THREE-DIMENSIONAL CODING OF MOTION VECTOR FIELDS IN VIDEO

by

Joseph Chiaming Yeh

Memorandum No. UCB/ERL M97/87

29 November 1997

THREE-DIMENSIONAL CODING OF MOTION VECTOR FIELDS IN VIDEO

by

Joseph Chiaming Yeh

Memorandum No. UCB/ERL M97/87

29 November 1997

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

THREE-DIMENSIONAL CODING OF MOTION VECTOR FIELDS IN VIDEO

by

Joseph Chiaming Yeh

Memorandum No. UCB/ERL M97/87

29 November 1997

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Three-Dimensional coding of Motion Vector Fields in Video

by

Joseph Chiaming Yeh

Abstract

Current video coding standards such as H.261/3 and MPEG rely mainly on a combination of Discrete Cosine Transform coding and Motion Estimation and compensation to achieve compression of image sequences. The resulting coding method exploits both spatial and temporal redundancies in the image data. Motion vector fields are used to predict one image from another, eliminating the need to resend image data for an object which has only moved translationally between two subsequent images. The more motion information sent, the better the prediction of the subsequent image. However, the motion information itself can occupy too much bandwidth to justify the better prediction quality.

Hence, it is necessary to find ways to efficiently compress motion vector information. Pel recursive schemes, which offer pixel accurate motion information at low bandwidth, have long been investigated in academic circles, however their computational cost has precluded them from wide acceptance. We present two methods of compressing motion vector information using standard "Blockmatching" techniques for estimating the motion vectors.

The first method takes advantage of an expected consistency between two successive motion vector fields. To explain, if the objects within the camera's view have not accelerated at all within a frame, then it should be possible to accurately predict the next motion vector field. Our first method predicts a subsequent motion vector field from a previous one, then uses this new motion vector field to estimate the next frame in an image sequence, eliminating the need for the encoder to transmit another field.

The second method uses the well-known zerotree method to code motion vectors. A modified version of Shapiro's zerotree algorithm is used to encode three dimensional array of motion vectors over space and time, taking advantage of both spatial and temporal redundancies to achieve coding efficiency. Experiments and results show that these methods may be promising approaches to pursue.

Acknowledgments

Many people have helped me out through my time at Cal. Above all, I want to thank my family, Cindy, and her family for their love and support. I am forever indebted to Ruth Gjerde and the folks at the Graduate Student Administrative Office for their indispensable help.

My advisor, Martin Vetterli, has been the primary supporter of this work. He introduced me to the topic of motion estimation in video over two years ago, and has since continually encouraged me on this subject. In general, Martin has encouraged me in my academic career. Martin has a unique interest in the intellectual development of his students, and in turn has a unique ability to inspire them.

My gratitude also goes out to my fellow students in the Wavelet Research Group: Grace Chang, Michael Goodwin, Vivek Goyal, Francis Ng, and Matt Podolsky. Masoud Khansari, now at Hewlett-Packard Laboratories, was very influential in the early stages of this work. Some of the work in this thesis was also inspired by discussions with Paul Haskell of NextLevel Systems, Janusz Konrad of INRS Telecommunications, and Michael Orchard of Princeton University. Some of the video sequences used in this work were supplied by Ralph Neff of the Video and Image Processing (VIP) Lab here at Cal. Thanks also go out to Avideh Zakhor, head of the VIP lab, for agreeing to be the second reader of this thesis.

Finally, I would like to thank all the professors, teaching assistants and fellow students I have not mentioned who have helped me educate myself throughout my six years at Cal, first as an undergraduate and then as a graduate student.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

It is well-known that video signals contain a large amount of redundancy along the temporal dimension. Therefore, considerable compression is achieved by removing this redundancy. This is indeed the approach taken in various video compression standards and other proposed coding algorithms. A popular approach is to detect and compensate for image motion to predict the present frame from one or more previous frames. The values of these motion vectors and the prediction residuals are then transmitted.

In very low bitrate video coding, the bitrate used to represent motion vectors can become a non-trivial portion of the bit budget. In medium to high bitrate coding, denser motion vector fields (MVF's) could predict frames better, but the bitrate consumed by the MVF's could exceed the bitrate saved by better prediction. Thus, all types of video coding need efficient methods of coding MVF's. So far, only intraframe dependency has been used to code MVFs and the dependencies among these consecutive MVFs in time have not been taken into consideration.

Currently, MPEG-1 and 2 use first-order intraframe differential coding of motion vectors to compress motion data. Although intraframe differential coding will be effective when the motion field is smooth, it does not achieve satisfactory results when the motion vectors vary greatly from one block to the next. Differential coding is the most common strategy used to compress MVF's, although work has been done to achieve compression with vector quantization of motion vectors [9] and layered decomposition of motion vectors [19].

This thesis presents two methods for taking advantage of redundancies between subsequent motion vector fields. The first one, "Autocompensation" is presented in Chapter 4. The second one, zerotree coding using Shapiro's method over three dimensions, is presented

in Chapter 5. The rest of this introduction will explain the structure of the video coders that employ these two methods by comparing and contrasting them with the structure of a conventional hybrid video coder.

1.1 Structure of conventional video coders

Figure 1.1 shows a block diagram for a conventional video coder. A image sequence is transmitted by first sending an initial image I_0 to a receiver, after which the motion compensation loop can begin. The encoder then segments the next frame I_1 to be sent into square blocks, all of the same size. For each block of I_1, the encoder tries to find the best approximate match in the previous image and records the displacement of this match from the location in I_1. The encoder then sends the displacements (motion vectors) of each block, so the decoder can attempt to assemble the I_1 out of blocks cut from I_0.

After this step, the decoder has $\hat{I}_1$, a (usually poor) approximation of I_1. The encoder then encodes the error (motion residual) $I_1 - \hat{I}_1$ block by block with a DCT transform and then sends the encoded residual to the receiver. The loop can begin again, with the encoder segmenting the next frame I_2 into blocks and then trying to find ideal displacements in I_1.

At times, new objects can appear in a image sequence. In this case, the encoder is unable to find a match in the previous image. In this case, no motion vector is sent, and the image data for the new objects is coded and sent to the receiver.

1.2 Motivation for Three-Dimensional Video Coders

Lately, subband coding for video has been a subject of interest, with Taubman and Zakhor [16], Karlsson and Vetterli [8], and Ohm [12] all making significant contributions. Typically, a group of frames is taken together as a three-dimensional array and filtered into different frequency bands along spatial and temporal dimensions. One motivation for coding frames together in a group is that coding a signal in whole is more efficient than dividing it into parts and encoding each part separately.

However, as of this writing, DCT based methods still dominate digital video communication. In general, subband coders have not been able to outperform MC-DCT coders enough to justify their increased complexity. The following section discusses a strategy to use simple MC-DCT techniques in three-dimensional coders, in order to still take advantage

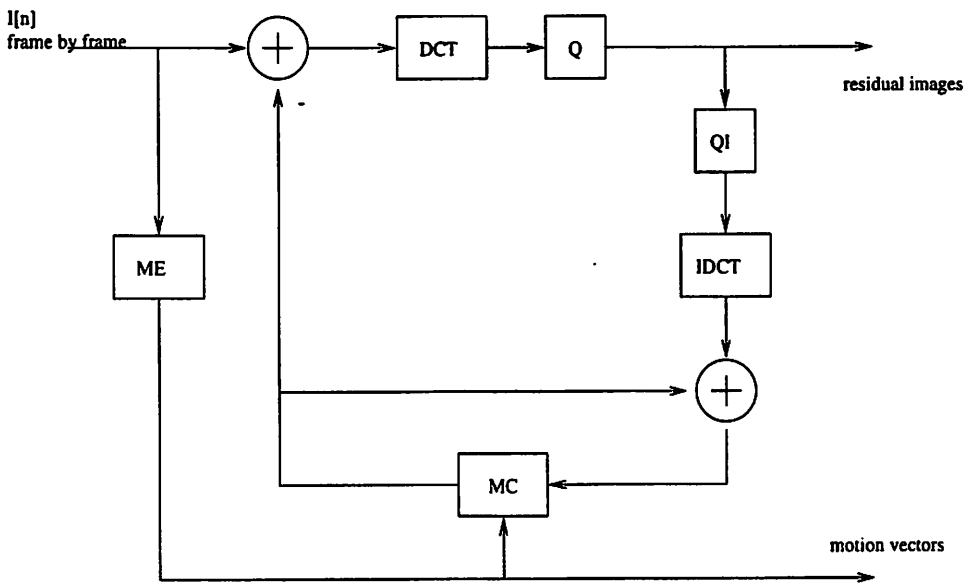


Figure 1.1: Block diagram of a conventional hybrid video coder.

the associated coding benefits of coding frames together in groups.

1.3 Structure of Three-Dimensional MC-DCT Coders

The coder described in this thesis mimics the conventional coder structure above except in one major aspect. Although the conventional coders exploit the similarity between adjacent frames of an image sequence, information is essentially sent frame by frame to the receiver. In our implementation, information is sent to a receiver in groups of N frames.

Figure 1.2 shows a general block diagram for our implementation. GMC stands for "Group Motion Compensation." Motion estimation and compensation (the reconstruction of the image using data from previous images) is done inside this unit. The N MVF's resulting from the procedure are then encoded and sent to the receiver. The N motion residuals are then gathered together and coded with a 3D DCT transform discussed in Chapter 3.

We employ this structure for our coder because we wish to exploit redundancies between subsequent MVF's, hence we need to code MVF's together in groups. We then code the residual with a 3D DCT for two reasons: 1) if image data is being sent in groups instead of frame by frame, then we might as well use a 3D DCT and exploit any existing temporal

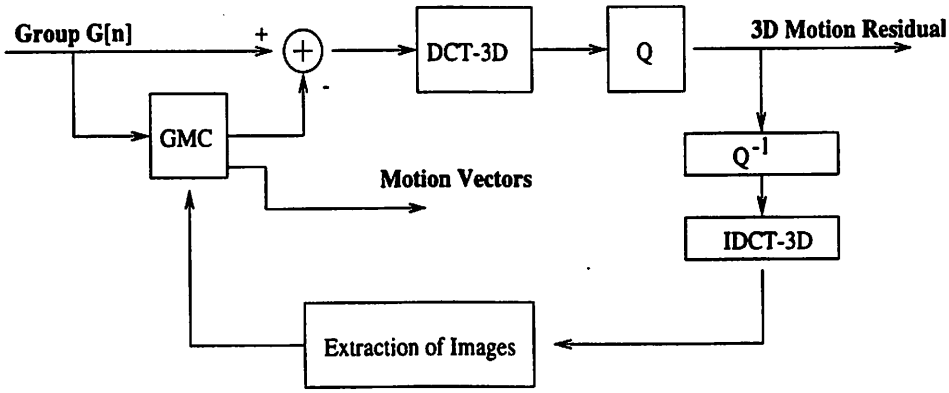


Figure 1.2: General block diagram of the coders in this thesis.

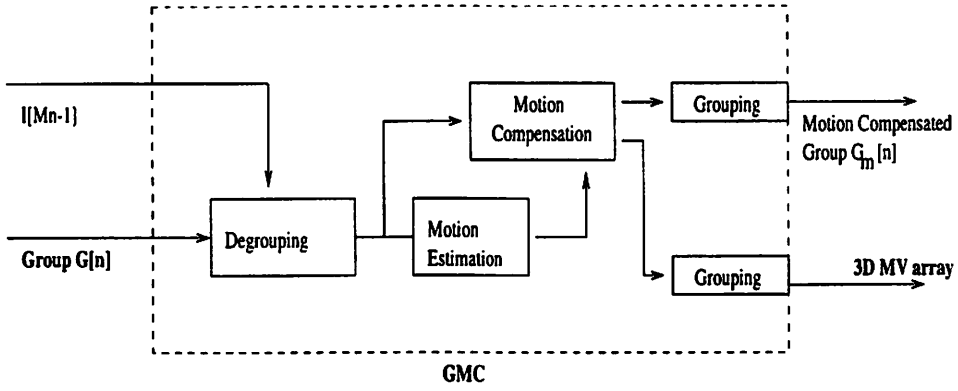


Figure 1.3: GMC diagram.

redundancy in the image data, 2) we are using small blocksizes in these implementations, which decrease the coding gain that we get from a 2D DCT; a 3D DCT helps us to achieve more.

This thesis presents experiments with different implementations of Group Motion Compensation. The zerotree implementation uses the method explained in Chapter 2 to estimate motion vectors, and the Autocompensation implementation uses the procedure explained in Chapter 4 to estimate motion vectors. In both cases, the GMC can be summarized by Figure 1.3. Motion estimation and compensation is done for all the images in the current group, without any residual being added in. The last images in the previous group need to be input to approximate the first image in the current group.

1.4 Outline of the Report

Chapter 2 describes Overlapped Block Motion Estimation (OBME), a slight modification to conventional block-matching techniques which is used to estimate high-density MVF's. Section 2.1 reviews the basics of block-matching motion estimation. Section 2.2 explains why conventional block-matching motion estimation might be detrimental in some situations, and Sections 2.3 through 2.5 present and explain OBME as an extension to conventional block-matching which can succeed where conventional block-matching fails. Section 2.6 presents some simulations which compare OBME to conventional block-matching techniques. Section 2.7 discusses the computational complexity of OBME.

Chapter 3 discusses the Three-Dimensional Discrete Cosine Transform used in the block diagrams above. Sections 3.1 through 3.4 discuss the mathematical formulation of the DCT, its extensions to two and three dimensions, and its use in coding image data. Section 3.5 discusses the computational complexity of the 3D-DCT, Section 3.6 presents a method for scanning 3D-DCT coefficients for transmission, and Section 3.7 concludes with some simulations.

Chapter 4 describes Autocompensation, a technique for estimating and encoding vectors which reduces the bitrate occupied by the vectors. Sections 4.1 and 4.2 give a general overview of the technique, and Section 4.3 explains some modifications to the technique for coding block-based motion vectors. Finally, Section 4.4 presents simulation results for a 3D MC-DCT coder based on Autocompensation.

Chapter 5 discusses the zerotree algorithm used to code 3D arrays of motion vectors. Section 5.1 discusses some differences between vector and image data which make it difficult for standard image compression techniques to be applied to motion data. The chapter then goes on to explain our implementation in Sections 5.2 and 5.3. Finally, simulations are presented in Section 5.4.

Chapter 6, the conclusion of this thesis, summarizes the work presented in the thesis and goes on to suggest future directions to pursue.

Chapter 2

Overlapped Block Motion Estimation

Schemes like autocompensation which attempt to reduce the bitrate occupied by MVF's need a consistent estimate of motion, not just an estimate which reduces the prediction error of an image. For this purpose, we have developed the motion estimation strategy presented in this chapter.

This chapter begins with an explanation of how blockmatching motion estimation algorithms are used in current video coding standards to reduce bandwidth. It then explains our method of using overlapped blocks and tries to justify why we need it in our coding algorithm.

2.1 Basic Blockmatching Motion Estimation

As mentioned in the introduction, consecutive frames in a video sequence will often differ very little in content. Most often, either objects in the image or the background might have shifted. An intuitive solution is to attempt to code the motion between one frame to the next, so that the decoder could compose a subsequent frame from a previous one. Blockmatching motion estimation is a common method used to deduce the motion between two frames.

In blockmatching motion estimation, a frame is partitioned into square blocks of size N by N. In coding algorithms such as MPEG and H.263, block sizes of 8 by 8 and 16 by 16 are used. The motion vector is calculated by searching for the (p, q) which minimizes

the following summation:

$$\sum_{i=0}^{N}\sum_{j=0}^{N} D[I_c(b_x+i, b_y+j), \hat{I}_p(b_x+i+p, b_y+j+q)]$$

where D is a mismatch function that indicates the amount of dissimilarity between two pixel values, I_c refers to the current frame to be coded, and $\hat{I}_p$ refers to the decoded version of the previous frame. The most commonly used distance metric is the Sum of Absolute Differences (SAD). In this case the mismatch function $D[u, v]$ is simply $|u - v|$.

Less commonly, a motion vector can be calculated by searching for the (p, q) which maximizes

$$\sum_{m,n\in B} S[I_c(m, n), \hat{I}_p(m+p, n+q)] \tag{2.1}$$

where S is a similarity function. One such similarity function is a product, where $S[u, v] = uv$. This would ideally estimate the cross-correlation between the two 2D signals. Vargas [17] discusses the advantages and implementation of using similarity functions for motion estimation.

In the case that a new object appears in a frame (e.g. a person coming in through a door), there would be no way to estimate that information based on a previous frame, so a motion vector cannot be estimated. Most likely, the above summation would not go below a certain threshold T, and the encoder could deduce that a suitable (p, q) does not exist.

2.2 Potential Problems of SAD Blockmatching Motion Estimation

After the motion vectors are estimated and sent to the decoder, the decoder compensates the previous image in order to try to produce an approximation to the current image. This approximation is usually far from acceptable as the final decoded version of the video frame, so an approximation to the residual must be sent to the decoder via transform coding. One reason that blockmatching motion vectors are so widely used is that they are extremely compatible with the DCT: the error comes in square blocks already "formatted" for the DCT.

Typically, the larger the block size, the greater the error will be over the whole image. This occurs because of two main reasons: (1) the larger block is more likely to contain

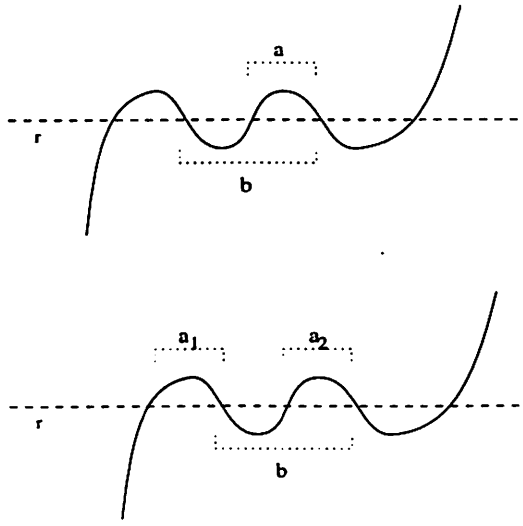


Figure 2.1: Illustration of ambiguity caused when blocksize becomes too small

image data from two different objects moving in two different directions, and is unable to describe the motion of both; (2) block motion vectors can only describe translation motions orthogonal to the line of sight of the camera, whereas objects might be rotating or the camera might be zooming in and out.

To further illustrate, we compare N by N block motion estimation with MN by MN block motion estimation. Essentially, N by N motion estimation will give better results because

$$\min \sum_{m,n \in K} D[I_c(m,n), \hat{I}_p(m+p, n+q)] \geq \min \sum_{B_i \in K} \sum_{m,n \in B_i} D[I_c(m,n), \hat{I}_p(m+p_i, n+q_i)]$$

where K is a MN by MN block, which can also be seen as a collection of M^2 blocks B_i of size N by N.

The solution is then to use smaller block sizes. However, when block sizes get smaller, there is a certain ambiguity. This ambiguity can be illustrated in Figure 2.1, which is a graph of a hypothetical signal in 1-D:

Say the signal on top is the current frame to be coded, and the signal on bottom is the decoded version of the previous frame. The signal moved slightly to the right with respect to reference point r. When we estimate the block corresponding to b, we are able to find it's displacement in the previous image.

Now we want to use block sizes that are half the size (in 2-D they would be quarter

the size). Then when we try to estimate the block corresponding to a, we see two possible locations in the previous image.

If the signal is predicted equally well from either location, then it shouldn't matter whether we use the vector corresponding to a_1 or the "true motion" in a_2. However, it is important to pick the correct motion because that will cause the resulting motion vector field to be smoother and therefore easier to encode.

2.3 Overlapped Blocks

Somehow, we would like to have the better predicting power of a dense motion vector field with small block sizes but still have a smooth motion vector field which actually represents the motion of the image. This section discusses Overlapped Block Motion Compensation (OBMC), and how we can use a similar approach to estimate a smooth, dense motion vector field using block based techniques.

First, in [1], Auyeung, Kosmach, Orchard, and Kalafatis introduced OBMC, which was eventually included in the teleconferencing video standard H.263 [5]. OBMC is currently primarily used to reduce the blocking artifacts associated with hybrid coded video. In OBMC, the decoder uses many vectors instead of just one to predict the value of a pixel. Without OBMC, the predicted value of a pixel $\tilde{I}(m,n)$ is

$$\hat{I}(m+p, n+q)$$

where $\hat{I}$ is the previous coded image and (p,q) is the motion vector of the corresponding block of pixel (m,n) In OBME, it is

$$\sum_{j=0}^{K} \alpha_j \hat{I}(m+p_j, n+q_j)$$

where the α's are weights under the constraint $\sum_{j=0}^{K} \alpha_j = 1$ and the (p_j, q_j)'s are the motion vectors of a group of blocks in the vicinity of (m,n)

OBMC uses more than one vector to compute a pixel value, so we propose a technique called Overlapped Block Motion Estimation (OBME) to use a specific pixel to compute more than one vector. More specifically, we generalize motion estimation so that for a given block B, we use the criteria

$$\sum_{m,n \in B'} D[I_c(m,n), \hat{I}_p(m+p, n+q)]$$

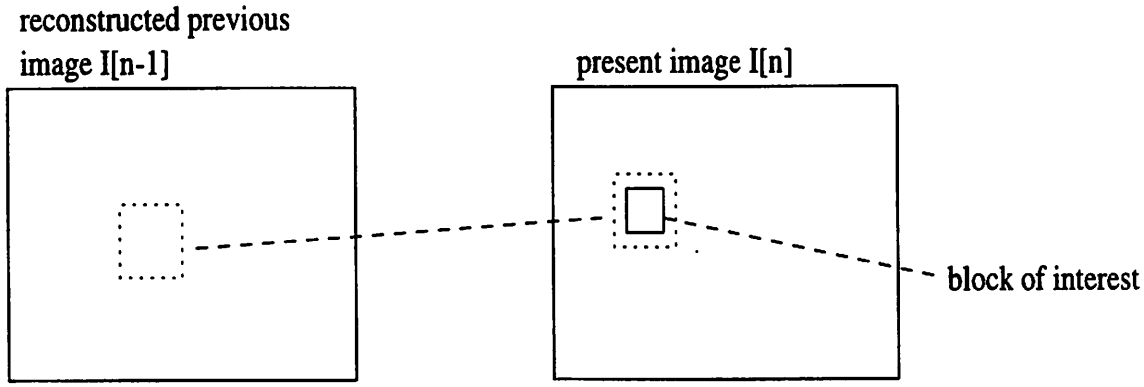


Figure 2.2: Illustration of Overlapped Block Motion Estimation

where $B \subseteq B'$, so pixels outside of B are used in the calculation. To further illustrate, Figure 2.2 shows the surrounding area of a block B used in estimating its motion vector.

In the zerotree method used to code vectors in this project, it helps to have motion vector fields that are not only smooth throughout a frame, but also motion vector fields that are smooth throughout time. For this reason, we have also considered using the criteria

$$\sum_{m,n \in B'} (D[I_c(m,n), \hat{I}_p(m+p, n+q)] + D[\hat{I}_p(m,n), \hat{I}_{p-1}(m+p, n+q)]) \qquad (2.2)$$

where $\hat{I}_{p-1}$ is the decoded frame preceding $\hat{I}_p$ in time. In other words, instead of only using pixels in B' in the current image, we also use pixels in B' from the previous image to help us estimate motion vector fields that are smooth over time. Figure 2.3 illustrates this concept.

2.4 Windowing

Although OBME results in a smoother motion vector field, the vectors themselves will not be as good at predicting pixel values. A vector (p_n, q_n) estimated in non-overlapped Block Motion Estimation gives the minimum for the mismatch function summed over all pixels in B, the vector (p_o, q_o) found in OBME cannot do any better, and usually will do worse. More specifically,

$$\sum_{m,n \in B} D[I_c(m,n), \hat{I}_p(m+p_o, n+q_o)] \geq \sum_{m,n \in B} D[I_c(m,n), \hat{I}_p(m+p_n, n+q_n)]$$

The solution then is to weigh the differences of the summation corresponding to pixels

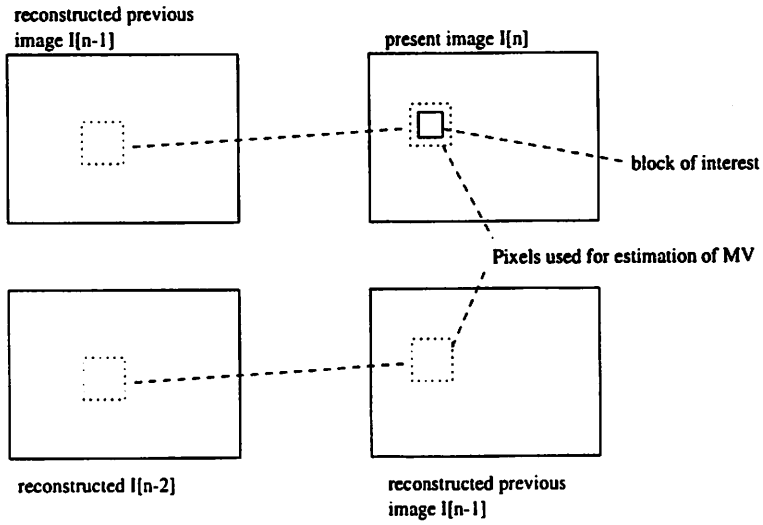


Figure 2.3: Illustration of OBME using pixels in B' from two frames.

(m, n) inside B more than the differences corresponding to pixels in $B' \backslash B$. The criterion for the ideal vector then becomes

$$\sum_{i=0}^{M-1} \sum_{j=0}^{M-1} \alpha_{i,j} D[I_c(b_x + i, b_y + j), \hat{I}_p(b_x + i + p, b_y + j + q)]$$

where α is a 2-D weighting matrix for each of the terms in the summation. Usually, as in OBMC, $\alpha_{i,j}$ is separable, i.e. $\alpha_{i,j} = \gamma_i \gamma_j$ where γ_k is an 1-D weighting array. In this project we use a "raised cosine" window defined by

$$\gamma_k = \frac{1}{2}(1 + \cos \frac{\pi(2k-7)}{8}), k = 0, 1, \ldots, 7$$

when $M = 8$.

As seen in the last section of this chapter, windowed OBME does lower the first-order entropy estimates of the dense motion vector fields while producing better prediction results than coarse motion vector fields. One disadvantage of OBME, especially in light of real time encoding and decoding, is the increased computational load, which will be discussed in the last section. The next section discusses the improvements made by OBMC from a mathematical standpoint.

2.5 AR(1) Model

Even if there is no ambiguity as described in Section 2.2, there is still a potential for error in the motion estimate. In this section we consider an abstraction of the block matching problem using a 1-D Gaussian AR(1) process. The intention is to gain insight into the effect of the blocksize on the reliability of motion estimates when there is a underlying "true" motion.

Although motion vectors are more often estimated by minimizing a dissimilarity function, in this section motion vector calculation is assumed to be done from maximizing the product, i.e. with the $S[u,v] = uv$ function shown in section 2. This makes the analysis in this section more manageable.

We further simplify the problem by considering the problem in 1-D (looking at signals $I[n]$ instead of $I(m,n)$ with no windowing). We then look at the current "image" $I[n]$ as a Gaussian AR(1) process with autocorrelation function $R_I[m] = \rho^{|m|}, \rho < 1$. We view it as a shifted version of the previous decoded image $\hat{I}[n]$, i.e.

$$I[n] = \hat{I}[n + \hat{p}]$$

where $\hat{p}$ is the "true displacement," and the motion estimation should ideally find this $\hat{p}$. The criterion to maximize (2.1) becomes:

$$\frac{1}{M} \sum_{n=0}^{M-1} I[n]\hat{I}[n+p]$$

or

$$\frac{1}{M} \sum_{n=0}^{M-1} I[n]I[n-\hat{p}+p] \tag{2.3}$$

Ideally, the cross correlation function $R_{I\hat{I}}[m] = E\{I[n]\hat{I}[n+m]\}$ should be maximized at $\hat{p}$, and the algorithm would return the ideal vector. However, since we can only work with estimates of the cross correlation function, there will be some variance, and hence our final guess of the vector may be wrong. The greater the variance is, the bigger the chance that we will make a mistake.

Setting $d = \hat{p} - p$, we can find the variance σ_e^2 of 2.3 as follows:

$$\sigma_e^2 = E\{\frac{1}{M} \sum_{n=0}^{M-1} I[n]I[n+d] - E\{\frac{1}{M} \sum_{n=0}^{M-1} I[n]I[n+d]\}\}^2$$

$$
\begin{aligned}
&= \frac{1}{M^2} E\{\sum_{n=0}^{M-1} (I[n]I[n+d] - E\{I[n]I[n+d]\})\}^2 \\
&= \frac{1}{M^2} E\{\sum_{n=0}^{M-1} I[n]I[n+d] - M\rho^d\}^2 \\
&= \frac{1}{M^2} E\{\sum_{n=0}^{M-1} I[n]I[n+d]\}^2 - \rho^{2d} \\
&= \frac{1}{M^2} \sum_{n=0}^{M-1}\sum_{m=0}^{M-1} E\{I[n]I[n+d]I[m]I[m+d]\} - \rho^{2d} \qquad (2.4) \\
&= \frac{1}{M^2} \sum_{n=0}^{M-1}\sum_{m=0}^{M-1} (E\{I[n]I[n+d]\}E\{I[m]I[m+d]\} + \qquad (2.5) \\
&\quad E\{I[m]I[n]\}E\{I[m+d]I[n+d]\} + E\{I[n]I[m+d]\}E\{I[m]I[n+d]\}) - \rho^{2d} \\
&= \frac{1}{M^2} \sum_{n=0}^{M-1}\sum_{m=0}^{M-1} (E\{I[m]I[n]\}E\{I[m+d]I[n+d]\} \\
&\quad + E\{I[n]I[m+d]\}E\{I[m]I[n+d]\}) \\
&= \frac{1}{M^2} \sum_{n=0}^{M-1}\sum_{m=0}^{M-1} (\rho^{2|m-n|} + \rho^{|d+(m-n)|}\rho^{|d-(m-n)|}) \\
&\leq \frac{2}{M^2} \sum_{n=0}^{M-1}\sum_{m=0}^{M-1} \rho^{|m-n|} \\
&= \frac{2}{M^2}(M + 2\sum_{n=0}^{M-1} (M-n)\rho^n) \\
&\leq \frac{2}{M}(1 + 2M\sum_{n=0}^{M-1} \rho^n) \qquad (2.6) \\
&\leq \frac{2}{M}(1+\alpha) \qquad (2.7)
\end{aligned}
$$

Where $\alpha = \frac{2}{1-\rho}$ and we use the fact that for four jointly Gaussian random variables W,X,Y, and Z, $E\{WXYZ\} = E\{WX\}E\{YZ\} + E\{WY\}E\{XZ\} + E\{WZ\}E\{XY\}$ in going from (2.4) to (2.5) [10]. We can then see that the upper bound for σ_e^2 decreases with M, so then the estimate of the motion vector is more certain for larger blocksizes.

Although there is no rigorous derivation for dissimilarity minimizing motion vector searches, the result for cross-correlation estimation should help in understanding why larger block sizes result in smoother motion vector fields.

2.6 Simulations

We did experiments on the first 9 frames of the standard video coding test sequences *Foreman, Carphone,* and the HDTV testing sequence *MIT.* For the first two, we used QCIF size frames, and for the last one, we cropped the 512 × 512 images to QCIF size. In the experiments, we used four different types of motion estimation:

1. One vector for every 8 × 8 block, estimated using only the pixels in the block ($B = B'$).
2. One vector for every 4 × 4 block, estimated using only the pixels in the block ($B = B'$).
3. One vector for every 4 × 4 block, estimated using the 8 × 8 block which contains the 4 × 4 block at its center ($B \subset B'$).
4. The same as above except the window from Section 2.4 is used on the 8 × 8 block, and weighted pixels from the 4 × 4 block in the previous frame are also used as in Equation 2.2 (Their effect is negligible).

We used the first frame to approximate the second, then used that approximation to approximate the third, etc. all the way to the ninth frame. We basically simulated a video coder with no addition of motion residual. With the 8 estimated frames, we then computed the mean absolute difference of each pixel from the original, and the mean PSNR from the original.

We also calculated a histogram of the motion vectors to model a pdf, and then calculated the entropy of that pdf. In this way we could estimate how hard it would be to encode the motion vectors. These results are presented in Tables 2.1 through 2.4.

As can be seen, the 4 × 4 vectors referred to in Table 2.2 exhibit much more first-order entropy than the 8 × 8 vectors in Table 2.1, but they do a much better job of predicting the images. In a Huffman-style coder, 4 × 4 vectors would be much harder to code because of the higher entropy and the fact that there are four times as many of them. Using 8 × 8 blocks to estimate 4 × 4 vectors in Table 2.3 reduces the first-order entropy, but the prediction quality is compromised. Finally, combining the window from Section 2.4 and the use of previous frame pixels as in Equation 2.2 allows us to keep the entropy under control with less compromise in the prediction quality.

Sequence	Mean DFD	Mean PSNR	Entropy (Bits per Vector)
Foreman	3.30	31.90	5.751
MIT	6.01	25.77	5.164
Carphone	3.20	31.46	6.900

Table 2.1: Simulation Results for 8 × 8 motion vectors

Sequence	Mean DFD	Mean PSNR	Entropy (Bits per Vector)
Foreman	2.25	34.71	7.411
MIT	4.44	29.11	8.795
Carphone	2.24	34.40	8.731

Table 2.2: Simulation Results for 4 × 4 motion vectors: ($B = B'$)

Sequence	Mean DFD	Mean PSNR	Entropy (Bits per Vector)
Foreman	3.09	32.76	5.661
MIT	5.88	26.29	5.377
Carphone	2.99	32.28	7.174

Table 2.3: Simulation Results for 4 × 4 motion vectors: ($B \subset B'$)

Sequence	Mean DFD	Mean PSNR	Entropy (Bits per Vector)
Foreman	3.04	33.57	4.835
MIT	4.16	29.49	3.181
Carphone	2.79	33.90	3.704

Table 2.4: Simulation Results for 4 × 4 motion vectors estimated with a 8 × 8 window in the current frame and a corresponding 4 × 4 window in the previous frame

2.7 Computational Complexity

Without OBME, when the blocksize for computing motion vectors with a minimum SAD criterion is reduced from MN by MN to N by N while the search region remains the same, the number of calculations made in computing the distance metric for each displacement (p, q) remains the same, while the number of comparisons goes up by a factor of M^2, because there are M^2 as many vectors to compute.

When OBME is introduced, the number of calculations made in computing the distance metric also goes up by a factor of $\frac{|B'|}{|B|}$, where $|X|$ is the number of pixels in set X. Furthermore, introducing windows (which may consist of floating point values) adds another computational overhead.

However, in the case of MSAD, many calculations will be repeated, since the sets B' are not mutually exclusive, so it may be wise to calculate and store the absolute differences for each pixel in the image to be coded over all possible displacements (p, q) in the previous decoded image, and then compute the block mismatch functions from the stored values. This would prevent some redundancy in calculation, but the memory requirements would be huge.

Chapter 3

Three Dimensional Discrete Cosine Transform

3.1 Definition and Purpose of Discrete Cosine Transform

The Discrete Cosine Transform (DCT) has been a workhorse of video coding since its introduction in the 1970's. Today, both MPEG and the H.26x standards are primarily based on a combination of motion estimation/compensation and the DCT. When an $N \times N$ block of pixels is viewed as a matrix A, the DCT is basically a similarity transform $CAC^{-1} = B$, where the matrix B is mostly comprised of numbers close to zero and is therefore easier to transmit and/or store after quantization as a matrix $\hat{B}$. When the image data needs to be used again, either when receiving encoded data or retrieving an image file for examination, an inverse transform $\hat{A} = C^{-1}\hat{B}C$ is performed for decoding.

3.2 Approximation to Karhunen Loeve Transform

The DCT compresses image data well because it is a very close approximation to a Karhunen-Loeve Transform (KLT) for an AR(1) process with correlation coefficient ρ close to 1. The KLT is defined as a linear transform $Ka = b$ where a is a block of samples from a wide-sense-stationary process, and the columns of K are the eigenvectors of the autocorrelation matrix R_a of the wide sense stationary process.

The KLT is the ideal transform for concentrating the "energy" of the elements of a in as few of the elements of b as possible. Therefore, since both rows and columns of images

can be reasonably modeled as AR(1) processes, it makes sense to perform the transform on blocks of images.

3.3 Mathematical Definition

The DCT [6] $A[k], k = 0, 1, \ldots, N-1$ of a sequence $a[n], n = 0, 1, \ldots, N-1$ is given by

$$A[k] = \beta[k] \sum_{n=0}^{N-1} a[n] \cos[\frac{\pi(2n+1)k}{2N}] \tag{3.1}$$

where $\beta[0] = \sqrt{\frac{1}{N}}$ and $\beta[k] = \sqrt{\frac{2}{N}}$ for $k \neq 0$.

This is also equivalent forming a column vector a out of $a[n]$ and left-multiplying it by a matrix C to produce a column vector A of $A[k]$'s. The matrix C is given by

$$C_{ij} = \begin{cases} \frac{1}{\sqrt{N}} & \text{if } i = 0 \\ \sqrt{\frac{2}{N}} \cos \frac{\pi(2j+1)i}{2N} & \text{otherwise} \end{cases}$$

This matrix is *unitary*, having the property that $C^{-1} = C^T$.

In the case of higher dimensional data such as 2D or 3D arrays of pixels, the DCT is implemented by doing the transform in each dimension separately. For example, in order to transform a $N \times N$ array of pixels, every row would be transformed, and then every column would be transformed (it does not matter whether rows or columns are done first). To be consistent with our definition in the first section, the left multiplication by C transforms the columns of A, whereas the right multiplication by $C^{-1} = C^T$ transforms the rows of A. To transform a 3D $N \times N \times N$ array, every $N \times N$ frame in the array would be transformed, and then every segment across time would be transformed (again, the order is not important).

3.4 Coding Gain

The DCT performs well in compressing signal data because it has a significant coding gain [7] G_{DCT} (over Pulse Code Modulation). This means that if a signal is first DCT-transformed and then quantized, the resulting signal to noise (SNR) ratio will be G_{DCT} times that if the signal is simply quantized. A lot more than quantization goes into the encoding of images, making the "coding gain" formulas incomplete for describing the whole process, but this is a significant first step.

Since the DCT is an approximation of the KLT, the coding gain G_{DCT} of the DCT will be less than or equal to the coding gain G_{KLT}. We can therefore use the coding gain analysis of the KLT to estimate an upper bound for the coding gain of the DCT. The KLT coding gain for coding N samples of a 1-D AR(1) signal is given by [7]

$$^{N}G_{KLT} = (1-\rho^2)^{-\frac{N-1}{N}}$$

Since $\rho < 1$, it seems as if we should let the size N of one side of the block become as large as possible. However, if N becomes too large, the AR(1) model of the data starts to fail. For this reason, Chan and Siu [2] have introduced the idea of doing 3D-DCT's of variable sizes, varying the size along the temporal axis to coincide with scene changes or fast motion in the video.

3.5 Computational Complexity

In Section 3.3, we discovered how the DCT transform in one dimension can be seen as a matrix multiplication $Ca = b$. In general, this calculation takes N^2 multiplications and $N(N-1)$ additions. A first glance at (3.1) would reach the same conclusion. However, certain matrices have structures which allow the product Ca to be computed faster.

In particular, the Discrete Fourier Transform (DFT) and other related transforms, including the DCT have "decimation in time" and "decimation in frequency" algorithms which allow the computation to be done in $N \log_2 N$ multiplications and $N \log_2 N$ additions if N is a power of 2 [13]. For this reason, DCT's are almost always done with a size which is a power of 2.

The decimation algorithms work by segmenting a size N transform into 2 size $N/2$ transforms recursively until the size N transform is broken down into $N/2$ size 2 transforms. Each segmentation cost N additions and N multiplications.

In transforming an N by N block from an image, $2N$ 1-D transforms are performed. In general, in transforming an M dimensional block with side N, MN^{M-1} 1-D transforms will be done, bringing the computational complexity up to $O(MN^M \log_2 N)$.

In the coder described in this paper, $\frac{MN}{16}$ $4 \times 4 \times 4$ transforms would be done on 4 consecutive images of size M by N. On the same set of images, a more conventional coder (such as H.263) would do $\frac{MN}{16}$ 8×8 transforms. According to the analysis above,

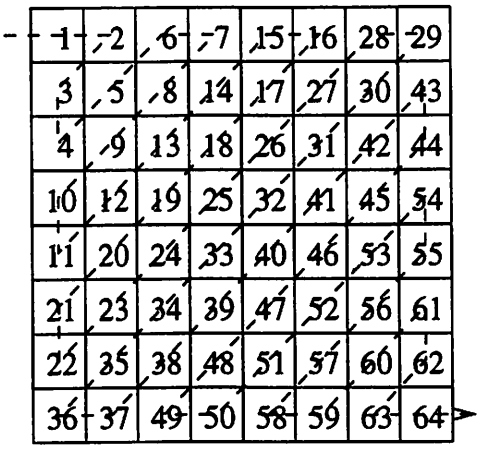


Figure 3.1: Illustration of zigzag scan for a 8×8 array

they each would perform 384 multiplications and additions for each transform, so the total computational cost is the same.

3.6 Transmission of DCT coefficients

Since a bitstream is one-dimensional in structure, a 2-D or 3-D array has to be scanned into a 1-D data structure for transmission or storage. One trivial way to do this would be to "raster" scan the data line-by-line along one dimension, but a better method would take advantage of the fact that most of the higher frequency AC coefficients will be zero, and the fact that the 1-D array of scanned coefficients will be run-length encoded before transmission.

For this reason, people have developed the zigzag scan. Figure 3.1 illustrates the zigzag scan for a 2-D array. The DC coefficient is scanned first, and then the AC coefficients are scanned, with lower frequency being scanned first and higher frequencies scanned later. We want to maximize the lengths of zeros in the scan; in particular, if we have a long block of zeros at the end of the scan, they do not need to be transmitted because an "end of block" (EOB) symbol is sent with the last nonzero coefficient.

This "zigzag" scan satisfies the following two criteria:

1. If we calculate the sum $i + j$ (where i is the row index and j is the column index) for each pixel in the array, we see that we scan the pixels where $i + j = 0$ (the DC

coefficient), and then all the pixels where the $i+j = 1$, and so on up to $i+j = 2(N-1)$ where N is the size of the block edge.

2. We also see that the nth and $(n+1)$th positions in the scan differ at the most by one in their row indices or their column indices. That is, if i_n and j_n are the row and column indices of the nth position in the scan, then $|i_n - i_{n+1}| \leq 1$ and $|j_n - j_{n+1}| \leq 1$.

The first criterion stems from an assumption that higher frequency AC coefficients are more likely to be zero, and the sum $i+j$ is a measure for how "high" the coefficients are in frequency. It can also be derived from the fact that data from image rows and columns can be modeled as AR(1) processes, and the closer the correlation coefficient ρ is to 1, the better the DCT will perform. Although the ρ of columns and rows will probably be different within a single block (i.e. a image block from a referee's shirt will have a large ρ within columns but a very small ρ within rows), the encoder can assume that ρ will be about the same between columns and rows over many blocks.

The second criterion stems from an assumption that DCT coefficients will satisfy a smoothness criterion; if a coefficient is zero, then there is a good chance that its neighbors are zero. The zigzag scan will then group these coefficients together, resulting in large runs of zeroes which can be coded efficiently with run-length coding.

From these two criteria, we can also derive a zigzag scan for coefficients in a 3-D array, or an N-D array in general:

1. If we calculate the sum $\sum_{l=0}^{L-1} i_l$ (where i_l is the index in dimension l) for each pixel in the array, we scan the pixels where $\sum_{l=0}^{L-1} i_l = 0$ (the DC coefficient), and then all the pixels where the $\sum_{l=0}^{L-1} i_l = 1$, and so on up to $\sum_{l=0}^{L-1} i_l = L(N-1)$ where N is the size of the block edge and L is the number of dimensions.

2. If i_{kl} is the lth dimensional index of the kth position in the scan, then $|i_{kl} - i_{(k+1)l}| \leq 1$ for all possible k and l.

A 3-D scan based on the two criterion above for a $4 \times 4 \times 4$ DCT transform is shown in Figure 3.2. The spatial "frequency" dimensions are labeled by i and j, and the temporal "frequency" dimension is labeled by k. The individual pixels are labeled by their corresponding order in the scan.

However, we may not necessarily be able to assume ρ will be about the same between different dimensions between blocks. More specifically, in the 3-D case, the ρ in the temporal

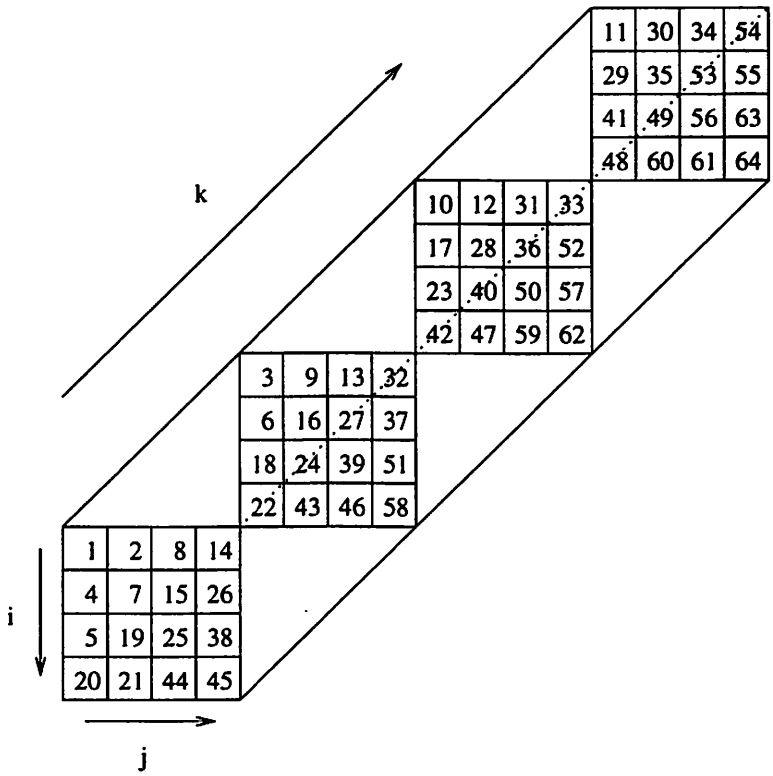


Figure 3.2: Illustration of zigzag scan for a $4 \times 4 \times 4$ array

direction depends on criteria such as the motion of the camera, the motion of objects being filmed, and whether or not frames are dropped in the compression process as a first step. At the very least, it cannot be assumed to be the same as the ρ in the spatial dimensions.

This brings up the question of what to do if the ρ is substantially different on average. The solution proposed here is to change the criterion where pixels are scanned in order of their "index sum" $i+j+k$ to a criterion where pixels are scanned in order of a "weighted index sum" $ai+aj+bk$. If ρ is larger in the temporal dimension, then $b > a$, if not, then $b \leq a$. In this case, however, we cannot have the second condition anymore, where every pixel in the scan is no more than one coordinate away in each direction.

The last section of this chapter describes simulations that were done with different scan orders.

3.7 Simulations

We simulated two scans, the "equally weighted" one shown in Figure 3.2 and another "modified" one where $a = 1$ and $b = 2$, with uniformly quantized DCT coefficients with a quantization factors of 5, i.e. the coefficients were floored to the nearest multiple of 5. The data came from the video-coding test sequences *Coast*, *Carphone*, and *MIT*. Within each sequence, a $144 \times 176 \times 4$ array of pixels (from four frames) was divided into 1584 $4 \times 4 \times 4$ 3D blocks, which were all transformed with a 3D DCT. We then had a total of 4752 blocks to work with.

For each location k in the scan, we tallied the total number of instances throughout the 4752 blocks where $p_k \neq 0$, where p_k is the pixel in location k in the scan. Ideally, this should monotonically decrease to zero in order for run-length/EOB coding to work well on the scanned coefficients.

In the first experiment (Figure 3.3), the 4 frames were from 30 frames/sec video sequences. In the second (Figure 3.4) experiment, the 4 frames were from 7.5 frames/sec video sequences. The purpose of this experiment was to see how the dropping of frames (a common step in video compression) affected the performance of the 3D-DCT.

As can be seen, the modified scan outperformed the even-weighted scan on the 30 fps sequence, taking advantage of the high temporal correlation between adjacent samples of the images, whereas both sequences demonstrated mediocre performance with the 7.5 sequence. This suggests that 3D DCT techniques should be more useful in high quality

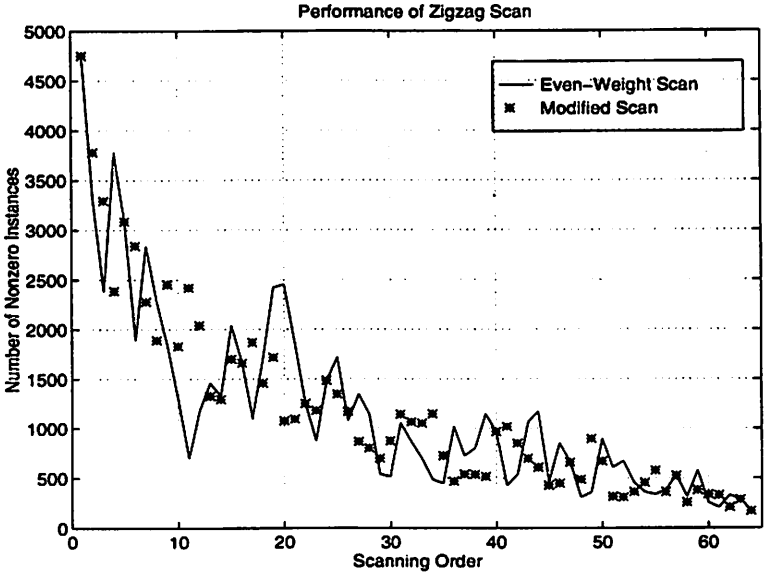


Figure 3.3: Performance of Zigzag scan on 30 fps sequences

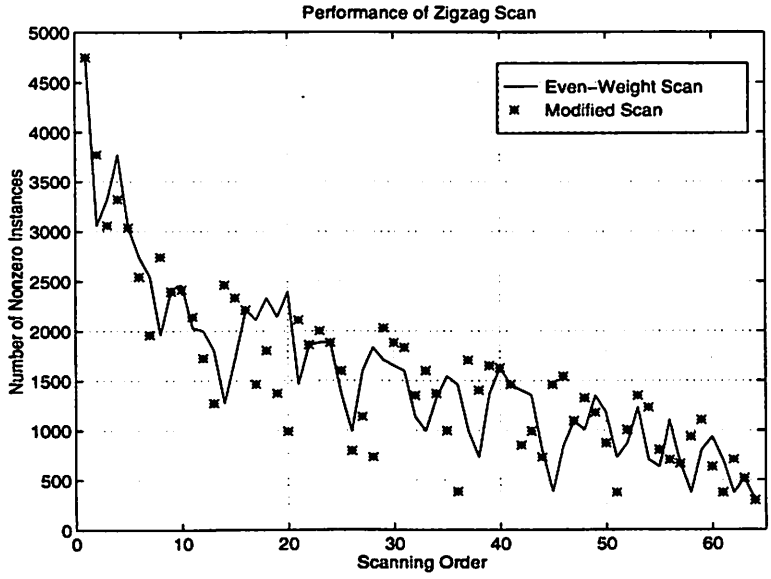


Figure 3.4: Performance of Zigzag scan on 7.5 fps sequences

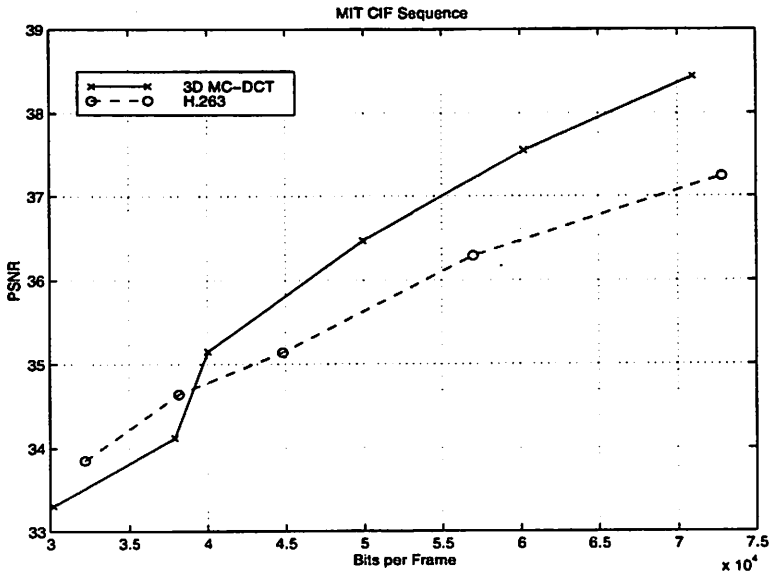


Figure 3.5: Performance of 3D MC-DCT coder compared with H.263. Different operating points were achieved by varying the quantizer step in both coders.

video communication systems with high framerates than in low-bitrate videoconferencing systems.

We also compared a 3D MC-DCT coder which used the motion estimation scheme of Chapter 2 to estimate motion vectors and the 3D DCT to code motion residual. With each coder, we coded the first eight frames of a CIF sized version of the *MIT* sequence. The 3D coder gradually outperforms H.263 as the PSNR of the coded result gets higher. The results are shown in Figure 3.5.

In the 3D MC-DCT coder, vectors were coded using the DPCM technique described in Section 5.4. The next chapter describes a technique which tries to improve on DPCM coding. At the time of this writing, the technique does not perform well enough in a full video coding system, but we have presented it in Chapter 5 in hope that a future version can be included in a 3D MC-DCT coder.

Chapter 4

Autocompensation

This chapter explains an algorithm that takes advantage of the temporal redundancy between motion vector fields to reduce the cost of sending motion information. Note that the algorithm is decoupled from the actual process of estimating motion from images, and can be used on vectors computed by any motion estimation scheme.

The following section will explain and illustrate our method of motion coding using a simple, three-frame image sequence. This report then discusses the implementation of the method, first discussing general implementations on MVF's and then discussing a specific implementation on "Block-Matching" based MVF's. Finally, the last section presents simulations and results.

4.1 Interframe coding of motion vectors

Consider three successive frames of a video sequence: I_1, I_2, and I_3. From this sequence, two motion fields can be defined; $\mathbf{V}_2$, which allows the decoder to compose I_2 from I_1, and $\mathbf{V}_3$, which allows the decoder to compose I_3 from I_2. Figure 4.1 shows the three underlying images in the first row, and quiver plots of the motion vector fields extracted from the three images in the second row.

Essentially, the sequence of motion vector fields has a few distinguishing differences with respect to the sequence of time-varying images. First, a two dimensional vector, namely x and y velocity components, is assigned to each block of pixels. Whereas an image is a two dimensional array of scalar values, a motion vector field is a two dimensional array of vector values. In video coding algorithms, motion vector fields are used to reconstruct images from

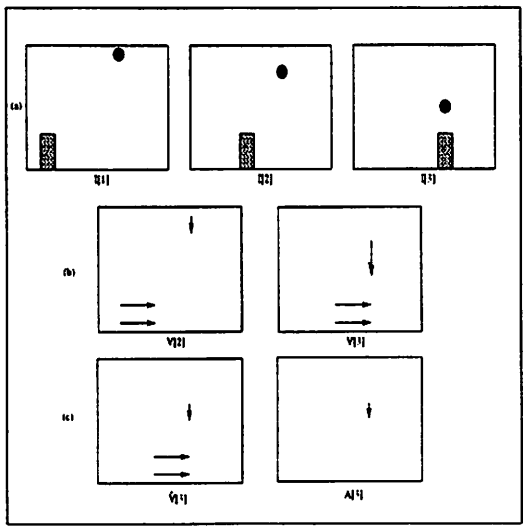


Figure 4.1: (a) Three hypothetical images of a video sequence. The rectangular object is moving at constant velocity, while the falling circle is accelerating. (b) The extracted motion sequence. (c) The compensated field $\hat{\mathbf{V}}_3$ and its difference from $\mathbf{V}_3$.

other images in the same video sequence. In that sense, a motion vector field can be seen as an *operation* or mapping, transforming an image I_n into another I_{n+1}, where n refers to an instance in time. In the same sense, a motion vector field can be applied to itself, transforming itself into another motion vector field.

An example can be given with pixel-accuracy motion vectors. If there is a vector $\langle 3, 4\rangle$ at pixel location $(1, 2)$, a guess could be made that in the next frame, pixel location $(1+3, 2+4) = (4, 6)$ has the vector $\langle 3, 4\rangle$.

To generalize, let $\mathbf{V}$ be a motion vector field where $\mathbf{V}(a, b)$ is the value of the motion vector field at location (a, b). Then one can find a new motion vector field through an operation called *Autocompensation*:

$$\hat{\mathbf{V}}_{n+1}(a+p, b+q) = \mathbf{V}_n(a, b) \tag{4.1}$$

where

$$\langle p, q\rangle = \mathbf{V}_n(a, b) \tag{4.2}$$

First, consider the case where the objects in the scene are moving with constant velocity (both the magnitude and the direction of the speed are constant). Then, the motion vector field at time $n+1$ can be found by autocompensating the field at time n using Equation 4.1. Similarly, by autocompensating the field at time $n+1$, one can find the field at time

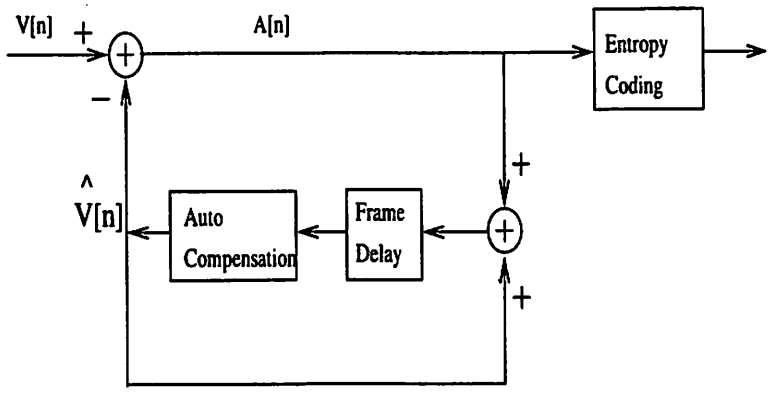


Figure 4.2: Block diagram of a whole "motion compensated" motion vector coding system.

$n+2$, etc. Therefore, the knowledge of $\mathbf{V}_0$ (the first motion vector field) is sufficient to generate the entire sequence.

However, in most cases, the velocity of objects is not constant, and a way must be found to account for this acceleration. Hence, let us now define

$$\mathbf{A}_n = \mathbf{V}_n - \hat{\mathbf{V}}_n \tag{4.3}$$

Note that $\mathbf{A}_n$ corresponds to the acceleration of the objects in the scene and also that knowing $\mathbf{V}_n$ and $\hat{\mathbf{V}}_n$ is sufficient to characterize $\mathbf{A}_n$. If we now assume that the direction of motion does not change with time and only the magnitude of motion is time-varying, then the direction of each element of $\mathbf{A}_n$ is the same as that of the corresponding motion vector in $\mathbf{V}_n$ and hence a scalar component would be sufficient to describe each component of $\mathbf{A}_n$. In general, however, both the magnitude and the direction of the velocity change with time and a vector component is necessary to characterize the acceleration.

The third row of Figure 4.1 shows the field $\hat{\mathbf{V}}_3$ extracted from $\mathbf{V}_2$ and also the acceleration field $\mathbf{A}_3$ resulting from subtracting $\hat{\mathbf{V}}_3$ from $\mathbf{V}_3$. As can be seen, the vector field $\mathbf{A}_3$ will be much easier to code than $\mathbf{V}_3$, while providing just as much information to the receiver. Figure 4.2 shows a block diagram of the entire vector coding system.

4.2 First Implementation of Method

Uncovered boundaries, uncovered objects, and fading have presented problems in the whole area of motion estimation from images. In turn, motion estimation of motion vectors presents even more problems. In the following paragraph, this paper explains two major

problems of motion estimation: ill-posedness, and vector location of non-integer valued vectors.

In luminance-based motion estimation from images, a portion of an image I_{n+1} may not be found in I_n, causing an ill-posed problem in the estimation. When estimating a subsequent motion field with 4.1, similar ill-posed problems can occur. For example, if two or more motion vectors in a motion field $\mathbf{V}_n$ point to the same location (a, b), then there is a conflict as to which vector will be represented in $\hat{\mathbf{V}}_{n+1}$. In the same way, there will be a problem if all of the vectors in the field point away from a location (a, b) Therefore, there will be no motion vector representing (a, b) in the subsequent field.

A problem related to and even more important than the above is vector location. In order to be useful, vectors in MVF's must correspond to certain pixel or block locations. Equation 4.1 does not guarantee this condition. Therefore, a re-interpolation must be done around pixel locations to compute the new MVF. A weighted average of vectors surrounding the location is used; this equation summarizes the method:

$$\hat{\mathbf{V}}_{n+1}(i,j) = (\sum_a \frac{1}{d_a^2})^{-1} \sum_a \frac{1}{d_a^2} \mathbf{v}_a \tag{4.4}$$

where $\mathbf{v}_a$'s are all of the vectors in the field $\mathbf{V}_n(a+p, b+q)$ which are located within a square search area centered around location (i, j), and the d_a's used for the weighing averages correspond to the distance of these vectors from location (i, j). Note that this method gets around the case of no vectors being located at (i, j), however, when a vector is actually located at (i, j), it takes precedence over all other vectors. When more than one vector is located at (i, j), our algorithm chooses one at random.

In [20], MVF's were coded using this algorithm. This thesis goes one step further and estimates its performance in a full video coding system where the bitrate of both motion vectors and motion residual is considered. The next section details further modifications made to this algorithm for coding block-estimated motion vectors.

4.3 Modifications

One aspect of motion estimation not covered by the above section is the possibility that a motion vector might not "exist" in a particular location; the introduction of Chapter 5 discusses this in greater detail. In this case, a NULL vector is encoded, telling the

encoder to directly encode the image data in the corresponding block instead of trying to approximate with data from the previous image. We choose to ignore these NULL vectors when performing the calculation in 4.4; accordingly, we also encode a NULL vector when no valid (non-NULL) vectors are found within a certain distance from location (i, j).

After Autocompensation we perform the following evaluation on every valid vector in $\hat{\mathbf{V}}_{n+1}(i, j)$. We calculate and store the distance metric (explained in Chapter 2) e_a of the motion residual resulting from the vector. We then find the distance metric e_o of the optimal vector. If $e_a - e_o < P$, where P is some factor by which we "prefer" the autocompensated vector, we use the autocompensated vector and set $\mathbf{A}_n(i, j)$ to 0. Otherwise, we use the optimal vector and set $\mathbf{A}_n(i, j)$ to be the difference between the optimal vector and the autocompensated vector. In this way, the decoder can deduce using $\mathbf{A}_n$ the MVF used by the encoder.

Figure 4.3 shows a flowchart which summarizes the above method for coding vectors in $\mathbf{A}_n$.

4.4 Simulations and Results

We used this method to code eight frames of test sequences *Mother and Daughter* and *Coast*. First an initial frame (the "zeroth" frame) is DCT encoded and transmitted. Then, GMC is performed on the next four frames. Motion estimation using the method in Chapter 2 is performed for the first frame, then the distance metric in Chapter 2 is used in conjunction with autocompensation to estimate the second, third, and fourth frames. The residual of the four frames is encoded with a 3D-DCT as explained in Chapter 3 and then transmitted. The cycle then repeats for the next four frames.

To illustrate further, Figure 4.4 shows four MVF's from a standard motion-compensated coder, and Figure 4.4 shows four MVF's from a coder using autocompensation. In the autocompensated coder, the last three MVF's are $\mathbf{A}_n$ fields which require much less bits to transmit than the corresponding $\mathbf{V}_n$ frames in the standard coder.

The total bitrate of the autocompensated coder is compared to the total bitrate of another coder which uses standard motion compensation, but also gathers residual over four frames and encodes it with a 3D-DCT. In both cases, the total bitrate is approximated by

$$R_{\text{vectors}} + R_{\text{residual}} + R_{\text{intra}}$$

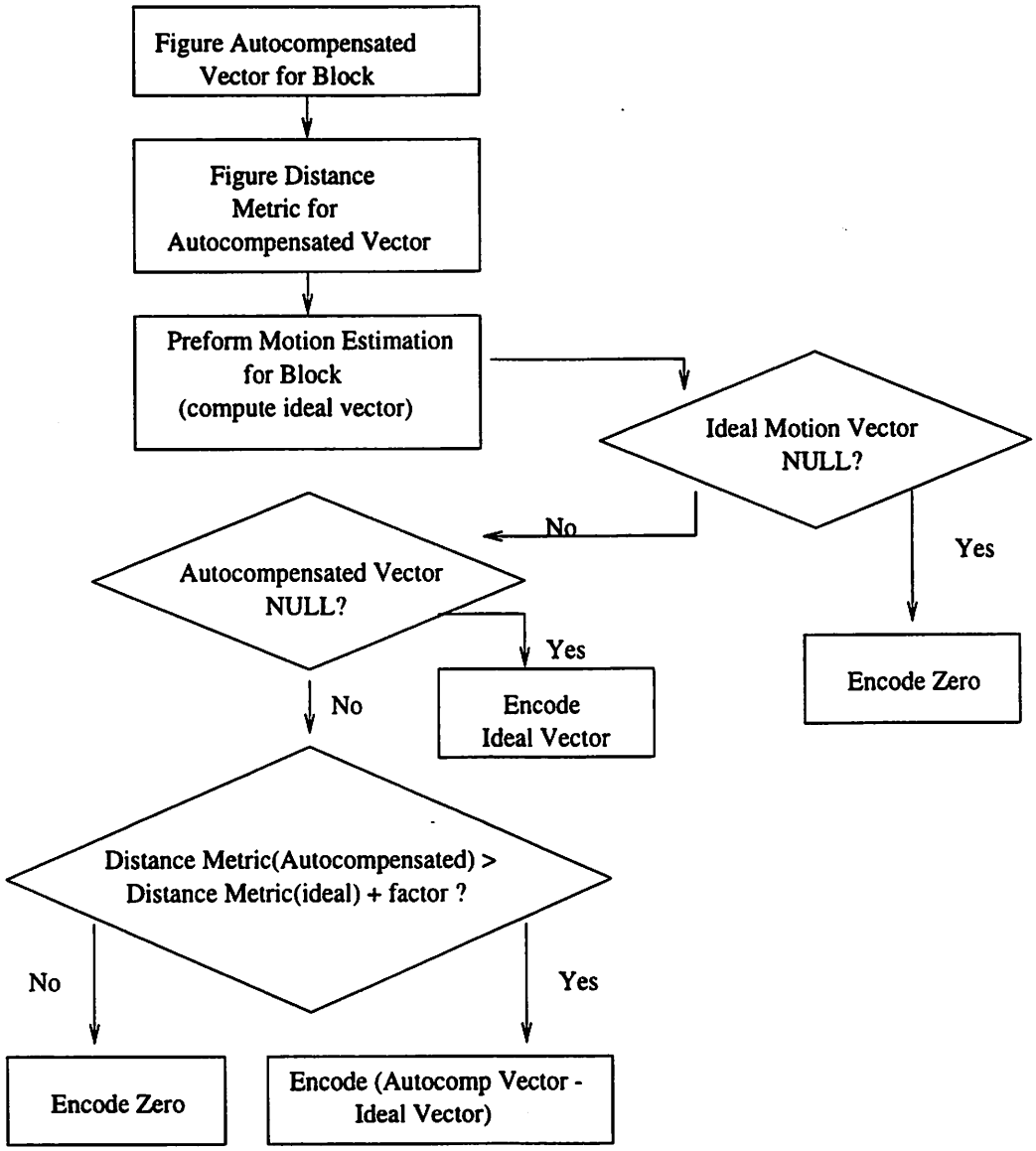


Figure 4.3: Flowchart for encoding each vector in $\mathbf{A}_n$. f the ideal MV is null, then that means there is no proper approximation in the previous image, and fresh image data has to be sent.

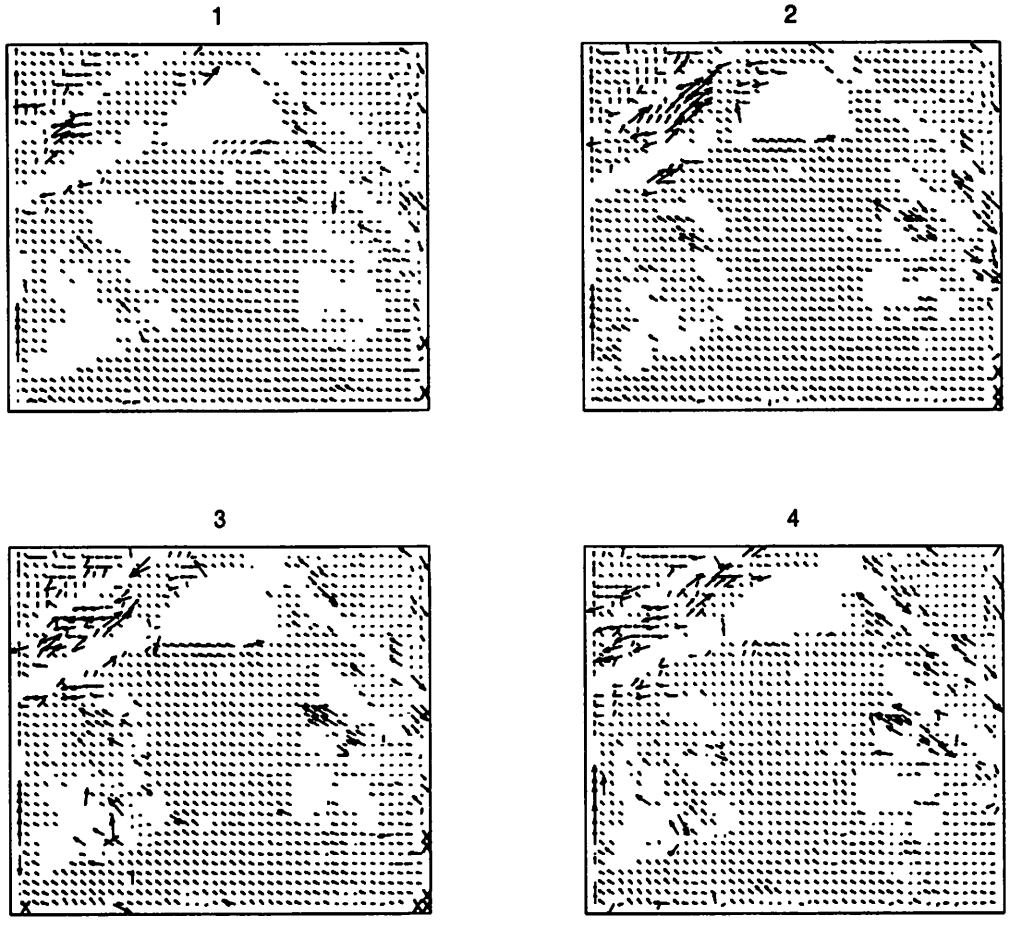


Figure 4.4: The first four MVF's of the sequence *Foreman* estimated from a conventional motion estimation algorithm.

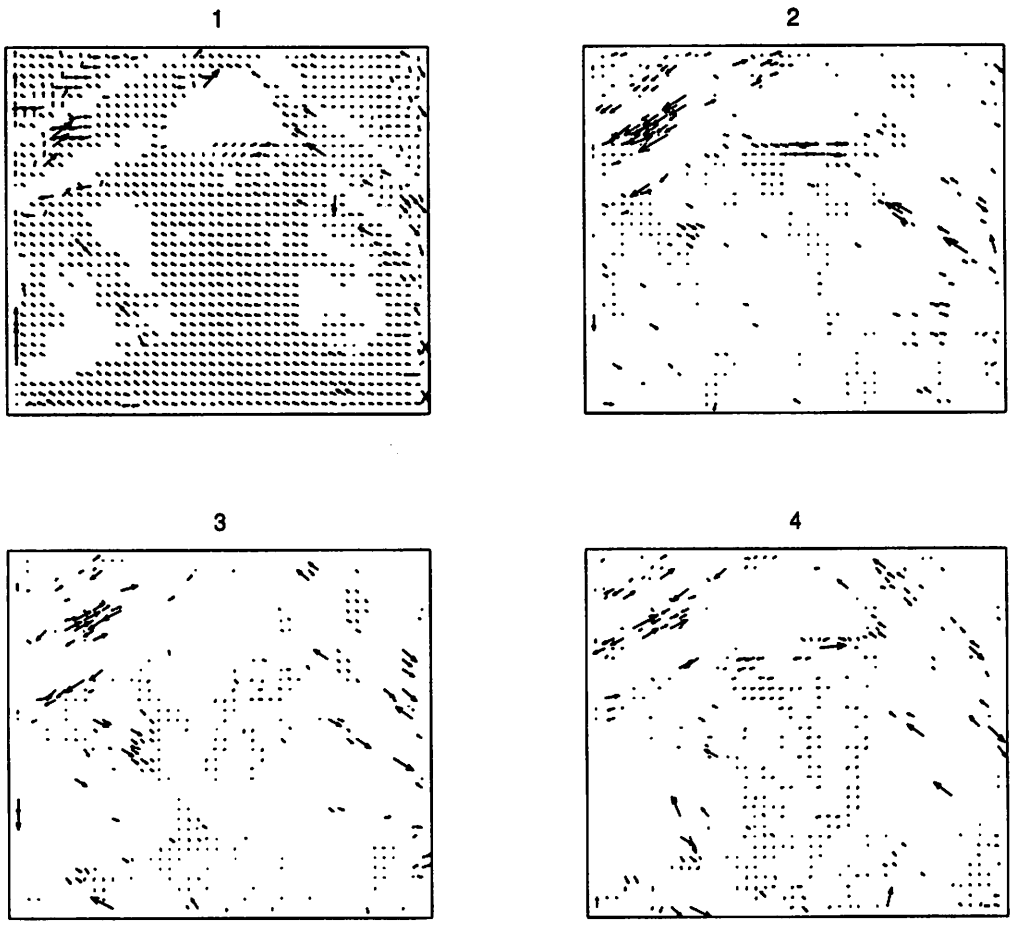


Figure 4.5: The first four MVF's of the sequence *Foreman* estimated with autocompensation. They represent $\mathbf{V}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$, and $\mathbf{A}_4$.

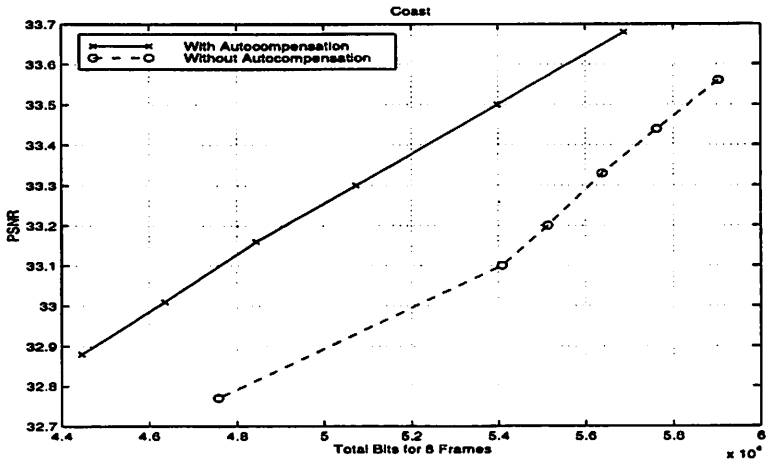
where *intra* refers to "uncompensable" information which couldn't be approximated by motion vectors. The motion vector fields (eight $\mathbf{V}_n$'s without autocompensation, two $\mathbf{V}_n$'s and six $\mathbf{A}_n$'s with autocompensation) are scanned with the DPCM method in Section 5.4 and then the entropy of the vector differences is scaled by the number of vectors to estimate R_{vectors}. The 3D-DCT residual coefficients and the DCT intra coefficients are zigzag scanned and run-length encoded. The entropy of runs and lengths is then scaled to estimate R_{residual} and R_{intra}. Although an actual encoder would use Huffman tables to encode the above data, the above method should give a fair comparison on a case by case basis.

Different operating points were achieved by varying the quantizer step used on the DCT residual coefficients. For a given quantizer step, the bitrate spent on vectors would always be lower for the autocompensation coder, but the bits spent on residual would be higher since the autocompensated vectors could not predict as well as the normal vectors.

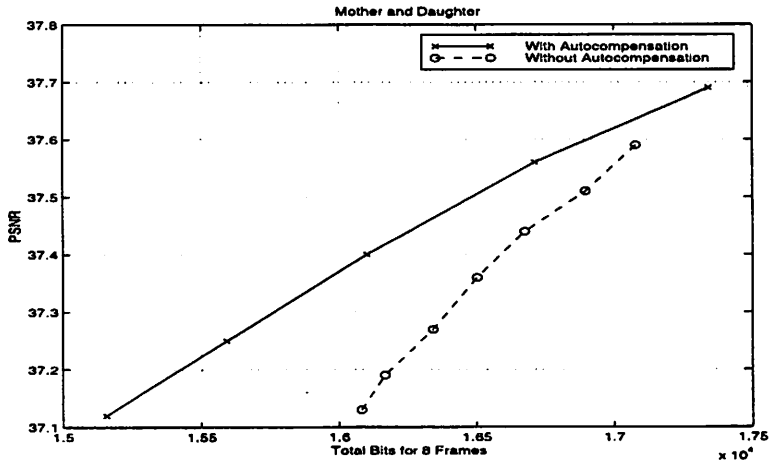
The results are shown in Figure 4.6. In the sequence *Coast* autocompensation almost introduces a constant gain over normal motion estimation. In the *Mother Daughter* sequence we see that the gain introduced by autocompensation "closes up" as the picture quality gets better. Autocompensation worked well on these two particular sequences because their motion is relatively simple. In other sequences with more complicated motion such as the *MIT* sequence used later in this thesis, the algorithm does not fare so well. The vector bitrate is always reduced, but the residual bitrate is increased too much.

4.5 Conclusion

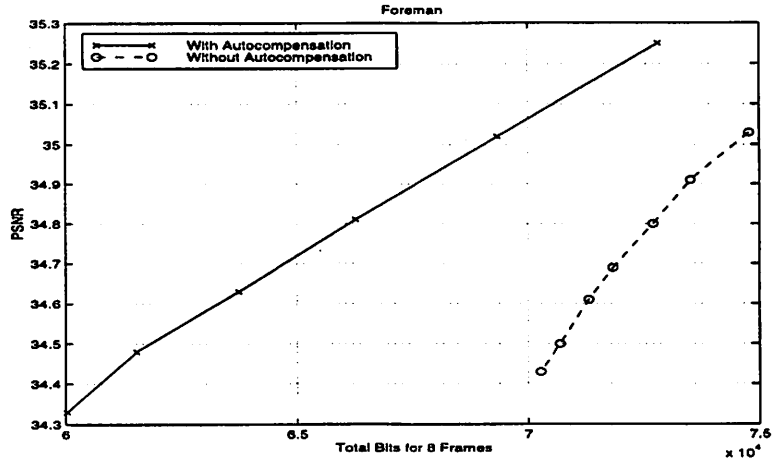
Autocompensation of motion vectors is a new approach to coding motion for very low bitrate applications. Taking advantage of both the temporal redundancy and the "mapping" property of motion vectors, this method predicts a subsequent motion vector field from a previous vector field. This predicted motion vector field is then used in turn to predict image data. A simple version of this method based on interpolation has been presented which shows promise, working well on sequences with mostly translational motion as well as "teleconferencing" sequences. With more experimentation and training, autocompensation can be adapted further and is expected to perform better on more complicated sequences.



(a)



(b)



(c)

Figure 4.6: (a) Coding results for the sequence *Coast*. (b) Coding results for the sequence *Mother and Daughter*. (c) Coding results for the sequence *Foreman*.

Chapter 5

3D Zerotree Coding of Motion Vector Fields

DPCM was used in the simulations at the ends of Chapter 4 and Chapter 3 to encode motion vector fields (MVF's). This chapter describes an alternative method to coding motion vectors. More specifically, we use three-dimensional zerotree coding after a Haar Transform. The coding method is an extension of Shapiro's zerotree method of coding image data. The following section discusses extending image coding methods to vector fields, and the rest of the chapter discusses our implementation of zerotree coding.

5.1 Differences between Vector and Image Data

People have tried to look at MVF's as images, and use image-coding methods to compress motion vector data. They have met with limited success, most likely due to some fundamental differences between motion vector data and image data.

At first glance, motion vector data seems similar to image data because there are often large areas that have exactly or approximately the same value, since they correspond to a large object moving at a constant speed. For this reason, Nickel and Husoy [11] and others have tried to use techniques such as the DCT to encode motion vector fields (MVF's).

However, there are important differences between motion vector and image data that must be addressed in order for such coding techniques to work well. First, if the coding is lossy, such as in a DCT-based system, the coder must take into account how the encoding error in motion vectors will affect the coding of motion residual data. Chen, Villasenor and

Park [3] have looked at this problem through posing the estimation and coding of motion vectors as a rate-distortion problem. Second, whereas an image will have a defined value for every pixel, a block-estimated MVF will not necessarily have a motion vector for every block. For example, when a new object appears in a video frame, no motion vectors can describe it from the previous frame. Filling in zeros may not be a good solution because most transform coders will perform less well when the input data has a lot of singularities.

Although our scheme is transform based, we address the two issues above by first coding the vector data losslessly and then by having an efficient method of treating blocks where motion vectors are not found. We first describe our transform in Section 5.2, and then describe the coding of the transform coefficients in Section 5.3.

5.2 Implementation of Haar Transform

The Haar Transform is essentially the simplest non-trivial subband decomposition possible. It can be used to transform a string of N numbers into two strings of $N/2$ numbers, representing a highpass and lowpass decomposition of the signal. In this way, the Haar Transform can be defined as

$$H_0[n] = h[2n] + h[2n+1]$$

$$H_1[n] = h[2n] - h[2n+1]$$

with inverse transform

$$h[n] = \frac{1}{2}\left(H_0[\frac{n}{2}] + H_1[\frac{n}{2}]\right) \text{ for } n \text{ even}$$

$$h[n] = \frac{1}{2}\left(H_0[\frac{n-1}{2}] - H_1[\frac{n-1}{2}]\right) \text{ for } n \text{ odd}$$

where $H_0[n]$ is the low-frequency portion of $h[n]$ and $H_1[n]$ is the high-frequency portion. The Haar Transform can also be used in many dimensions when applied separately in each dimension.

The above transform can also be seen as putting $h[n]$ through both a lowpass and a highpass filter in parallel and then subsampling the outputs by 2, keeping the total number of samples constant. In *Octave-band* filter banks, the subsampled lowpass output is again put through a lowpass and highpass filter, and each output is subsampled by 2. This happens K times, where 2^K is the largest power of 2 which can divide into the number of

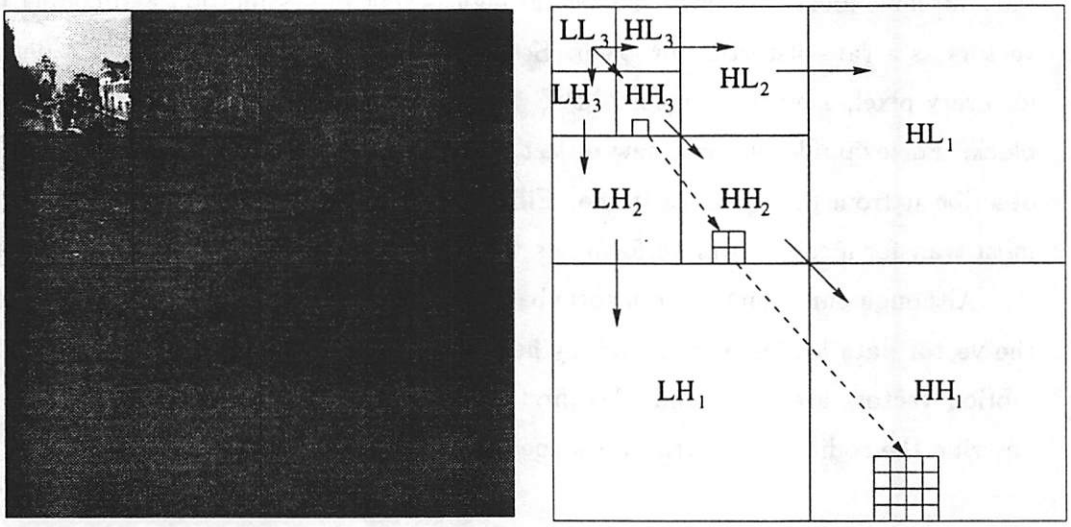


Figure 5.1: (a) A 512 × 512 image passed through two stages of octave band decomposition. (Values have been normalized and adjusted for display purposes) (b) An illustration of a possible zerotree in two dimensions.

samples N. Figure 5.1 (a) shows an image which has gone through two iterations in both the vertical and horizontal dimensions.

When we take these sums and differences, however, we run into the danger of creating numbers of larger magnitude which will be even harder to code. To explain, if two signed integers A and B can be represented with N bits in two's-complement notation, then $A+B$ and $A-B$ will each need $N+1$ bits. This will result in one more pass needed by our coding algorithm. However, we can economize by noting that $A+B$ and $A-B$ are either both odd or both even. Therefore, we can "throw away" the LSB of $A+B$ or $A-B$ since it holds redundant information. We choose to throw away the least significant bit of $A-B$ (the high frequency component). We then have a "Modified Haar Transform" equivalent to the "Sequential" transform of [14], shown here:

$$H_0[n] = h[2n] + h[2n+1] \tag{5.1}$$

$$H_1[n] = \lfloor \frac{1}{2}(h[2n] - h[2n+1]) \rfloor \tag{5.2}$$

with inverse transform

$$h[n] = \left\lceil \frac{1}{2} H_0[\frac{n}{2}] \right\rceil + H_1[\frac{n}{2}] \text{ for } n \text{ even} \tag{5.3}$$

$$h[n] = \left\lfloor \frac{1}{2} H_0[\frac{n}{2}] \right\rfloor - H_1[\frac{n}{2}] \text{ for } n \text{ odd} \tag{5.4}$$

When the motion estimator cannot find an appropriate motion vector for a block, it records a DONTCARE value in its place. When either $h[2n]$ or $h[2n+1]$ is a DONTCARE, we will transform according to the following rules:

$$H_0[n] = 2h[2n] \text{ if } h[2n+1] \text{ is DONTCARE}$$

$$H_0[n] = 2h[2n+1] \text{ if } h[2n] \text{ is DONTCARE}$$

$$H_0[n] = \text{ DONTCARE if both are}$$

$$H_1[n] = 0$$

The DONTCARE in the third equation is then propagated to the next octave-band decomposition, or a transform in another dimension.

Using the decoding rules in (5.3) and (5.4), all vectors which are not DONTCARE's will be decoded correctly, and those which are DONTCARE's will be decoded with the value of their closest neighbor which is not DONTCARE. However, this will cause no problem because the motion vector will be ignored in the decoding algorithm.

5.3 Zerotree Algorithm

Our coder uses 4 × 4 sized macroblocks on 288 × 352 sized images (CIF size). Over 8 images, we then have a 72 × 88 × 8 array of motion vectors, which is put through three stages in octave-band decomposition in each of three dimensions. Figure 5.2 shows how the coefficients are arranged in the final array before encoding.

Our coder then takes advantage of structures called *zerotrees* to code efficiently. This concept was first presented by Shapiro [15] in 1993. This concept is illustrated in Figure 5.1 (b) for two dimensions, and in Figure 5.2 for three dimensions.

First the data is organized into "trees," with (using Figure 5.1 to explain) "parent" nodes in bands LH_{n+1}, HH_{n+1}, and HL_{n+1}, and "children" nodes in bands LH_n, HH_n, and HL_n, respectively. Depending on how many stages there are in the octave-band decomposition, the "children" can have "children" of their own, and so on. In an N-dimensional array,

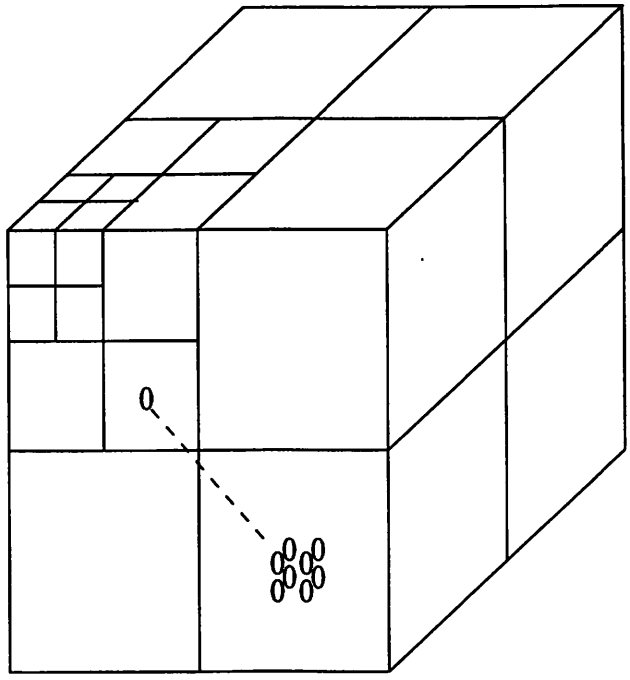


Figure 5.2: Arrangement of coefficients in a three dimensional transformed array. The lowest frequency band is shown in the front top left.

there are 2^N children for each parent, located at the same spatial location corresponding to the parent.

A zerotree-based coder then assumes that the (normalized) magnitude of a parent will more likely be greater than that of its children than smaller. Therefore, if we find that the magnitude of a parent node is low, then it is likely that the magnitude of its corresponding children will be low. Thus, we can take advantage of zerotrees, illustrated by the boxes in Figure 5.1 and the zeroes in Figure 5.2. Zerotrees occur when a parent's magnitude along with the magnitude of *all* its descendants are less than a particular coding threshold.

Our coding method is similar to [15] in that we use four symbols *POS* (positive), *NEG* (negative), *IZ* (isolated zero) and *ZT* (zerotree). It is somewhat simpler because we are performing lossless coding; more specifically we do not use a subordinate pass. The general procedure is outlined below:

1. The coefficients in LLL_N (the lowest frequency band) are directly encoded (without using any entropy-coding technique).

2. Each coefficient in LLH_N, LHL_N, LHH_N, ... HHH_N is processed by a recursive

coding procedure, described below, which will encode it along with all its descendants.

The vectors have a range of -15 to 15 (they are measured in half-pixels), and so they would need 5 bits each for encoding without any processing. Every time an octave-band decomposition is done, the magnitude in the low frequency band increases by a factor of 2, and one more bit is needed. Therefore, $14 = 5 + 3 \times 3$ (3 decompositions in 3 dimensions) bits will be needed for each coefficient in the lowest frequency band. Although this may seem wasteful at first, it is well amortized in the rest of the coding algorithm.

For the recursive coding procedure, we successively run through each coefficient in LLH_N, LHL_N, LHH_N, ... HHH_N and go through the following procedure:

1. We set an initial threshold T_i to 2^M, where M is 2 less than the number of bits that would be needed to code the coefficients in a particular band directly. For example, $M = 11$ for bands LLH_N, LHL_N, and HLL_N, $M = 10$ for bands HHL_N, HLH_N, and LHH_N, and $M = 9$ for band HHH_N.

2. We label the coefficient x of the root node as follows: they are labeled POS if $x \geq 2^M$, NEG is $x \leq -2^M$, and ZERO otherwise.

3. We then decrease the threshold to $2^{-3}T_i$ and then label all the children of the node according to this threshold. When a child is labeled, it's own children are also labeled with threshold $2^{-6}T_i$.

4. Then if the node and all descendants are ZERO, we code a ZEROTREE, otherwise we code the node label and then recursively code the descendants by checking if they are roots of zerotrees or not.

5. After coding is done, all nodes labeled POS have the threshold subtracted from them, and all nodes labeled NEG have the threshold added to them.

6. Coding starts again with a threshold of 2^{M-1}, and steps 2 through 5 follow again in a loop. After a node is labeled with threshold $2^0 = 1$, it will be coded once more to see if what remains is -1, 0 or 1.

7. After coding is done, the stream of symbols is entropy coded.

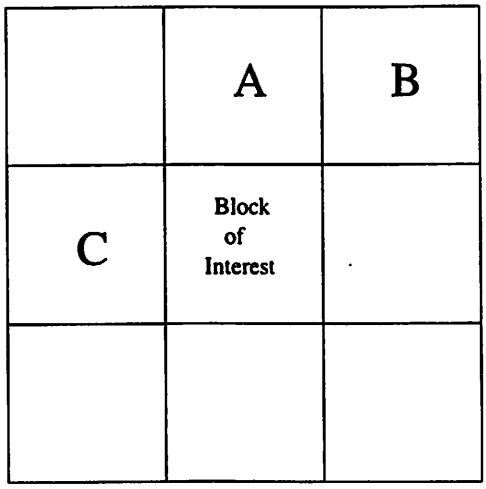


Figure 5.3: Illustration of the method used in H.263 to code motion vectors. The median is taken from the motion vectors of blocks A, B, and C in both the x and y directions to compute a predictor for the motion vector in the block of interest. The difference between the predicted vector and the actual vector is then coded.

Sequence	Zerotree Bits per MVF	DPCM Bits per MVF
Hall Monitor	21766	16981
MIT	16960	17368
Container	2571	12861

Table 5.1: Simulation Results for Coding Motion Vector Fields from CIF Sequences

5.4 Simulations and Conclusions

We tested our algorithm on the motion vectors estimated with the scheme in Chapter 2 on the video coding test sequences *Hall Monitor, Container Ship* and *MIT*. (We skip through the first 40 frames on *Hall Monitor* because there is no motion, only image noise.) We compared our method of coding with the DPCM technique currently used in H.263 to code motion vectors illustrated in Figure 5.3 [5].

For comparison, we entropy coded the symbols from the zerotree method and we entropy coded the differences from the DPCM method. We then compared the total bitrate in both cases. The results are shown in Table 5.1.

The zerotree method under-performs relative to DPCM for the *Hall Monitor* sequence,

where there is a large amount of noise in the underlying images. It achieves somewhat better results on the *MIT* sequence, a sequence with complex motion where many objects are moving around in different directions. It far outperforms DPCM on the *Container* sequence, a sequence with slight motion where most of the vectors are zero.

Ideally, a video coding algorithm would be able to switch (or the user could set it as a parameter) between 3-D zerotree coding of motion vectors or DPCM or some other 2-D method of coding motion vectors. More attention could be given to coding the motion vectors in a lossy context, or using another wavelet with better frequency separation properties than the Haar wavelet.

Chapter 6

Conclusion and Future Work

In Chapters 2 and 3 the thesis discussed the advantages of overlapped block motion estimation and the 3D DCT. These two algorithms are currently part of the general structure of the proposed 3D MC-DCT coder. Further exploration into reducing the amount of unnecessary computation in OBME is needed in order for the coder to work in real time. Further exploration is also needed to see if other 3D transforms might work better than the 3D DCT for coding residual. Finally, the overall structure of the 3D MC-DCT coder could be adapted for backwards and bidirectional motion estimation and compensation, which has proved beneficial in video coding standards.

This thesis proposed two approaches to reducing temporal redundancy in MVF's for bitrate reduction. First, it defined the "Autocompensation" algorithm used to predict a subsequent motion field from a previous one. Second, it proposed the method of removing temporal redundancy in the last chapter, which was to take a group of MVF's over time and then code them with a three dimensional adaptation of Shapiro's zerotree algorithm. The following two paragraphs detail the advantages of each algorithm and suggest future directions to pursue.

The Autocompensation algorithm shows promise in reducing the bitrate consumed by motion information and the total bitrate in general. For a given SNR, the bitrate it saves on vectors outweighs the added residual bitrate. Besides conserving bandwidth, Autocompensation could also help in other areas of research. Because it takes advantage of the consistency of motion between frames, Autocompensation merits more investigation into its application to image sequence interpolation and in computer vision problems involving motion.

Although the zerotree algorithm shows promise in coding motion vector fields estimated from fresh, uncorrupted raw images, the algorithm at present is less effective at coding motion vector fields in a full video coding scheme. Other schemes of "smoothing" MVF's besides OBME need to be considered in conjunction with the zerotree algorithm. The promising performance of the Autocompensation algorithm demonstrates that there is significant advantage to reducing temporal redundancy of MVF's in video coding, and further exploration is needed to discover the best way to reduce this redundancy in zerotree coding.

Bibliography

[1] C. Auyeung, J. Kosmach, M. Orchard, and T. Kalafatis, "Overlapped Block Motion Compensation", *SPIE Conf. Visual Communication Image Processing*, vol. 1605, pp. 561–571, November 1992.

[2] Y.L. Chan and W.C. Siu, "Variable Temporal-Length 3-D Discrete Cosine Transform Coding", *IEEE Trans. on Image Processing*, vol. 6, no. 5, pp. 758–763, May 1997.

[3] F. Chen, J.D. Villasenor, and D.S. Park, "A Low Complexity Rate- Distortion Model for Motion Estimation in H.263", *Proc. of IEEE International Conference on Image Processing*, 1996.

[4] K. Illgner and F. Müller, "Hierarchical Coding of Motion Vector Fields", *Proc. of IEEE International Conference on Image Processing*, vol. 1, pp. 566–569, October 1995.

[5] ITU-T/SG15/LBC Special Rapporteur for Very Low Bitrate Visual Telephony, "Draft Recommendation H.26P (Video coding for telecommunication channels at < 64 kbits/s)," PTT Netherlands, 1995. LBC-95-027.

[6] A. Jain. *Fundamentals of Digital Image Processing*, Prentice Hall, 1989.

[7] N.S. Jayant and Peter Noll, *Digital Coding of Waveforms*, Prentice Hall, 1984.

[8] G. Karlsson and M. Vetterli, "Three Dimensional Sub-band Coding of Video", *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, pp. 1100–1103, April 1988.

[9] Y.Y. Lee and J. W. Woods, "Motion Vector Quantization for Video Coding", *IEEE Trans. on Image Processing*, March 1995.

[10] A. Leon-Garcia, *Probability and Random Processes for Electrical Engineering*, p. 265, Addison Wesley, 1993.

[11] M. Nickel and J.H. Husoy, "A hybrid coder for image sequences using detailed motion estimates", *Proc. of the SPIE*, vol. 2501, pt. 1, pp. 963–71, 1991.

[12] J.-R. Ohm, "Three-Dimensional Subband Coding with Motion Compensation", *IEEE Trans. on Image Processing*, vol. 3, pp. 559–571, September 1994.

[13] A. Oppenheim and R. Schafer, *Discrete-Time Signal Processing*, Chapter 9, Prentice Hall, 1989.

[14] A. Said and W.A. Perlman, "An image multiresolution representation for lossless and lossy image compression" , *IEEE Trans. on Image Processing*, vol. 6, pp. 1303–1310, June 1996.

[15] J.M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients", *IEEE Trans. on Signal Processing*, vol. 41, no. 12, December 1993.

[16] D. Taubman and A. Zakhor, "Multirate 3-D Subband Coding of Video", *IEEE Trans. on Image Processing*, vol. 3, pp. 572–588, September 1994.

[17] R. Vargas. *Fast Algorithms Using MSE for MPEG Motion Estimation*, Masters dissertation, U.C. Berkeley, May 1996.

[18] M. Vetterli and J. Kovačević, *Wavelets and Subband Coding*, Prentice Hall, 1995.

[19] J.Y.A. Wang and E.H. Adelson, "Representing Moving Images with Layers", *IEEE Trans. on Image Processing*, vol. 3, no. 5, pp. 233–242, September 1994.

[20] J. Yeh, M. Khansari, and M. Vetterli, "Motion Compensation of Motion Vectors", *Proc. of IEEE International Conference on Image Processing*, vol. 1, pp. 574–577, October 1995.