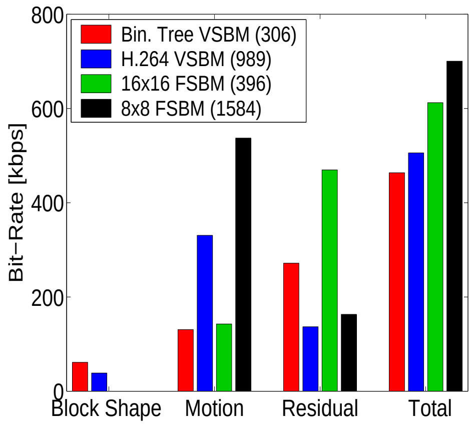
Clearly, there is a cost of coding the partition information for each macroblock. However, as can be seen, this a very small component of the bit-stream when compared to the motion and residual information. Even for the low bit-rate examples shown, the block structure information constitutes less than around 7% of the bit-stream.

- The Binary Partition Tree approach to VSBM (shown in red) performs optimal block partitioning, and then prunes the partition tree to retain those blocks that contribute the most towards minimising error. As can be seen from the results, this method generates significantly fewer blocks than the other three methods. As a result, the motion component of the bit-stream is noticeably reduced, particularly in the case of B frames.

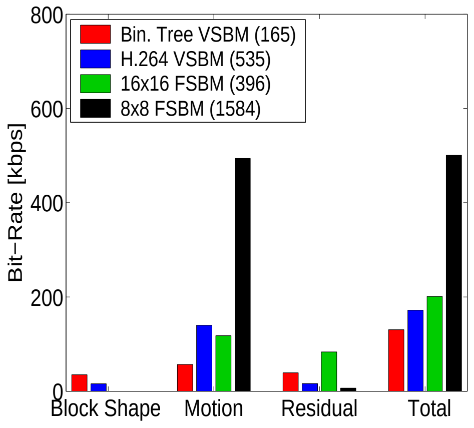
  The cost of representing the block structure using a Binary Partition Tree can be two or three times that associated with the H.264/AVC method. For B frames at very low bit-rates, this translates to around 20% of the bit-stream, which is quite substantial. However, as can be seen from the results, this extra cost is more than compensated for by the savings that arise when coding the motion information.

7.5.4 Subjective Performance

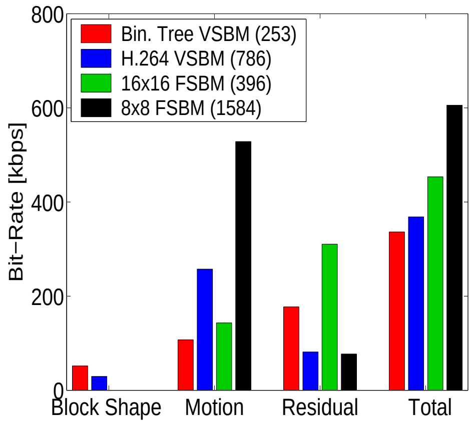
PSNR provides an objective measure of image fidelity. However, it does not always correlate well with the picture quality perceived by a human observer. This section therefore provides some sample frames in order to demonstrate the performance of the video codec described in this chapter. (Note that the results presented in this section only show the central portion of each frame.



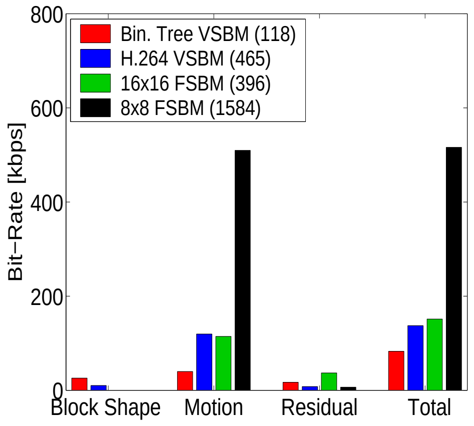
(a) P-frames for QP = 32



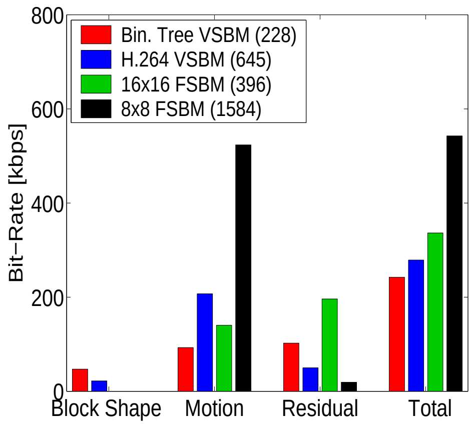
(b) B-frames for QP = 32



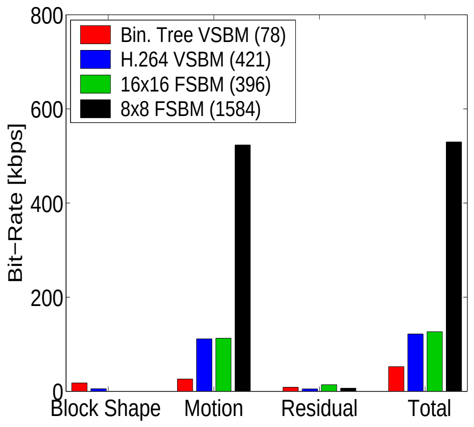
(c) P-frames for QP = 34



(d) B-frames for QP = 34

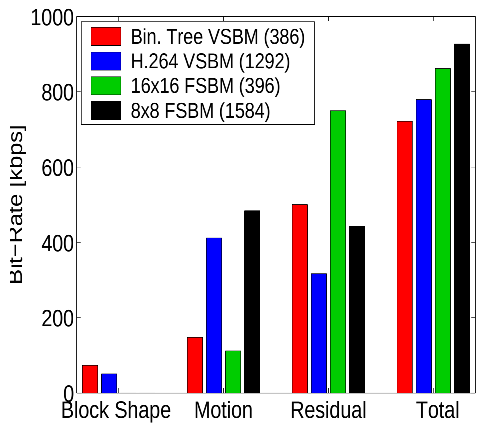


(e) P-frames for QP = 36

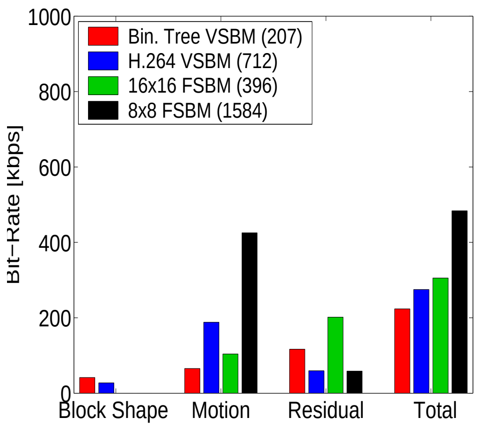


(f) B-frames for QP = 36

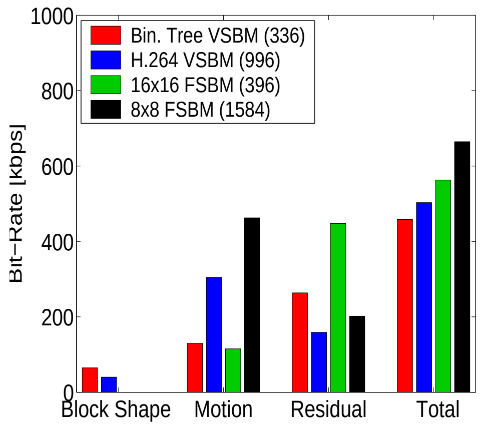
Figure 7.20: Analysis of the Bit-Stream components for *Foreman*



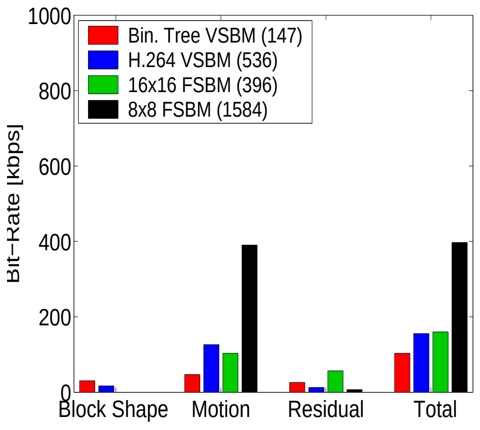
(a) P-frames for QP = 32



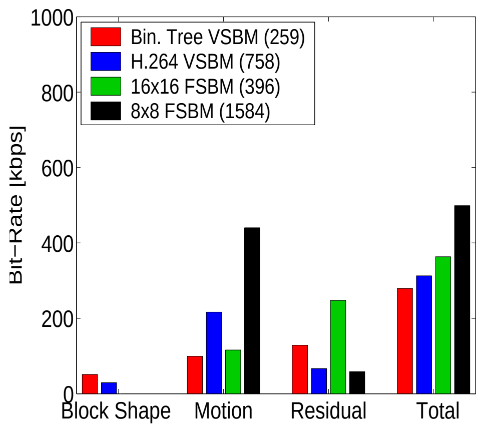
(b) B-frames for QP = 32



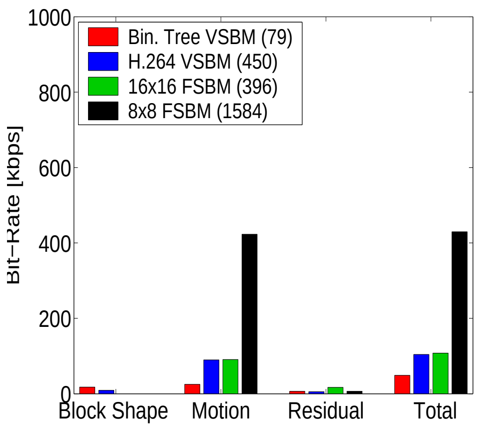
(c) P-frames for QP = 34



(d) B-frames for QP = 34

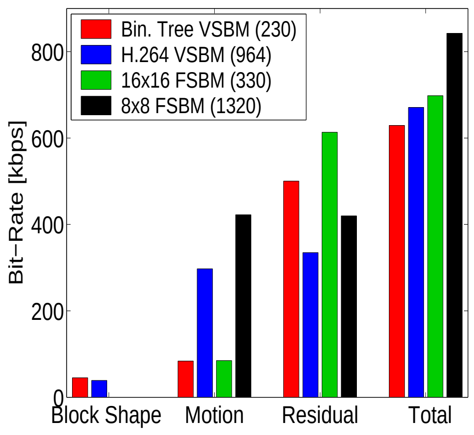


(e) P-frames for QP = 36

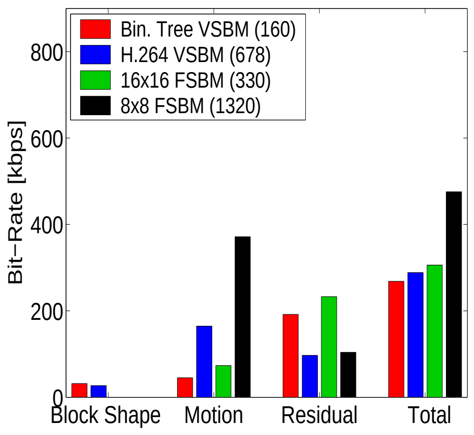


(f) B-frames for QP = 36

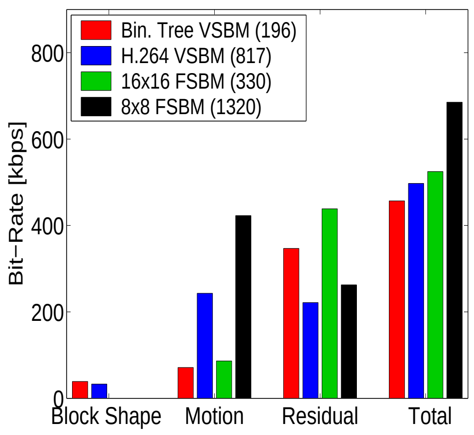
Figure 7.21: Analysis of the Bit-Stream components for *City*



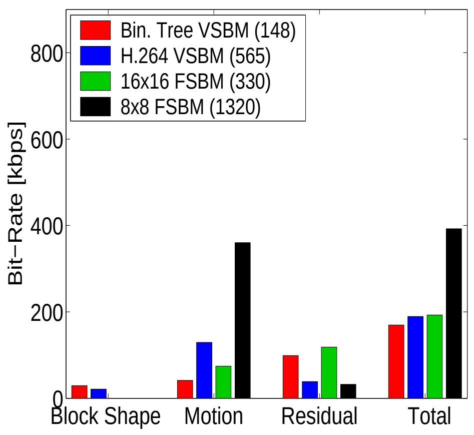
(a) P-frames for QP = 40



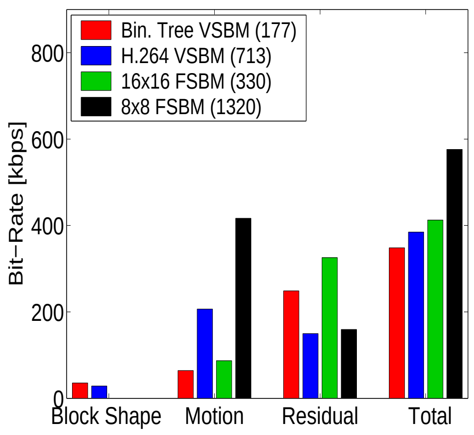
(b) B-frames for QP = 40



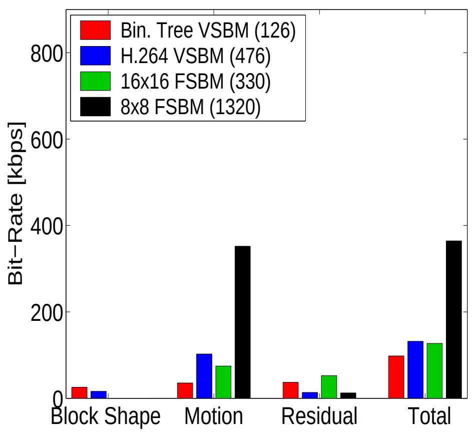
(c) P-frames for QP = 41



(d) B-frames for QP = 41

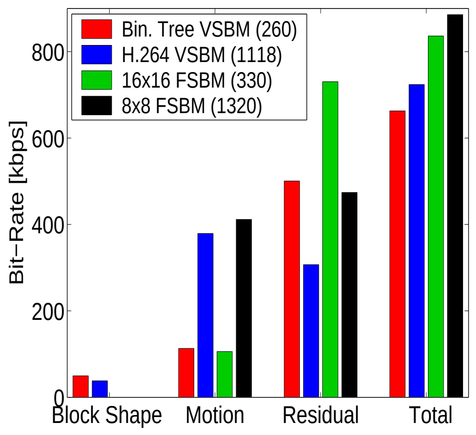


(e) P-frames for QP = 42

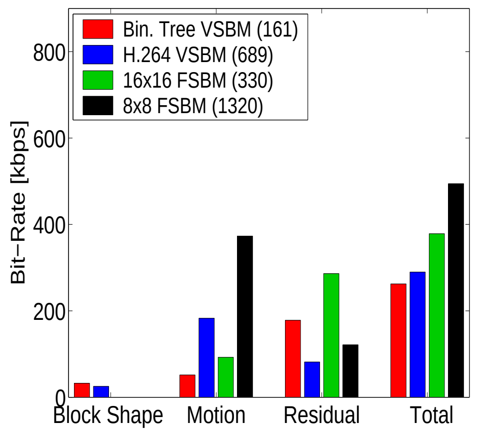


(f) B-frames for QP = 42

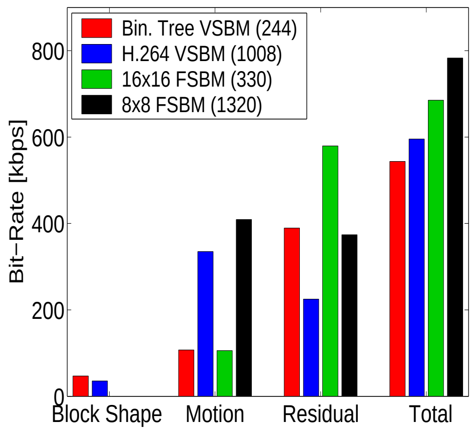
Figure 7.22: Analysis of the Bit-Stream components for *Mobile & Calendar*



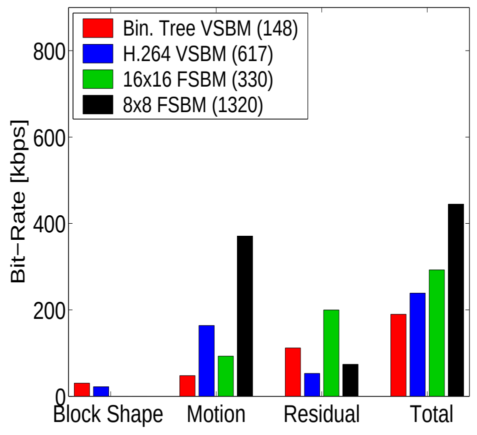
(a) P-frames for QP = 38



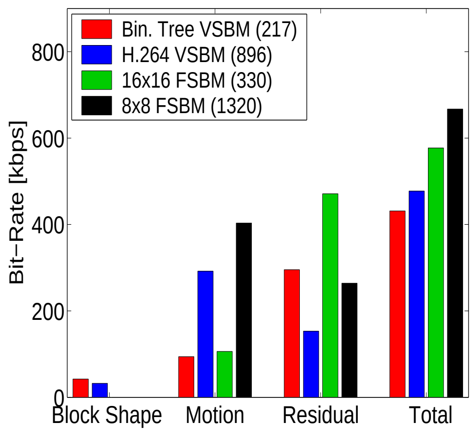
(b) B-frames for QP = 38



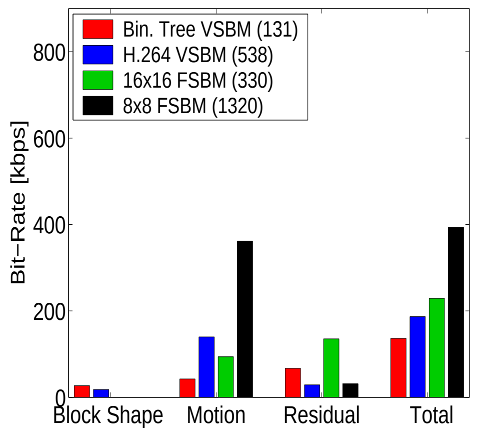
(c) P-frames for QP = 39



(d) B-frames for QP = 39



(e) P-frames for QP = 40



(f) B-frames for QP = 40

Figure 7.23: Analysis of the Bit-Stream components for *Stefan*

This allows for the picture detail to be observed more clearly.) For a complete set of results, the reader is referred to the accompanying DVD (Appendix D).

Comparison of Binary Partition Tree VSBM and H.264/AVC VSBM

Figure 7.24 shows the eighth (B) frame of each 15-frame GOP for the *Foreman* sequence. The first row of pictures are from the original sequence. The frames in the second row are from a version coded using H.264/AVC VSBM (with $QP = 36$), producing a sequence coded at 210 kbps. The third row shows frames coded using Binary Partition Tree VSBM (with $QP = 34$), yielding a bit-rate of 214 kbps. This allows for these two motion compensation methods to be compared at (almost) the same bit-rate.

The fourth row depicts the difference between the first and second rows (i.e. the motion compensation error associated with H.264/AVC VSBM), while the fifth row shows the difference between the first and third rows (i.e. the motion compensation error associated with Binary Partition Tree VSBM). Both compressed sequences exhibit significant coding artifacts:

- In both sequences, a significant lack of texture is noticeable. This is primarily due to the large quantisation parameter (QP) used for intra coding. However, this loss of detail is generally unavoidable at such a low bit-rate. A high value of QP also causes the colours to appear somewhat "washed out."

- Some minor block edge artifacts are noticeable. For example, frame 83 in row 2 (on the right[22] cheek) and frame 68 in row 3 (on the chin). This type of distortion arises because blocks are motion compensated independently of one another, which can introduce discontinuities in the motion vector field. The absence of major block boundary artifacts shows that OBMC has been largely successful.

[22] right from the *viewer's* perspective

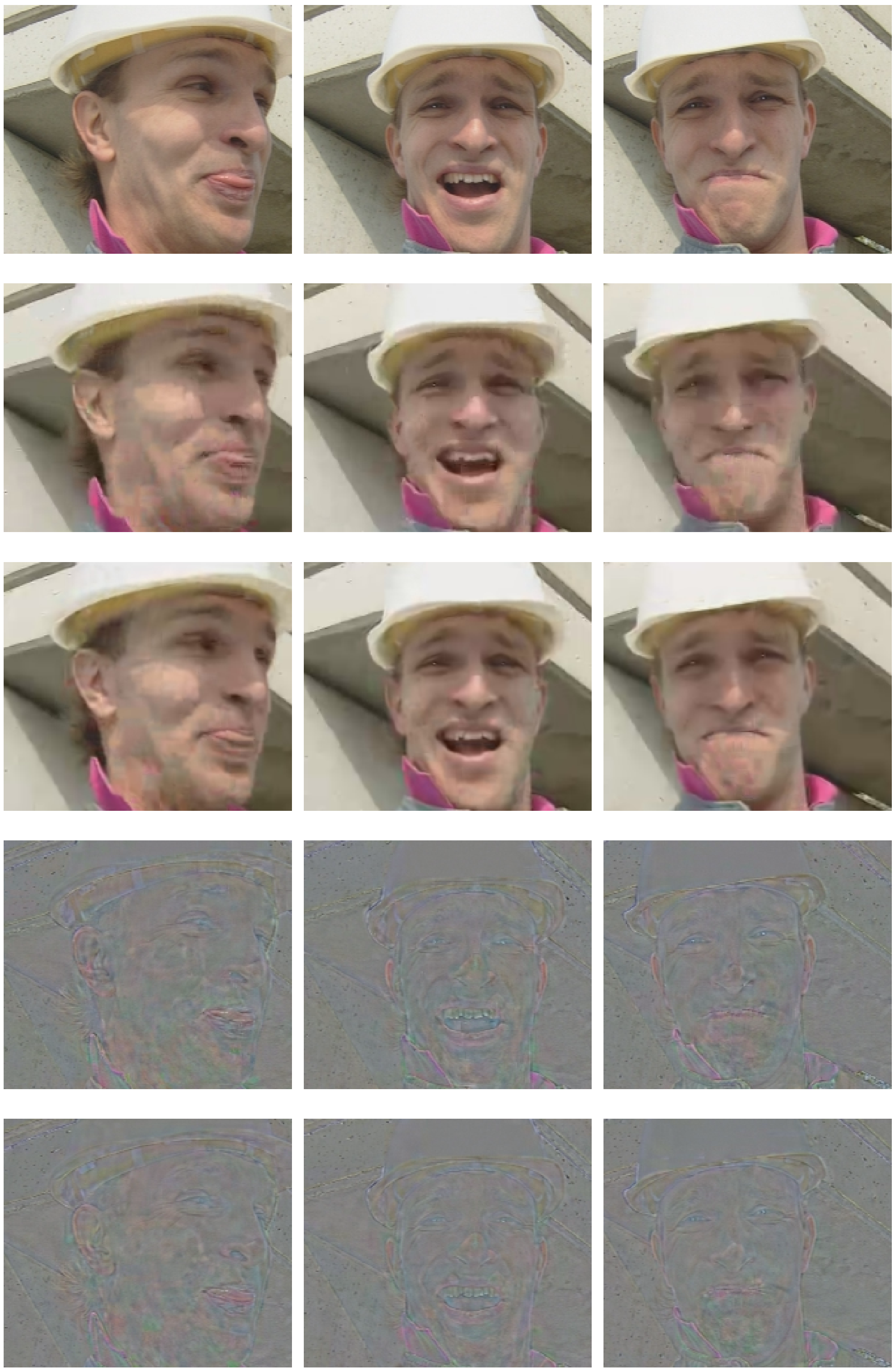
- There are some instances of chroma mis-matching. For example, frame 68 in row 2 (on left side of the chin and on the bottom right of the neck) and frame 8 in row 3 (on the neck/cheek area). This type of noise occurs because matching is performed using the luma component only, and is generally more likely for smaller blocks. These errors suggest that the intra/inter block mode decision could possibly be improved, since it may have been better for these blocks to have been flagged as intra blocks. Alternatively, the encoder could perform motion estimation using the chroma components as well.

Overall, the sequence in the second row (using H.264/AVC VSBM) shows more noticeable distortion than the one in the third row (using Binary Partition Tree VSBM). This can perhaps be most easily observed by examining the detail in the ears and mouth across the sequence. In addition, the difference images provide a guide as to where the most significant errors are located.

Comparison of Binary Partition Tree VSBM and 16 $\times$ 16 FSBM

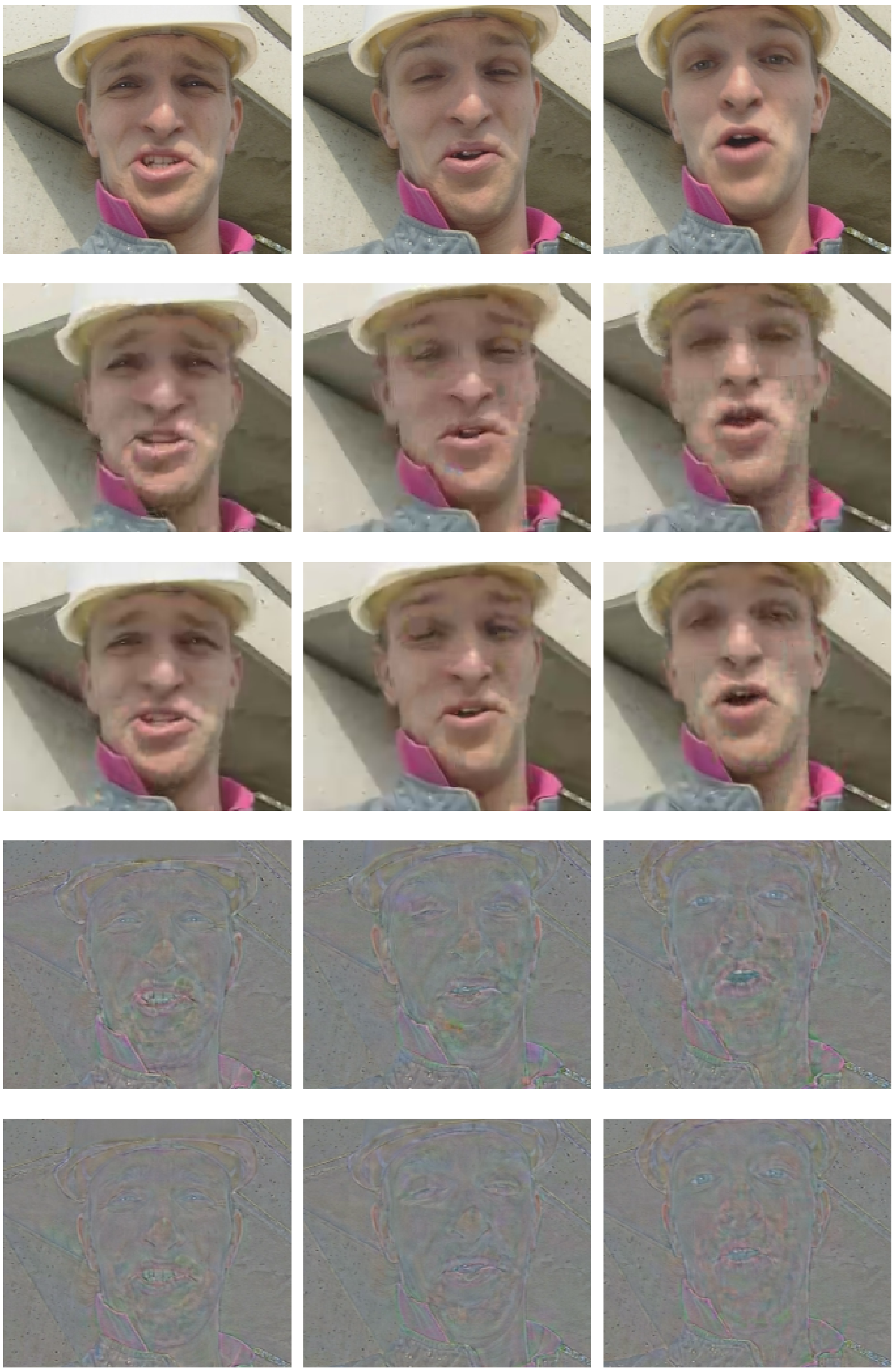
Figure 7.25 shows the seventh (P) frame of each 15-frame GOP for the *Stefan* sequence. The first row of pictures are from the original sequence. The frames in the second row are from a version coded using 16 $\times$ 16 FSBM (with $QP = 38$), producing an average bit-rate of 584 kbps. The third row shows frames coded using Binary Partition Tree VSBM (with $QP = 37$), yielding a bit-rate of 579 kbps. This allows for these two motion compensation methods to be compared at similar bit-rates. Rows four and five show the difference frames corresponding to the two respective motion compensation methods.

There is a significant amount of motion present in this sequence. The foreground motion is fast and involves a non-rigid object (i.e. the tennis player) that is not easy to motion compensate from one frame to the next. The background motion is due to the panning and zooming camera, and can be observed by noting the degree to which the advertising text moves. In addition, there is the motion of people within the highly-textured crowd.



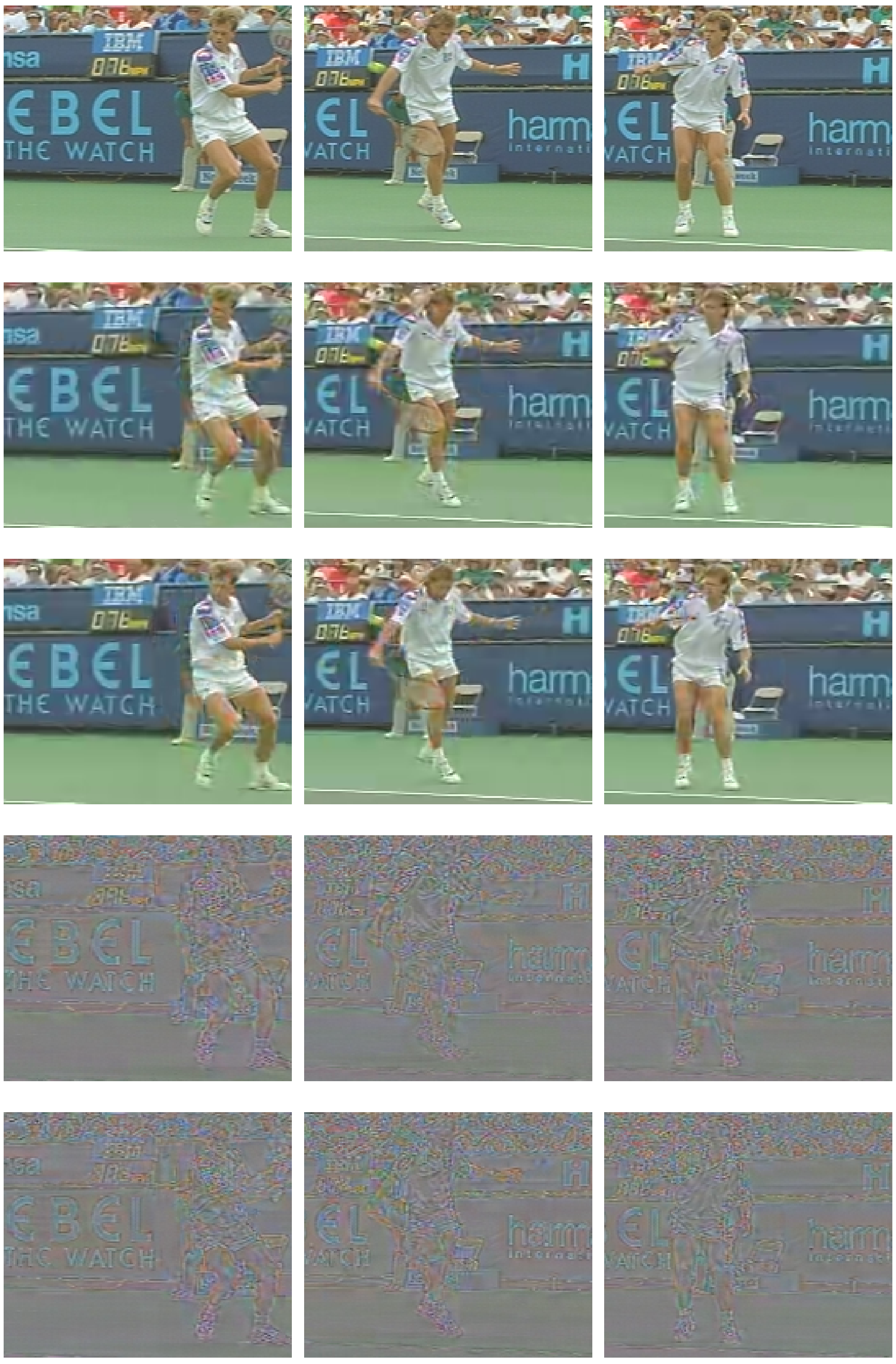
(a) Frame 8 (b) Frame 23 (c) Frame 38

Figure 7.24: Foreman frames (detail)



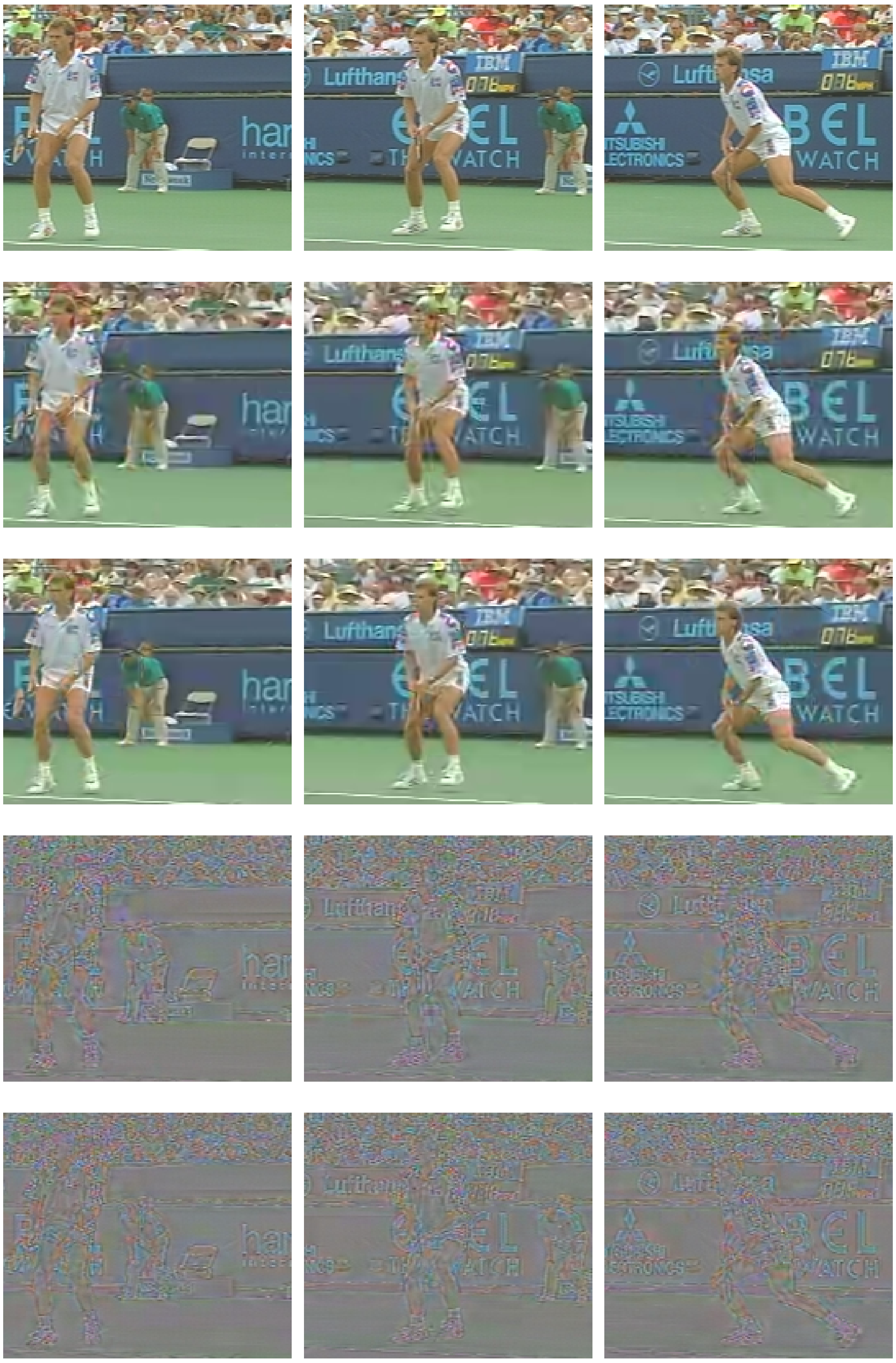
(d) Frame 53 (e) Frame 68 (f) Frame 83

Figure 7.24: Foreman frames (detail)



(a) Frame 7 (b) Frame 22 (c) Frame 37

Figure 7.25: Stefan frames (detail)



(d) Frame 52 (e) Frame 67 (f) Frame 82

Figure 7.25: Stefan frames (detail)

Both compressed sequences suffer from easily noticeable distortion. However, a close examination reveals that detail tends to be less well preserved when using FSBM (second and fourth rows) than Binary Partition Tree VSBM (third and fifth rows). For example, it is interesting to compare the tennis racquet in frame 7, the player's face in frames 37, 67 and 82, and the line judge in frame 52.

Some of the distortion is less noticeable when the sequence is played back at 30 fps. This is due to the varying sensitivity of the human visual system. In addition, spatial masking[23] causes some artifacts to be less noticeable than might be expected.

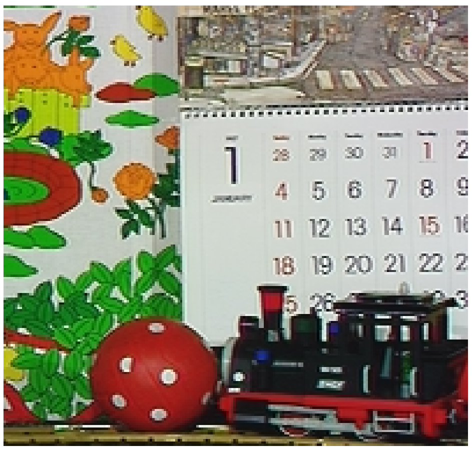
The Effect of Varying Quantisation

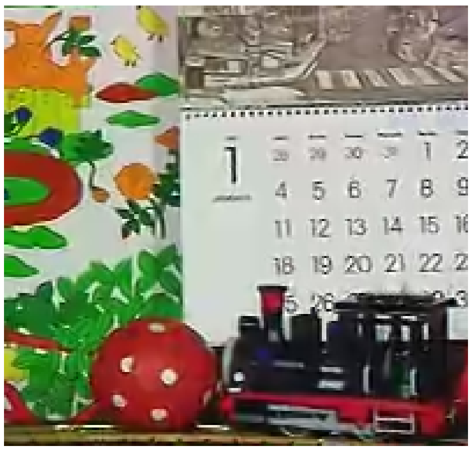
The quantisation parameter (QP) directly influences the quality of each intra frame. This is then used to control the quality of inter frames in the remainder of each GOP. Figures 7.26 and 7.27 demonstrate the effect of varying QP on the quality of a single frame in the *Mobile & Calendar* and *City* sequences, respectively.

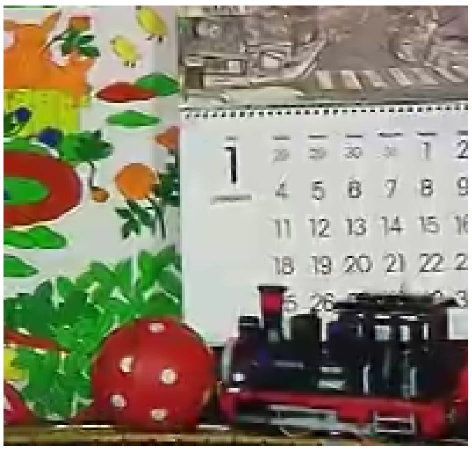
As QP increases, there is a gradual loss in texture, and edges become more and more blurred. The loss of fine detail is particularly evident in the calendar text, which is also subject to a loss of chroma information. The view of the city also suffers from a loss in detail that is most noticeable among the smaller background objects.

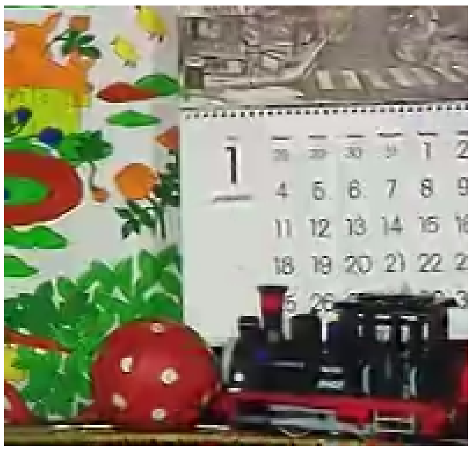
Another artifact in the *City* sequence is caused by a combination of compression and spatio-temporal aliasing. This occurs as a result of the motion of high-spatial-frequency components (such as those buildings with many small windows). This type of distortion is not visible when looking at individual frames, but only when viewing the sequence being played back in real-time. As a result, there appears to be some flicker among many of the building windows.

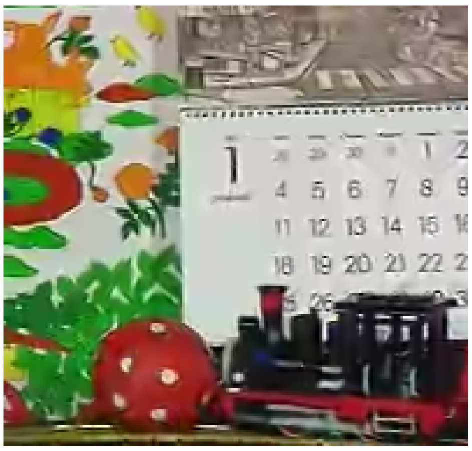
[23] Spatial masking describes the phenomenon that occurs when the visibility of a pattern appears to be influenced by its immediate neighbourhood. Thus Gaussian noise is generally less perceptible in textured areas or near edges, than in relatively flat regions of an image.


(a) original


(b) Binary Partition Tree (QP = 38)


(c) Binary Partition Tree (QP = 39)


(d) Binary Partition Tree (QP = 40)


(e) Binary Partition Tree (QP = 41)

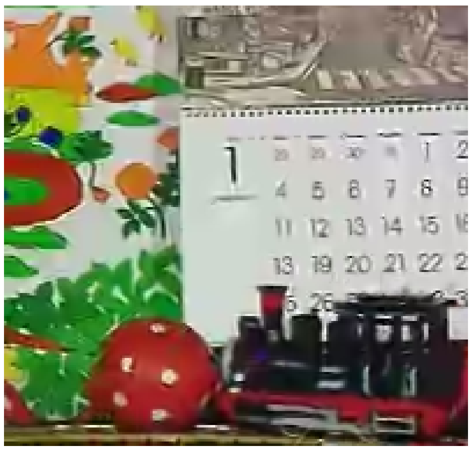
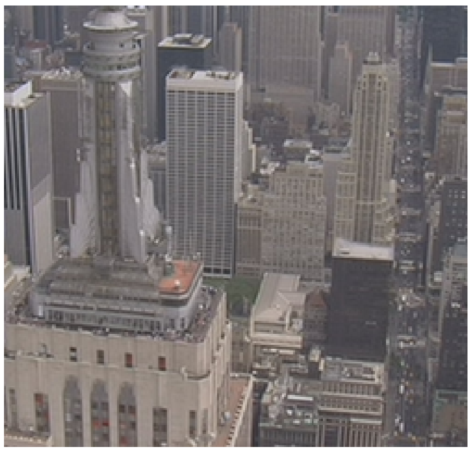
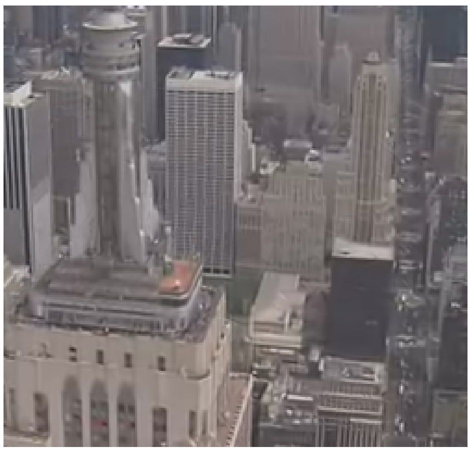
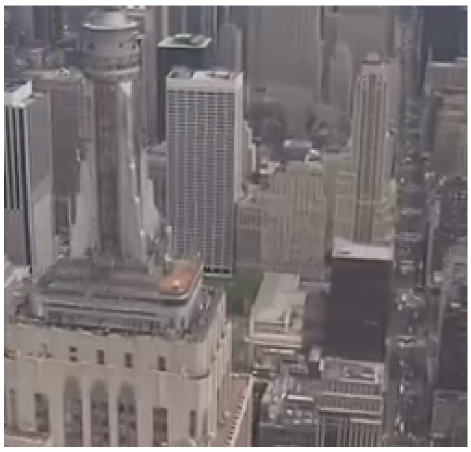
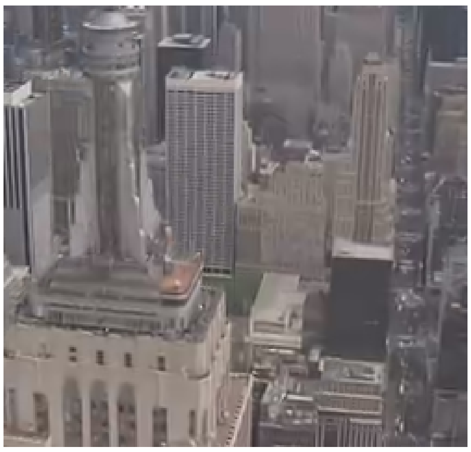

(f) Binary Partition Tree (QP = 42)

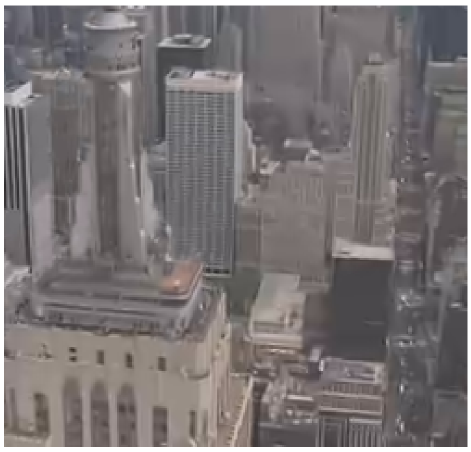
Figure 7.26: Frame 80 of *Mobile & Calendar* (detail)


(a) original


(b) Binary Partition Tree (QP = 30)


(c) Binary Partition Tree (QP = 31)


(d) Binary Partition Tree (QP = 32)


(e) Binary Partition Tree (QP = 34)

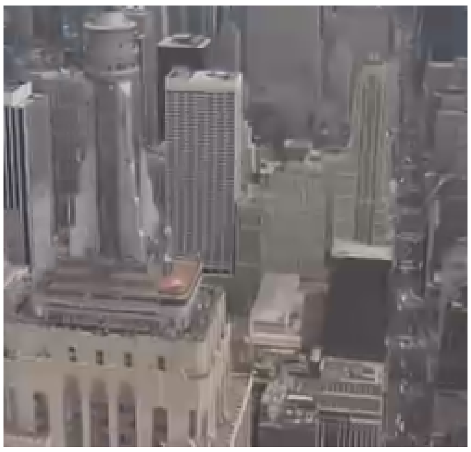

(f) Binary Partition Tree (QP = 36)

Figure 7.27: Frame 20 of *City* (detail)

Spatio-temporal aliasing is present to a limited degree in the original sequence, but it is more noticeable in the compressed versions. This is because the coding process introduces some distortion on the façades of these buildings that does not move in a consistent way from one frame to the next. These types of artifacts could potentially be reduced using spatio-temporal filtering and post-processing.

7.6 Conclusion

This chapter described the design and analysis of a hybrid video codec that allows the user to choose one of four block-based motion estimation and compensation methods. Based on a 15-frame IBBPBBP... structure, the codec uses H.264/AVC for intra coding and then performs either fixed-size or variable-size block matching (FSBM or VSBM) to code inter frames.

When using VSBM, the block structure needs to be coded. The structure is determined by the encoder using either the H.264/AVC approach to block splitting, or the Binary Partition Tree method described in Chapter 6. For each block, the intra or inter mode choice needs to be coded, after having been determined by the encoder.

Intra blocks are represented using the mean values of their luma and chroma components. For each inter block, its motion vector is coded predictively, along with the choice of reference frame (in the case of B pictures). Finally, the prediction error of each inter frame is approximated and coded using the Matching Pursuit technique.

Throughout the encoding process, Lagrangian rate-distortion optimisation is employed to ensure that each component of the encoder operates using the same rate-distortion trade-off relationship. This Lagrangian λ parameter is used to maintain a roughly constant picture quality across frames in a GOP. During the motion compensation stage, overlapping blocks are employed to help reduce blocking artifacts that are common to many block based codecs.

The proposed codec was tested on a number standard test sequences in order to compare the performance of the four block matching methods. In general, 16×16 blocks were found to provide superior results to 8×8 ones, particularly at low bit-rates. The H.264/AVC implementation of VSBM was in turn shown to out-perform both FSBM methods, by allowing macro-blocks to be split where it is advantageous to do so.

Further gains of up to 1 dB were observed when using Binary Partition Tree VSBM. The largest improvement was found to be at very low bit-rates, where this equates to a reduction in bit-rate of up to 24%. It was shown that the Binary Partition Tree method requires fewer blocks than the other three techniques. Consequently, fewer motion vectors need to be coded, resulting in a significantly lower bit-rate — particularly in the case of B frames.

Chapter 8

Conclusion

The research reported in this dissertation focused on three methods of content-based motion estimation and compensation. The most promising of these methods was then integrated into a hybrid video codec in order to allow for it to be compared to existing techniques. This chapter concludes this thesis by reviewing the preceding chapters, discussing the novel contributions of this work, and suggesting ideas for further research.

8.1 Summary

The need for the compression of digital video remains undiminished despite substantial increases in bandwidth and storage capabilities over the past few years. Standards have evolved significantly during the last decade, but the basic principles of video coding remain the same. Most codecs combine two fundamental techniques: transform coding (used to compress both original and residual frames) and motion-based prediction. The research described in this dissertation has focused on the latter — in particular the use of content-based methods of motion estimation and compensation.

When using an object-based approach, it is necessary to specify the shape and position of each object within a scene. From a compression point of view, this should be done as efficiently as possible, while still allowing a reasonable quality representation. A method was proposed whereby regions in a scene can be represented using a collection of polygons. Since a scene generally consists of several objects, it seems reasonable to use an approach in which the common boundaries can be represented efficiently, rather than having to code the shape of each object independently of its neighbours.

The first of the three content-based methods of motion compensation uses a triangular mesh. The mesh is designed by first performing spatial segmentation of the current frame, as well as dense motion estimation (relative to a reference frame). The motion field is then used to refine the segmentation by splitting regions with different types of motion, and merging areas with similar motion characteristics. The resulting segmentation map is approximated using the polygon-based approach described above, and a triangular mesh is then generated within each polygon. This method was shown to perform fairly well, however it proved difficult to control the number of triangles generated for each frame.

The second content-based motion compensation technique begins by using a similar spatial segmentation method. The optimal translational motion vector is then estimated for each region, and the resulting motion compensated error is noted. Following this, the area with the largest error is split, and the splitting process is repeated a number of times (each time for the region with the greatest error). Neighbouring regions with the same motion vector are then merged. The resulting set of regions are approximated with polygons, using the technique outlined above. Each (polygon-shaped) region is then populated with a grid of square blocks, with the number of blocks in a region being proportional to its motion compensation error. This results in small blocks being present in regions of complex motion, while allowing larger blocks to span the background areas. This approach has the advantage of enabling precise control over the number of blocks in each frame, which allows it to be more robust than the mesh-based method. However, its performance remains sensitive to the parameters used during the segmentation stage.

The third content-based motion compensation method attempts to combine the segmentation and block generation phases. It does this by starting off with one rectangular block that spans the entire frame. This block is then split into two (using a straight horizontal or vertical line) in a way that achieves the maximum reduction in motion compensation error. The process is applied repeatedly to the block with the largest motion compensation error, thus creating a binary tree of blocks. More blocks are generated than are required, but sibling blocks that provide the smallest gain by having been partitioned are re-merged into one block. A method for coding the binary partition information in an efficient way was also developed. This Binary Partition Tree VSBM technique was shown to provide a substantial improvement in quality over 16×16 FSBM performed at the same bit-rate.

It was decided to integrate the Binary Partition Tree method of motion estimation and compensation into a hybrid video codec. With a 15-frame IBBPBBP... structure, the codec uses the H.264/AVC method of intra coding. The user can select one of four block structures for motion estimation and compensation: 8×8 FSBM, 16×16 FSBM, H.264/AVC VSBM and Binary Partition Tree VSBM. Following motion compensation (using overlapping blocks), the residual is coded using a Matching Pursuit method. At each stage, the encoder uses Lagrangian optimisation in order to improve the rate-distortion performance, and a simple algorithm is employed to control the picture quality among each of the three (I, P and B) frame types.

Tests were performed on standard test sequences, using each of the four block-based motion compensation methods. Rate-distortion results (presented in Section 7.5) demonstrate the superior performance of Binary Partition Tree VSBM over the other techniques. In comparison to the state-of-the-art H.264/AVC VSBM method, Binary Partition Tree VSBM was found to reduce the total bit-rate by up to 24% (for the same picture quality). It was shown that the latter method allows for accurate motion compensation to be performed using relatively few blocks, particularly in the case of bi-directionally predicted (B) frames.

8.2 Contributions

The research described in this dissertation provides a number of novel contributions, which are listed below:

- As discussed in Section 3.2 shape coding is generally applied to individual objects, rather than a collection of regions in the form of a segmentation map. Vertex-based methods for coding segmentation maps do exist [48, 80, 73], however the progressive polygon approximation method described in Chapter 3 is believed to be novel, in that it allows for an embedded bit-stream to be produced. A coarser representation of a coded segmentation map can then be obtained by decoding only the initial portion of the bit-stream.

- The triangular mesh-based motion compensation method described in Chapter 4 uses a variety of existing techniques when performing segmentation, triangulation and motion estimation. Thus, although none of the components of the system are novel, it combines them in a new way into a chain of processes.

- Most implementations of VSBM tend to use either a binary-tree or quad-tree approach to generating variable-size blocks. The method of varying the block size by region, as described in Chapter 5, is believed to be novel. This approach operates by making the number of blocks in a region proportional to the motion compensation error for that region.

- Chapter 6 provides a detailed description of Binary Partition Tree VSBM. Much of this technique is believed to be original, in particular the method of partitioning blocks using the (horizontal/vertical) straight line that minimises motion compensation error. The way in which the partition tree structure is coded also appears to be novel.

The two region-based methods (discussed in Chapters 4 and 5) demonstrate some new ideas and show some definite promise, yet their performance was not considered to be consistent enough for integration into a video codec. The Binary Partition Tree VSBM method was found to provide some significant advantages over state-of-the-art H.264/AVC VSBM, while at the same time containing a number of original ideas.

8.3 Future Work

The progressive polygon approximation technique (described in Chapter 3) performs reasonably well as a stand-alone method for coding segmentation maps. However, when being used within a region-based video codec, it could be advantageous to also consider the content of the scene being segmented. Instead of using a fixed distance-based measure of distortion (d_{max}), it might be advantageous to allow large values of d_{max} along boundaries with similar spatial and temporal characteristics. Conversely, d_{max} could be reduced along boundary segments separating regions with very different spatial or motion characteristics. This would allow major edges to be approximated more precisely, while allowing coarser boundaries in other areas.

One of the major drawbacks of the mesh-based motion compensation method (described in Chapter 4) is that the number and shape of triangles within each (polygon-approximated) region is only dependent on the region's polygon shape. The mesh design process would need to be improved to make the number of triangles controllable, while still ensuring that they do not straddle motion boundaries.

In both of the region-based motion compensation approaches (described in Chapters 4 and 5), the segmentation process is performed using a fixed set of parameters. These values were found to work reasonably well on the material used in testing, however this represents only a small sample of video material. Ideally, it should be possible to tune the segmentation-related parameters in order to control the sensitivity of the process and thus the number of regions that are created. This would also allow performance to be improved through the use of rate-distortion optimisation.

A number of aspects of the Binary Partition Tree VSBM method (described in Chapter 6) could be improved. It may be advantageous to remove the restriction that blocks may only be split along their major dimension. In addition, the process of coding the partition tree could potentially be improved upon.

The main drawback of Binary Partition Tree VSBM is perhaps the computational cost involved with growing the tree. In particular, the process of finding the optimal partition for each block can be time consuming. This is especially true for large blocks, although one solution might be to use a frequency-domain method (such as phase correlation [49]) when performing motion estimation of larger blocks. In addition, a simple gradient-based search method might prove useful when searching for a block's optimal partition point.

All three of the proposed content-based motion compensation methods determine a new mesh-based or block-based structure for each frame, independently of the structures used from other frames. It is likely that some advantage could be achieved by using the motion compensation structures from previous frames. Thus, instead of generating a new mesh (or block structure) from scratch for each frame, it could evolve from one frame to the next. Such a strategy may allow for faster mesh/block generation and also more efficient coding.

The codec proposed in Chapter 7 lacks many of the features provided by H.264/AVC (e.g. multiple reference frames, more advanced rate and quality control, etc.). One possibility for further work would be to introduce some of these features into the codec. Another option worth further investigation is the possibility of incorporating Binary Partition Tree VSBM into a H.264/AVC framework.

Appendix A

The RGB to YC_bC_r Colour Transform

When transforming a pixel with red (R), green (G) and blue (B) values to luma and chroma components, two transforms are commonly used. One is for standard-definition pictures, while the other applies to high-definition images. As described in Section 2.2.3, the RGB to YC_bC_r transform is often useful for compressing images and video.

As an aside, it is worth noting the distinction between *luminance* and *luma*. Luminance is a measure of the brightness of a colour signal and is a weighted sum of linear RGB components. Luma refers to the Y component of the transformed colour signal, and is a weighted sum of (non-linear) gamma-corrected[1] RGB components [85]. Sometimes, luminance and luma are labelled as Y and Y' respectively. However, in this dissertation, Y is used to refer to the luma component.

[1] Gamma-correction is a non-linear process applied to pixel values prior to coding and transmission, in order to correct for the non-linear nature of most displays.

Transform for Standard Definition Pictures (Rec. 601):

Assuming that gamma-corrected *RGB* data lies in the range 0 to 255, recommendation ITU-R BT.601 [29] defines the appropriate colour transform as:

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} +0.257 & +0.504 & +0.098 \\ -0.148 & -0.291 & +0.439 \\ +0.439 & -0.368 & -0.071 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix}$$

with an inverse transform:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} +1.164 & +1.596 & 0 \\ +1.164 & -0.813 & -0.391 \\ +1.164 & 0 & +2.018 \end{bmatrix} \begin{bmatrix} Y - 16 \\ C_r - 128 \\ C_b - 128 \end{bmatrix}$$

Note that the above pair of transforms applies to all of the test sequences referred to in this dissertation.

Transform for High Definition Pictures (Rec. 709):

Assuming that gamma-corrected *RGB* data lies in the range 0 to 255, recommendation ITU-R BT.709 [30] defines the appropriate colour transform as:

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} +0.183 & +0.614 & +0.062 \\ -0.101 & -0.338 & +0.439 \\ +0.439 & -0.399 & -0.040 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix}$$

with an inverse transform:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} +1.164 & +1.793 & 0 \\ +1.164 & -0.534 & -0.213 \\ +1.164 & 0 & +2.115 \end{bmatrix} \begin{bmatrix} Y - 16 \\ C_r - 128 \\ C_b - 128 \end{bmatrix}$$

Appendix B

The use of Binarisation for Modelling and Coding

Much of the initial work on arithmetic coding involved its application to coding symbols from a binary alphabet, such as bi-level images [50]. In this case arithmetic coding offers potentially substantial advantages over Huffman coding [27], since the latter method produces at least one bit per symbol, while arithmetic coding can achieve a fractional number of bits per symbol. This allows for a good degree of compression if symbol probabilities are skewed (i.e. not close to 50/50).

Restricting the source alphabet to a binary one allows for the encoder and decoder to be simplified and optimised for speed. However, the overall throughput (measured in terms of information per CPU clock cycle) is not necessarily high, since each symbol in a binary alphabet only contains an average of one bit of information [65, 93].

There have been several applications of using binary arithmetic coders to code multi-symbol alphabets. This can easily be done by representing an alphabet of N symbols with a binary tree consisting of N leaves. Note that if arithmetic coding is used, there is no compression loss that results from representing data in a binary fashion (as opposed to using a multi-symbol format) [93, 61].

Recently, binarisation has come to be used as a relatively simple and effective way of combining context modelling and arithmetic coding. Context-based Adaptive Binary Arithmetic Coding (CABAC) [59, 57, 58, 61] consists of three stages: binarisation, context modelling and binary arithmetic coding. The binarised values are arranged into bins, and different contexts are maintained for each bin (or various groups of bins). This allows for accurate modelling of probabilities and a good way of handling the zero-frequency problem. As a result of its good performance, CABAC has been incorporated into the H.264 video coding standard [61].

CABAC supports four binarisation methods based on the following code trees: *unary*, *truncated unary*, *kth order Exp-Golomb*, and *fixed length* codes. These are described in more detail in [61], but an example of unary binarisation is provided below in order to illustrate its operation:

An Example of Unary Binarisation and Context Modelling

For this example, assume that (block) motion vector differences are to be encoded, and that these have a magnitude ranging from 0 to 32. To simplify matters, only consider the magnitude of the horizontal component of the motion vector differences, $|\text{MVD}_x|$. Each value is then binarised using a unary code. This is done by transforming a value n to its binary representation of n zeros followed by a one, as illustrated in Table B.1.

Binarisation allows for context modelling to be applied to each *bin* (bit position). The choice of how many contexts to use is left up to the codec designer, but the first few bins are usually assigned their own individual contexts. In fact, the first bin is often supported with multiple contexts, depending on the values of previously coded neighbours.

In the current example, there are three possible contexts (A, B and C) for the first bin of $|\text{MVD}_x|$. The context that is chosen depends on the values of $|\text{MVD}_x|$ in those blocks neighbouring the current block. With reference to

Table B.1: Binarisation using a unary code

Value of $\lvert\mathrm{MVD}_x\rvert$	Binary representation (Unary code)														
0	1														
1	0	1													
2	0	0	1												
3	0	0	0	1											
4	0	0	0	0	1										
5	0	0	0	0	0	1									
$\vdots$															
30	0	0	0	0	0	0	0	0	0	0	0	…	1		
31	0	0	0	0	0	0	0	0	0	0	0	…	0	1	
32	0	0	0	0	0	0	0	0	0	0	0	…	0	0	1
Bin number:	1	2	3	4	5	6	7	8	9	10	11	…	31	32	33

Figure B.1, let the neighbourhood prediction error metric for the current block, e_x, be calculated as $e_x = |\mathrm{MVD}_x(b_p)| + |\mathrm{MVD}_x(b_q)| + |\mathrm{MVD}_x(b_r)| + |\mathrm{MVD}_x(b_s)|$.

When e_x is small, it is more likely that $|\mathrm{MVD}_x|$ will also be small. For this reason, multiple contexts are used for the first bin (which effectively indicates whether $|\mathrm{MVD}_x|$ is zero or not).

Each context model is used to represent the probability of a bit in that context being zero or one. As shown in Table B.2, three context models are used for the first bin, one context model each for bins two to four, and two contexts for the high-order bins. (Note that the eight context models used here are not optimal but are merely used as an example. However, most implementations of

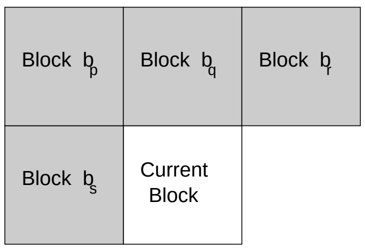


Figure B.1: The current block and its previously encoded (decoded) neighbours

Table B.2: Context Models used for each of the Binarisation Bins

Bin number	Context Model
1	Model A (if $e_x < 3$) Model B (if $3 \leq e_x < 20$) Model C (if $e_x \geq 20$)
2	Model D
3	Model E
4	Model F
5 to 10	Model G
11 to 32	Model H

CABAC follow a roughly similar approach.)

With the introduction of CABAC, it has been shown that binarisation can combine well with adaptive arithmetic coding when relatively few context models are used. Unary code binarisation is particularly suited to modelling Laplacian probability density functions, and these distributions are very common in image and video compression systems. In summary, the main advantages of CABAC are:

- Context modelling can be performed at a sub-symbol level, due to values being decomposed into a binary representation.
- Relatively few context models are necessary, which alleviates the problem of context dilution. (Context dilution arises when there are a large number of contexts with insufficient data available to suggest accurate models for all contexts.)
- Similarly, the zero-frequency problem is alleviated because binary alphabets are unlikely to contain novel symbols. Furthermore, if there are relatively few contexts, each context model can adapt fairly quickly to changing statistics.

Appendix C

The 1D Wavelet Transform

Consider a one-dimensional signal $f(t)$ which varies with time.[1] It can be represented as a sum of *basis functions*:

$$f(t) = \sum_i c_i \Psi_i(t)$$

where $\{\Psi_i(t)\}$ are the basis functions and $\{c_i\}$ are the coefficients with which each basis function is weighted. In Fourier Analysis the basis functions used are sines and cosines, in which case the c_i correspond to the Fourier coefficients of the signal.

It is interesting to note that $f(t)$ directly conveys information about the signal's behaviour in time, but not in frequency. On the other hand, the frequency domain representation describes the function in terms of its constituent frequencies, but gives no indication of the time domain behaviour of the function.

In signal compression it is often desirable to have information about the frequency of a signal over a specific region or time interval [24]. This is particularly relevant for many *natural* signals with regions of high correlation such as images or video sequences. Wavelets are basis function which can be used to achieve

[1] Alternatively, one could consider a signal which varies in space, $f(x)$ say.

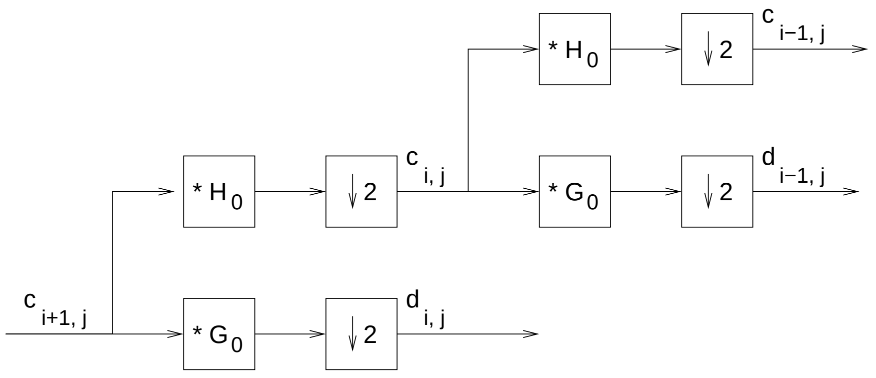


Figure C.1: The Discrete Wavelet Transform (DWT) implemented with convolution and down-sampling. Down-sampling is achieved by discarding every alternate value. (Notation: i indicates the sub-band; j is a coefficient index.)

such a function decomposition, since they are *localised* in both time (or space) and frequency.

Multi-resolution Analysis was developed by Mallat [55] for the construction of dyadic wavelets. Various methods exist for implementing the wavelet transform. The two most common are perhaps the *lifting scheme* [22] and the *filter bank* implementation [110], which is described below.

A wavelet decomposition may be achieved by filtering with appropriately designed high pass and low pass filters. Figure C.1 illustrates how the Discrete Wavelet Transform (DWT) is performed. First the signal is convolved with the low pass (H_0) and high pass (G_0) filters, and then down-sampled by a factor of two. This results in sets of *approximation* coefficients ($c_{i,j}$) and *detail* coefficients ($d_{i,j}$) respectively.

The approximation coefficients are a representation of the original signal at a coarser resolution. The detail coefficients represent the information lost by approximating the signal at the coarser sub-band. The process is applied recursively to the approximation coefficients in each sub-band, for the desired number of decomposition levels.

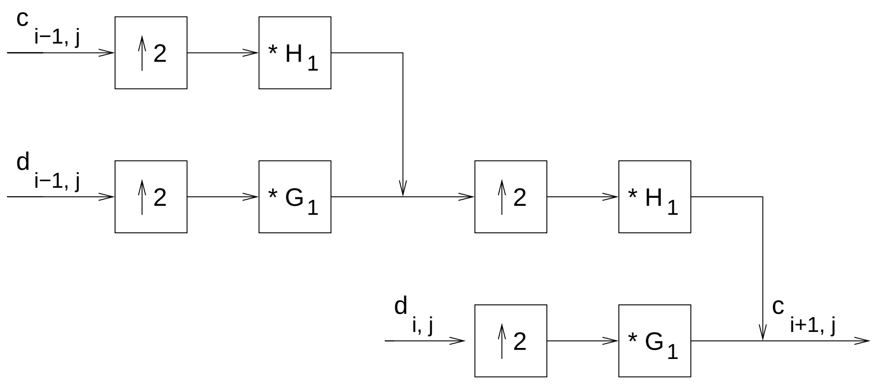


Figure C.2: The Inverse Wavelet Transform implemented with up-sampling and convolution. (Up-sampling is achieved by inserting zeros between each pair of coefficients.)

The inverse wavelet transform is simply the reverse of the procedure outlined above. Figure C.2 illustrates the process. Note that in both the forward and inverse transforms, either periodic or symmetric extension of the signal is usually performed prior to convolution, with the latter being more appropriate for compression purposes.

Bibliography

[1] Y. Altunbasak and A.M. Tekalp. Occlusion-Adaptive, Content-Based Mesh Design and Forward Tracking. *IEEE Trans. Image Proc.*, 6(9), September 1997.

[2] R. Arnold and T.C. Bell. A Corpus for the Evaluation of Lossless Compression Algorithms. In *Designs, Codes and Cryptography*, pages 201–210, 1997. `http://corpus.canterbury.ac.nz`.

[3] M.J. Black and P. Anandan. Robust Dense Optical Flow: Source Code, 1996. `http://www.cs.brown.edu/people/black/ignc.html`.

[4] M.J. Black and P. Anandan. The robust estimation of multiple motions: parametric and piecewise-smooth flow fields. *Computer Vision and Image Understanding*, 63(1), 1996.

[5] D.B. Bradshaw. *Motion Estimation and Compensation of Video Sequences using Affine Transforms.* PhD thesis, Cambridge University, December 1998.

[6] D.B. Bradshaw and N.G. Kingsbury. A Combined Affine and Translational Motion Compensation Scheme using Triangular Tessellations. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 2, pages 2645–2648, April 1997.

[7] CCITT. Video Codec for Audiovisual services at $p \times 64$ kbit/s, CCITT Recommendation H.261, 1990.

[8] I. Celasun and A.M. Tekalp. Optimal 2-D Hierarchical Content-Based Mesh Design and Update for Object-Based Video. *IEEE Transactions on Circuits and Systems for Video Technology*, 10(7):1135–1153, October 2000.

[9] M. Chan, Y. Yu, and A. Constantinides. Variable Size Block Matching Motion Compensation with Applications to Video Coding. *IEE Proceedings on Communication, Speech and Vision*, 137(4):205–212, August 1990.

[10] C-C. Chang, L-L. Chen, and T-S. Chen. An Improvement of Bottom-Up Variable-Sized Block Matching Technique for Video Compression. *IEEE Transactions on Consumer Electronics*, 44(4):1234–1242, November 1998.

[11] G. Cote, B. Erol, M. Gallant, and F. Kossentini. H.263+: Video Coding at Low Bit Rates. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(7):848–866, November 1998.

[12] I. Daubechies. *Ten Lectures on Wavelets*. CBMS-NSF Reg. Conf. Series in Applied Math. SIAM, 1992.

[13] J.G. Daugman. Uncertainty Relations for Resolution in Space, Spatial Frequency, and Orientation Optimized by Two-Dimensional Visual Cortical Filters. *Journal of the Optical Society of America*, 2(7):1160–1169, July 1985.

[14] Commission Internationale de L'Eclairge (CIE). Official Recommendations on Uniform Color Spaces, Color-Difference Equations, and Metric Color Terms, 1976. CIE Publication No. 15, Supplement Number 2 (E-1.3.1).

[15] B.N. Delaunay. Sur la Sphère Vide. *Izvestia Akademia Nauk SSSR, VII Seria, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7:793–800, 1934.

[16] Y. Deng and B.S. Manjunath. Unsupervised Segmentation of Color-

Texture Regions in Images and Video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(8):800–810, August 2001.

[17] Y. Deng and B.S. Manjunath. JSEG: Segmentation of Color-Texture Regions in Images and Video, 2004. `http://vision.ece.ucsb.edu/segmentation/jseg/`.

[18] M. Dudon, O. Avaro, and C. Roux. Triangular active mesh for motion estimation. *Signal Processing: Image Communication*, 10(1–3):21–41, July 1997.

[19] F. Dufaux and F. Moscheni. Motion Estimation Techniques for Digital TV: A Review and a New Contribution. *Proceedings of the IEEE*, 83(6):858–876, June 1995.

[20] M. Eden and M. Kocher. On the Performance of a Contour Coding Algorithm in the Context of Image Coding. Part I: Contour Segment Coding. *Signal Processing*, 8:381–386, July 1985.

[21] P.E. Eren and A.M. Tekalp. Bi-Directional 2-D Mesh Representation for Video Object Rendering, Editing and Superresolution in the Presence of Occlusion. *Signal Processing: Image Communication*, 18(5):313–336, May 2003.

[22] G. Fernández, S. Periaswamy, and W. Sweldens. LIFTPACK: A software package for wavelet transforms using lifting. In M. Unser, A. Aldroubi, and A. F. Laine, editors, *Wavelet Applications in Signal and Image Processing IV*, pages 396–408. Proc. SPIE 2825, 1996.

[23] H. Freeman. On the Encoding of Arbitrary Geometric Configurations. *IRE Transactions on Electronic Computers*, EC-10(2):260–268, June 1961.

[24] M. L. Hilton, B.D. Jawerth, and A. Sengupta. Compressing Still and Moving Images with Wavelets. *Multimedia Systems*, 2(5):218–227, 1994.

[25] B.K.P. Horn and B.G. Schunck. Determining Optical Flow. *Artificial Intelligence*, 17:185–203, August 1981.

[26] M. Hu, S. Worrall, A.H. Sadka, and A.M. Kondoz. A Scalable Vertex-based Shape Intra-coding Scheme for Video Objects. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 273–276, May 2004.

[27] D.A. Huffman. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, 40(9):1098–1101, September 1952.

[28] R.W.G. Hunt. *The Reproduction of Colour in Photography, Printing and Television.* Fountain Press, 1987.

[29] International Telecommunications Union. Encoding Parameters of Digital Television for Studios, 1992. Recommendation ITU-R BT.709-3.

[30] International Telecommunications Union. Parameter Values for the HDTV Standards for Production and International Programme Exchange, February 1998. Recommendation ITU-R BT.709-3.

[31] ISO/IEC. MPEG-1: Information Technology - Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbit/s - Video, Geneva, 1993.

[32] ISO/IEC JTC1 and ITU-T. MPEG-2/H.262: Generic Coding of Moving Pictures and Associated Audio Information - Part 2: Video, November 1994.

[33] ISO/IEC JTC1/SC24 and World Wide Web Consortium (W3C). Portable Network Graphics (PNG): Functional Specification (Second Edition), Document ISO/IEC 15948:2003(E), November 2003. `http://www.w3.org/TR/PNG/`.

[34] ISO/IEC JTC1/SC29/WG1. JPEG 2000 Image Coding System–ISO/IEC 15444-1:2000, December 2000. `http://www.jpeg.org/public/fcd15444-2.pdf`.

[35] ISO/IEC JTC1/SC29/WG11. MPEG-4 Video Verification Model: Version 18.0, WG11 Document N3908, Pisa, January 2001. `http://www.chiariglione.org/mpeg/working_documents.htm#MPEG-4`.

[36] ISO/IEC JTC1/SC2/WG10. Information Technology – Coded Representation of Picture and Audio Information – Digital Compression and Coding of Continuous-Tone Still Images (JPEG standard), 1993.

[37] ITU-T. Video Coding for Low Bit-rate Communication, ITU-T Recommendation H.263, March 1996.

[38] ITU-T. Video Coding for Low Bit-rate Communication, ITU-T Recommendation H.263, version 2 (H.263+), January 1998.

[39] ITU-T and ISO/IEC JTC1. Advanced Video Coding for Generic Audiovisual Services, ITU-T Recommendation H.264 ISO/IEC 14496-10 AVC, 2003.

[40] E. Izquierdo and M. Ghanbari. Key Components for an Advanced Segmentation System. *IEEE Transactions on Multimedia*, 3(1):97–113, March 2002.

[41] J.R. Jain and A.K. Jain. Displacement Measurement and its Application to Interframe Image Coding. *IEEE Trans. on Comm.*, COM-29(12):1799, December 1981.

[42] P. Jost, P. Vandergheynst, and P. Frossard. Tree-Based Pursuit: Algorithm and Properties. Technical Report 2005-13, EPFL, May 2005.

[43] K. Suehring. H.264/AVC Reference Software, 2005. `http://iphome.hhi.de/suehring/tml/`.

[44] A.K. Katsaggelos, L.P. Kondi, F.W. Meier, J. Ostermann, and G.M. Schuster. MPEG-4 and Rate-Distortion-Based Shape-Coding Techniques. *Proceedings of the IEEE, special issue on Multimedia Signal Processing*, 86(6):1126–1154, June 1998.

[45] J.W. Kim and S.U. Lee. Hierarchical Variable Block Size Motion Estimation for Motion Sequence Coding. *Optical Engineering*, 33(8):2553–2561, August 1994.

[46] J.W. Kim and S.U. Lee. Video Coding with R-D Constrained Hierarchical Variable Block Size (VBS) Motion Estimation. *Journal of Visual Communication and Image Representation*, 9(3):243–254, 1998.

[47] L.P. Kondi, G. Melnikov, and A.K. Katsaggelos. Joint Optimal Object Shape Estimation and Encoding. *IEEE Trans. Circuits Syst. Video Tech.*, 14(4), April 2004.

[48] J. Konrad, A-R. Mansouri, E. Dubois, V-N. Dang, and J-B. Chartier. On Motion Modelling and Estimation for Very Low Bit Rate Video Coding. In *Proc. VCIP*, pages 262–273, 1995.

[49] C. Kuglin and D. Hines. The Phase Correlation Image Alignment Method. In *Proceedings of the IEEE International Conference on Cybernetics and Society*, pages 163–165, 1975.

[50] G.G. Langdon and J.J. Rissanen. Compression of Black-White Images with Arithmetic Coding. *IEEE Transactions on Communications*, COM-29(6):858–867, June 1981.

[51] C. Le Buhan Jordan and T. Ebrahimi. Progressive Polygon Encoding of Shape Contours. In *Proc. IPA*, pages 17–21, 1997.

[52] C. Le Buhan Jordan, T. Ebrahimi, and M. Kunt. Progressive Content-Based Shape Compression for Retrieval of Binary Images. *Computer Vision and Image Understanding*, 71(2):198–212, August 1998.

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

[54] B.D. Lucas and T. Kanade. An Iterative Image Registration Technique

with an Application to Stereo Vision. In *Proceedings of the DARPA Image Understanding Workshop*, pages 121–130, April 1981.

[55] S. Mallat. Multiresolution approximations and wavelet orthornormal bases in $\mathbf{L}^2(R)$. *Transactions of the American Mathematics Society*, 315(1):69–87, September 1989.

[56] S. Mallat and Z. Zhang. Matching Pursuit with Time-Frequency Dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, December 1993.

[57] D. Marpe, G. Blattermann, G. Heising, and T. Wiegand. Further Results for CABAC Entropy Coding Scheme. Technical report, Video Coding Experts Group (VCEG), March 2001.

[58] D. Marpe, G. Blattermann, G. Heising, and T. Wiegand. Video Compression using Context-based Adaptive Arithmetic Coding. In *Proceedings of the International Conference on Image Processing*, October 2001.

[59] D. Marpe, G. Blattermann, and T. Wiegand. Adaptive Codes for H.26L. Technical report, Video Coding Experts Group (VCEG), January 2001.

[60] D. Marpe, V. George, H.L. Cycon, and K.U. Barthel. Performance Evaluation of Motion-JPEG2000 in Comparison with H.264/AVC Operated in Intra Coding Mode. In *Proc. SPIE*, pages 129–139, February 2004.

[61] D. Marpe, H. Schwarz, and T. Wiegand. Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):620–636, July 2003.

[62] G.N.N. Martin. Range Encoding: An Algorithm for Removing Redundancy from a Digitised Message. In *Proceedings of the Video and Data Recording Conference*, July 1979.

[63] G.R. Martin, M.K. Steliaros, and R.A. Packwood. Efficient Motion Estimation and Coding for Arbitrary-Shaped Video Objects. *Journal of*

Visual Communication and Image Representation, 12(1):66–83, March 2001.

[64] V. Mezaris, I. Kompatsiaris, and M.G. Strintzis. Video Object Segmentation using Bayes-Based Temporal Tracking and Trajectory-Based Megion Merging. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(6):782–795, June 2004.

[65] A. Moffat, N. Sharman, I.H. Witten, and T.C. Bell. An Empirical Evaluation of Coding Methods for Multi-Symbol Alphabets. In *Proceedings of the Data Compression Conference*, pages 108–117, March 1993.

[66] K.T. Mullen. The Contrast Sensitivity of Human Colour Vision to Red-Green and Blue-Yellow Chromatic Gratings. *Journal of Physiology*, 359:381–400, 1985.

[67] Y. Nakaya and H. Harashima. Motion Compensation based on Spatial Transforms. *IEEE Transactions on Circuits and Systems for Video Technology*, 4(3):339–356, June 1994.

[68] R. Neff and A. Zakhor. Very Low Bit-Rate Video Coding Based on Matching Pursuits. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(1):158–171, February 1997.

[69] R. Neff and A. Zakhor. Matching Pursuit Video Coding - Part I: Dictionary Approximation. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(1):13–26, January 2002.

[70] R. Neff and A. Zakhor. Matching Pursuit Video Coding - Part II: Operational Models for Rate and Distortion. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(1):27–39, January 2002.

[71] F.I. Van Nes and M.A. Bouman. Spatial Modulation Transfer in the Human Eye. *Journal of the Optical Society of America*, 57(3):401–406, March 1967.

[72] A.N. Netravali and B.G. Haskell. *Digital Pictures: Representation, Compression and Standards*, page 280. Plenum Press, 1995.

[73] H. Nicolas, S. Pateux, and D. Le Guen. Minimum Description Length Criterion and Segmentation Map Coding for Region-Based Video Compression. *IEEE Trans. Circuits Syst. Video Tech.*, 11(2):184–198, February 2001.

[74] J. Nieweglowski, T. Campbell, and P. Haavisto. A Novel Video Coding Scheme based on Temporal Prediction using Digital Image Warping. *IEEE Transactions on Consumer Electronics*, 39(3):141–150, August 1993.

[75] K.J. O'Connell. Object-Adaptive Vertex-Based Shape Coding Method. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(1):251–255, February 1997.

[76] M. T. Orchard and G. J. Sullivan. Overlapped Block Motion Compensation: An Estimation-Theoretic Approach. *IEEE Transactions on Image Processing*, 3(5):693–699, September 1994.

[77] A. Ortega and K. Ramchandran. Rate-Distortion Methods for Image and Video Compression. *IEEE Signal Processing Magazine*, 15:23–50, November 1998.

[78] P. Jost. TREE BASED PURSUIT 1.0, 2004. `http://lts2www.epfl.ch/~jost/downloads/TBP1.0/index.html`.

[79] P.A. Packwood, M.K. Steliaros, and G.R. Martin. Variable Size Block Matching Motion Compensation for Object-based Video Coding. In *Proc. IPA*, pages 56–60, July 1997.

[80] S. Pateux and C. Labit. Codage efficace de carte de segmentation pour la compression oriente rgions de squences d'images. Technical report, IRISA publication 1073, Jan. 1997.

[81] I. Patras, E.A. Hendriks, and R.L. Langendijk. Video Segmentation by MAP Labeling of Watershed Segments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3):326–332, March 2001.

[82] W.B. Pennebaker and J.L. Mitchell. *JPEG Still Image Data Compression Standard*, page 24. Van Nostrand Reinhold, 1993.

[83] W.B. Pennebaker and J.L. Mitchell. *JPEG: Still Image Data Compression Standard.* Von Nostrand Reinhold, New York, 1993.

[84] F. Pereira, editor. *Signal Processing: Image Communication*, volume 15. European Association for Signal Processing (EURASIP), January 2000. Tutorial Issue on the MPEG-4 Standard.

[85] Charles A. Poynton. *A Technical Introduction to Digital Video.* John Wiley & Sons, Inc., New York, NY, USA, 1996.

[86] A. Puri, H-M. Hang, and D.L. Schilling. Interframe Coding with Variable Block-size Motion Compensation. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*, pages 85–90, November 1987.

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

[88] I. Rhee, G.R. Martin, S. Muthukrishnan, and R.A. Packwood. Quadtree-Structured Variable-Size Block-Matching Motion Estimation with Minimal Error. *IEEE Transactions on Circuits and Systems for Video Technology*, 10(1):42–50, Feb. 2000.

[89] J. Ribas-Corbera and D.L. Neuhoff. On the Optimal Block Size for Block-based, Motion-Compensated Video Coders. In *Proc. VCIP*, volume 3024, pages 1132–1143, February 1997.

[90] I.E.G. Richardson. *Video Codec Design: Developing Image and Video Compresson Systems.* John Wiley and Sons Ltd., 2002.

[91] I.E.G. Richardson. *H.264 and MPEG-4 Video Compression*. John Wiley and Sons Ltd., 2003.

[92] J. Rissanen and G.G. Langdon. Arithmetic Coding. *IBM Journal of Research and Development*, 23(2):149–162, 1979.

[93] A. Said. Introduction to Arithmetic Coding - Theory and Practice. Technical report, HP Laboratories, April 2004.

[94] P. Salembier and F. Marques. Region-Based Representations of Image and Video: Segmentation Tools for Multimedia Services. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(8):1147–1169, December 1999.

[95] G. Schuster, G. Melnikov, and A. Katsaggelos. Operationally Optimal Vertex-Based Shape Coding. *IEEE Signal Processing Magazine*, 15(6):91–108, November 1998.

[96] G. M. Schuster and A. K. Katsaggelos. An Optimal Polygonal Boundary Encoding Scheme in the Rate Distortion Sense. *IEEE Transactions on Image Processing*, 7(1):13–26, January 1998.

[97] V. Seferidis and M. Ghanbari. General approach to Block-Matching Motion Estimation. *Optical Engineering*, 32(7):1464–1474, July 1993.

[98] V. Seferidis and M. Ghanbari. Generalised Block-Matching Motion Estimation using Quad-Tree Structured Spatial Decomposition. *IEE Proceedings on Vision, Image and Signal Processing*, 141(6):446–452, December 1994.

[99] M. Servais. Video Coding Project Web Page, 2005. `http://www.ee.surrey.ac.uk/CVSSP/VMRG/hdtv/`.

[100] M.P. Servais, T. Vlachos, and T. Davies. Bi-Directional, Affine Motion Compensation Using a Content-Based, Non-Connected, Triangular Mesh. In *Proceedings of the 1st IEE European Conference on Visual Media Production (CVMP)*, pages 49–58, March 2004.

[101] M.P. Servais, T. Vlachos, and T. Davies. Motion Compensation using Content-based Variable-Size Block-Matching. In *Proceedings of the 24th Picture Coding Symposium (PCS)*, December 2004.

[102] M.P. Servais, T. Vlachos, and T. Davies. Progressive Polygon Encoding of Segmentation Maps. In *Proceedings of the IEEE International Conference on Image Processing*, pages 1121–1124, October 2004.

[103] M.P. Servais, T. Vlachos, and T. Davies. Affine Motion Compensation using a Content-based Mesh. *IEE Proceedings on Vision, Image and Signal Processing*, 152(4):415–423, August 2005.

[104] M.P. Servais, T. Vlachos, and T. Davies. Motion-Compensation using Variable-Size Block-Matching with Binary Partition Trees. In *Proceedings of the IEEE International Conference on Image Processing*, September 2005.

[105] A. Shamim and J.A. Robinson. Object-Based Video Coding by Global-to-Local Motion Segmentation. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(12):1106–1116, December 2002.

[106] C.E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.

[107] J.R. Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In M.C. Lin and D. Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, 1996. From the First ACM Workshop on Applied Computational Geometry.

[108] J.R. Shewchuk. Triangle: A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator, 2005. `http://www-2.cs.cmu.edu/~quake/triangle.html`.

[109] C. Stiller and J. Konrad. Estimating Motion in Image Sequences. *IEEE Signal Processing Magazine*, 16:70–91, July 1999.

[110] G. Strang and T.Q. Nguyen. *Wavelets and Filter Banks.* Wellesley-Cambridge Press, 1996.

[111] G.J. Sullivan and R.L. Baker. Rate-Distortion Optimized Motion Compensation for Video Compression using Fixed or Variable Size Blocks. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*, pages 85–90, November 1991.

[112] G.J. Sullivan and R.L. Baker. Efficient Quadtree Coding of Images and Video. *IEEE Transactions on Image Processing*, 3(3):327–331, May 1994.

[113] G.J. Sullivan and T. Wiegand. Rate-Distortion Optimization for Video Compression. *IEEE Signal Processing Magazine*, 15:74–90, November 1998.

[114] G.J. Sullivan and T. Wiegand. Video Compression – From Concepts to the H.264/AVC Standard. *Proceedings of the IEEE*, 93(1):18–31, January 2005.

[115] D.S. Taubman and M.W. Marcellin. *JPEG 2000: Image Compression Fundamentals, Standards and Practice.* Kluwer Academic Publishers, Boston, 2002.

[116] A.M. Tekalp, P. Van Beek, C. Toklu, and B. Gunsel. Two-Dimensional Mesh-Based Visual-Object Representation for Interactive Synthetic/Natural Digital Video. *Proceedings of the IEEE*, 86(6):1029–1051, June 1998.

[117] P. Van Beek, A.M. Tekalp, N. Zhuang, I. Celasun, and M. Xia. Hierarchical 2-D Mesh Representation, Tracking, and Compression for Object-Based Video. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(2):353–369, March 1999.

[118] UB Video. Emerging H.26L Standard - White Paper. Technical report, UBVideo Inc., Vancouver, BC, Canada, February 2002.

[119] T. Vlachos and T. Davies. Personal Communication, January 2005.

[120] H. Wang, G.M. Schuster, A.K. Katsaggelos, and T.N. Pappas. An Efficient Rate-Distortion Optimal Shape Coding Approach Utilizing a Skeleton-Based Decomposition. *IEEE Transactions on Image Processing*, 12(10):1181–1193, October 2003.

[121] T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G.J. Sullivan. Rate-Constrained Coder Control and Comparison of Video Coding Standards. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):688–703, July 2003.

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

[123] I.H. Witten, R.M. Neal, and J.G. Cleary. Arithmetic Coding for Data Compression. *Communications of the ACM*, 30(6):520–540, June 1987.

[124] J. Zhang, M.O. Ahmad, and M.N.S. Swamy. A New Variable Size Block Motion Compensation. In *Proceedings of the IEEE Conference on Image Processing (ICIP)*, pages 164–167, October 1997.

[125] J. Zhang, M.O. Ahmad, and M.N.S. Swamy. Quadtree Structured Region-Wise Motion Compensation for Video Compression. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(5):808–822, August 1999.

[126] K. Zhang, M. Bober, and J. Kittler. Variable Block Size Video Coding with Motion Prediction and Motion Segmentation. In *Proceedings of the SPIE Conference on Digital Video Compression: Algorithms and Technologies*, volume 2419, pages 62–70, February 1995.