Explicit Bit Minimization
for Motion-Compensated Video Coding

(extended abstract)

Dzung T. Hoang* Philip M. Long† Jeffrey Scott Vitter‡

Department of Computer Science
Duke University
Box 90129
Durham, NC 27708–0129

Abstract

We compare methods for choosing motion vectors for motion-compensated video compression. Our primary focus is on videophone and videoconferencing applications, where very low bit rates are necessary, where the motion is usually limited, and where the frames must be coded in the order they are generated. We provide evidence, using established benchmark videos of this type, that choosing motion vectors to minimize codelength subject to (implicit) constraints on quality yields substantially better rate-distortion tradeoffs than minimizing notions of prediction error. We illustrate this point using an algorithm within the $p \times 64$ standard. We show that using quadtrees to code the motion vectors in conjunction with explicit codelength minimization yields further improvement. We describe a dynamic-programming algorithm for choosing a quadtree to minimize the codelength.

1 Introduction

The typically strong correlation between successive frames of a video sequence makes video highly compressible, since the pixels of the previous frame can be used to predict the intensities of the current frame. The difference between the predicted and the true frame often is small and can be encoded efficiently, for example, by a lossy transform coder using the two-dimensional discrete cosine transform (2D DCT).

Improved compression is readily obtained by first estimating what portions of the current frame correspond to moving objects and then transmitting *motion vectors*

*Affiliated with Brown University. Supported in part by an NSF Graduate Fellowship and by Air Force Office of Strategic Research grant F49620–92–J–0515.

†Support was provided in part by Air Force Office of Strategic Research grant F49620–92–J–0515.

‡Support was provided in part by Air Force Office of Strategic Research grant F49620–92–J–0515, Army Research Office grant DAAH04–93–G–0076, and by an associate membership in CESDIS.

that tell the decoder where to look on the previous frame for predictions of the intensity of each pixel in the current frame. The most popular method for estimating these motion vectors originated with Jain and Jain [8] and is called block matching. In their approach, the current frame is divided into blocks (usually 8×8) whose pixels are assigned the same motion vector. (Carpentieri and Storer [2] group together blocks with the same motion vector into *superblocks*, and describe a method for keeping track of which block is in which superblock, in order to reduce the encoding of the motion vectors.) Jain and Jain's approach is taken by the CCITT in Recommendation H.261 (also known as the $p \times 64$ standard) [3, 10]. The motion vector for a given block B is usually obtained by (approximately) minimizing, from among candidates $\vec{v}$ within a limited search area, some norm of the difference between B and the prediction obtained from $\vec{v}$. The mean squared error (which is the square of the ℓ_2-norm) is a commonly used measure, although for example the mean absolute difference is often substituted because it can be implemented efficiently. This is done, for example, in the implementation of the $p \times 64$ standard made available by the Portable Video Research Group (PVRG) [6].

In this paper, we report on work in progress investigating the use of heuristics that more directly estimate the effect of the choice of a given motion vector on the total codelength. Our experimental results give evidence that this approach yields substantially better rate-distortion tradeoffs. While the error-minimization approach enables one to separate into modules the tasks of predicting the current frame and coding the resulting error, our experiments suggest that this separation comes at a substantial cost: integrating the two can lead to a better coder. We illustrate this point using three programs. The first two implement this idea within the $p \times 64$ standard, and the third adds the idea of using a quadtree to transmit the motion vectors to exploit spatial redundancy in the estimated motion field.

In this paper, our emphasis is on codelength and quality, not on computation time, in order to determine the limits on the compressibility of video. We can initially use computationally intensive coders to set a standard against which more efficient algorithms can be judged, such as those obtained, for example, by modifying the computationally intensive coders. Furthermore, our algorithms are highly parallelizable, and special-purpose chips are already available for many subroutines used by our algorithm.

The $p \times 64$ standard is intended for applications like videophone and video conferencing, where very low bit rates are required, not much motion is present, and frames are to be transmitted essentially as they are generated. Unlike the case of the MPEG standard, we cannot first compress a subsampling of frames, and then use frames both before and after a given frame to predict it. Our experimental results are for benchmark videos typical of the type for which the $p \times 64$ standard was intended: they consist of a single speaker sitting at a table.

Using the block-matching approach, we create only a crude, but concise, model of the motion. For video coding, we do not necessarily want to find the "correct" motion vectors, in contrast to a goal of research in optic flow, for example [13]. If a motion vector field that does not correspond to the actual motion in the scene yields the shortest description, that is sufficient for purposes of compression. However, an accurate motion field is desirable for motion interpolation, where a non-coded frame is interpolated from two successive coded frames by performing motion compensation using an interpolated motion field. We plan to apply our techniques to the compression-related optic flow techniques of [12], which should result in improved motion estimates.

In the next section, we describe the PVRG implementation of the $p \times 64$ standard, and then show how to modify the PVRG implementation, but remain within the $p \times 64$ standard, to choose motion vectors that more directly minimize codelength. For comparable quality, at the level roughly required for transmitting 15 CIF frames at 128 Kbits/sec, explicit bit minimization reduces the codelength by about 17% on average for a particular benchmark video. In the $p \times 64$ standard, two binary decisions must be made from time to time (for details, see Section 2). In the PVRG implementation, heuristics based on prediction error are used to make these decisions. When the explicit bit minimization philosophy is also applied here, the improvement becomes a significant 33%. Rate-distortion plots appear in Section 2.

In Sections 3 and 4, we present a non-standard approach based on quadtree decompositions using the explicit bit minimization paradigm in which coding improvement for interframe prediction at low bit rates can reach about 40%. (A rate-distortion plot for this experiment is given in Section 3.) Such an improvement suggests that this approach has potential in the related field of optic flow motion estimation.

To the best of our knowledge, ours is the first work investigating the effect of minimizing codelength subject to quality constraints to choose motion vectors. Puri and Hang [11] considered using transform coding for the error, adaptively choosing a transform for each block from among those in a given list by taking the one that resulted in the shortest code.

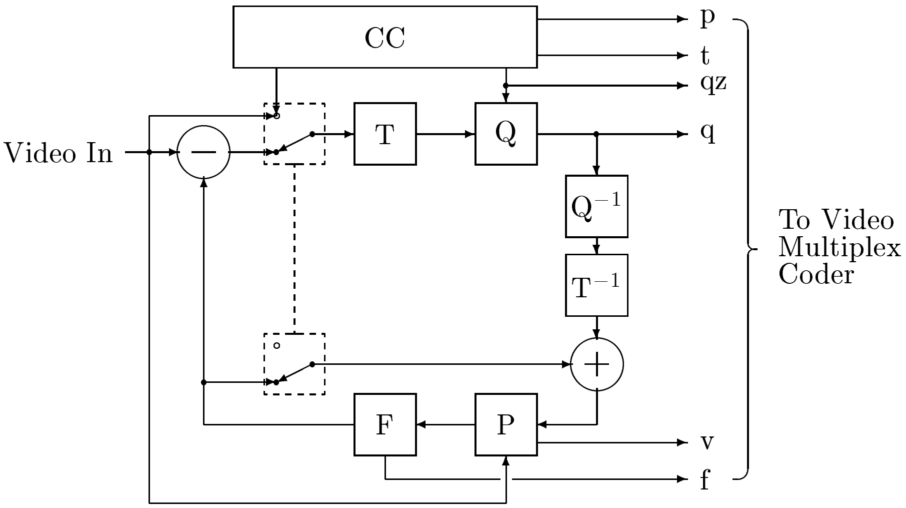
2 Within the $p \times 64$ standard

In this section, we compare the performance of three algorithms which conform to the $p \times 64$ standard [3, 10]. The first algorithm[1] chooses motion vectors to minimize a notion of the prediction error. The second is the same as the first, except that motion vectors are chosen in order to minimize a local approximation to the contribution of the choice of the given motion vector to the total codelength. In the third, certain binary decisions made in the first algorithm using heuristics based on error are instead made again to minimize total codelength.

2.1 Overview of the $p \times 64$ standard

We first provide a brief overview of key components of the $p \times 64$ standard. The $p \times 64$ standard specifies a three-component color system as the format for the video data. The three components are a luminance band Y and two chrominance bands C_{B} and C_{R}. Since the human visual system is more sensitive to the luminance component and less sensitive to the chrominance components, C_{B} and C_{R} are subsampled by a factor of 4 compared to Y. The image is decomposed into *macroblocks* (MB) each consisting of four 8×8 Y blocks, one 8×8 C_{B} block and one 8×8 C_{R} block. Motion prediction and compensation are performed by treating each macroblock as an atomic entity; that is, there is one motion vector per macroblock.

[1] The code for this algorithm was obtained via anonymous ftp from PVRG. We modified this code for the other implementations of algorithms operating within the $p \times 64$ standard.



T: Transform
Q: Quantizer
P: Picture Memory with motion-compensated variable delay
F: Loop Filter
CC: Coding Control

p: Flag for INTRA/INTER
t: Flag for transmitted or not
qz: Quantizer indication
q: Quantizing index for transform coefficients
v: Motion vector
f: Switching on/off of the loop filter

Figure 1: Block Diagram of the $p \times 64$ Source Coder [3]

Figure 1 shows a block diagram of the $p \times 64$ coder. At a high level, the basic process is as follows. The macroblocks are scanned in a linear order. For a macroblock M, the encoder chooses a motion vector $\vec{v}$ (how this is done is left unspecified), and the difference between $\vec{v}$ and the motion vector for the previous macroblock is transmitted, using a static Huffman code. For each 8×8 block B contained in M, a lossy version of the block of prediction errors obtained by using $\vec{v}$ to predict B is then transmitted. This is done by applying a 2D DCT to the block of prediction errors, quantizing the resulting coefficients, and sending the result using a run-length/Huffman coder, where the coefficients are scanned in a zigzag order.

As indicated in the diagram, the encoder has the option of changing certain aspects of the above process. First, the encoder might simply not transmit the current macroblock; the decoder is then assumed to use the corresponding macroblock in the previous frame in its place. If transmitted, the macroblock can be transform coded with motion compensation (interframe coding) or without (intraframe coding). If motion compensation is used, there is an option to apply a linear filter to the previous decoded frame before using it for prediction.

2.2 PVRG implementation of $p \times 64$

In the PVRG implementation, a motion vector $\vec{v}$ is determined for each macroblock M by means of block matching. Only the luminance blocks are compared to the determine the best match, with the mean absolute difference being used as the measure of similarity. The variance V_{P} of the resulting prediction errors for the luminance blocks in M by using $\vec{v}$ is compared against the variance of the luminance blocks in M to determine whether to perform intraframe or interframe coding. The loop filter in interframe mode is enabled if V_{P} is below a certain threshold.

The decision of whether to transmit a transform-coded block is made individually for each block in a macroblock by considering the sum of absolute values of the quantized transform coefficients. If the sum falls below a predefined threshold, the block is not transmitted.

2.3 Algorithm M1

A good first place to apply the bit-minimization principle is in choosing the motion vectors. Instead of performing block matching to minimize the mean absolute difference, we minimize the actual number of bits needed to code the current macroblock. In computing the codelength, we make the same coding decisions as the original PVRG $p \times 64$ implementation and perform the appropriate encoding steps for each choice of motion vector within the search area, picking the motion vector that results in the minimum codelength for the entire macroblock. The computed codelength includes the coding of the transform coefficients, the motion vector, and all other side information. We call this algorithm M1.

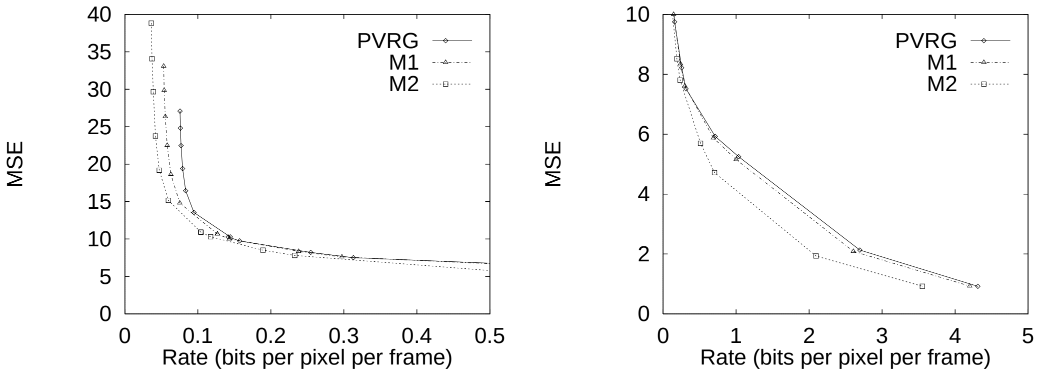
When choosing the motion vector to minimize the coding of the current macroblock, we use the fact that the motion vectors for previous macroblocks (in scan order) have been determined to compute the codelength. However, since the choice of a motion vector for the current macroblock affects the codelength of future macroblocks, this is a greedy minimization procedure, and we may not obtain a globally minimal codelength.

Since we are explicitly attempting to minimize the codelength, we are almost assured to have higher prediction error than if we attempted to minimize the prediction error. Instead of attempting to deal with quality directly, we rely on the transform coder and quantizer to deliver a desired level of quality; that is, the M1 coder may require a finer quantization step size to deliver the same quality as the PVRG coder. As we will see in the results section, bit-minimization does indeed result in consistently better rate-distortion curves.

2.4 Algorithm M2

In Algorithm M1, the decisions of whether to use a filter and whether to use motion compensation are made the same way as in the PVRG $p \times 64$ implementation. In algorithm M2, however, these decisions are also made to minimize codelength: All three combinations of the decisions are tried, and the one resulting in the smallest codelength is used. Here, even more than with M1, we rely on the transform coder and the quantizer to code the prediction errors with adequate quality. Our hope is that the gain in compression efficiency will offset the decrease in reconstruction quality for a given quantization step size; that is, to achieve a certain quality level, we may be able to use finer quantization and still get improvements in compression.

Since M2 is able to make decisions on how to code each macroblock, it is able to take into account the coding of side information in minimizing the codelength. For low bit rates, where the percentage of side information is significant compared to the



(a) Low Bit Rates

(b) High Bit Rates

Figure 2: MSE vs. Rate for Miss America

coding of motion vectors and transform coefficients, one would expect that M2 will be able to reduce the codelength of side information.

2.5 Experimental results

We performed experiments using 150 frames of the "Miss America" sequence in CIF (352×288) format. We ran the three $p \times 64$ algorithms for various quantization step sizes. The search region used for block matching is ± 7 in both directions. Rate-distortion curves are plotted in Figure 2.

As indicated in the plots, M1 performs slightly better than the PVRG implementation and M2 significantly better. For instance, transmitting a CIF-format video sequence at 15 frames per second (fps) on a 128 kbits-per-second line would allow for 8,738 bits per frame. For the same distortion achieved by the PVRG codec at this rate, the M1 and M2 coders would require 17% and 33% less bandwidth, respectively. Equivalently, for the same bandwidth, M1 would be able to code a sequence at 17.5 fps and M2 at 20 fps.

It is interesting to note that, for the one quantization level for which we did this experiment, increasing the search range from ± 7 to ± 15 yielded an improvement only for the M2 coder. The codelength actually increased for the PVRG coder and was unchanged for the M1 coder. This result points out one property of the bit-minimization approach: increasing the search range can never increase the codelength.

Though the experiments were performed with no rate control, we expect similar relative performance even with rate control.

In summary, for one particular test sequence both M1 and M2 exhibit significantly better rate-distortion curves than the PVRG $p \times 64$ coder at low bit rates. In addition, M2 has better performance at high bit rates.

3 Quadtrees for motion vector coding

Like the underlying image data that they are computed from, motion vector fields exhibit an appreciable amount of spatial correlation. Approaches such as the $p \times 64$

standard exploit this observation by coding the differences between successive motion vectors in a one-dimensional scan order. Potentially better results can be achieved by directly exploiting the two-dimensional correlation of motion vectors. A quadtree data structure can be used for this purpose by encoding a hierarchical decomposition of a frame into variable-sized regions of uniform motion [4]. For video sequences where there are large regions of uniform motion, a quadtree decomposition could reduce the number of bits required to encode the motion field compared to a method which used fixed-sized blocks.

In this section, we consider the instantiation of the bit-minimization principle in an algorithm which uses a quadtree to code motion vectors, thereby departing from the $p \times 64$ standard.

3.1 Previous work

Puri and Hang [11] considered an algorithm for motion-compensated video coding which, when an 8×8 block B is not coded well (that is, when coding it requires a lot of bits), chooses a separate motion vector for each of B's four 4×4 subblocks. Bierling [1] described a hierarchical algorithm for choosing motion vectors in which initially motion vectors are chosen for large blocks (64×64) by minimizing the prediction error.[2] Then for each large block B, motion vectors are chosen for subblocks of B again by minimizing prediction error, except looking only at motion vectors close to the motion vector chosen for B. This process results in a smoother motion field, and experiments suggest that it is closer to the "true" motion than is a motion field obtained by separately minimizing error on the small blocks. While Bierling did not discuss how to code motion vectors obtained through by his method, Chan, Yu, and Constantinides [4] described a method where motion vectors are again chosen in a top-down fashion, starting with large blocks and refining with smaller blocks, except when the average squared prediction error for a given block B is below a given threshold, the algorithm does not refine the motion vector chosen for B by looking at subblocks of B. Similarly, if the use of separate motion vectors for the subblocks of B does not reduce the error significantly, the subblocks of B are "merged" with B. After this process is completed, the tree obtained by making a given block the parent of its subblocks is transmitted, together with motion vectors for each of the leaves. Methods for taking a tree structure like the above (except expanded completely) and then "smoothing" the motion vectors by making children tend to be like their parents and vice-versa, were discussed by Dufaux and Kunt [5]. Zhang, Cavenor, and Arnold [15] considered various ways of using quadtrees to code the prediction error.

3.2 Description of our algorithm

In [4], Chan, Yu, and Constantinides describe several coding schemes using quadtrees in which the split/merge operations used to construct the tree are controlled by prediction error criteria. We propose to use a quadtree to encode motion vectors for a block-matching motion-compensated video coder in which the tree is constructed using the bit-minimization principle. The basic coder design is similar to the $p \times 64$ coder shown in Figure 1. The difference is that now motion vectors are coded with a

[2] In fact, a heuristic search [8, 9] was used to only approximately minimize the error.

quadtree whose leaves represent regions of uniform motion. Conceptually, one might associate a motion vector with each *node* of the quadtree, which, for internal nodes, is refined further down the tree. Using this viewpoint, for each node other than the root, the difference between the node's and its parent's motion vectors is transmitted. Thus, one can construct the motion vector for each leaf by adding the root's motion vector to the sum of the differences encountered along the path from the root to the given leaf.

Given a particular quadtree decomposition, we code the structure of the tree using an adaptive arithmetic code to code whether each node is a leaf or not (a different adaptive coder is used for each level). The motion vector differences at each node are coded using another adaptive arithmetic code (again, using a different coder for each level). For each leaf node, the 8×8 transform coded blocks subsumed by the node are transmitted in scan order. The decision of how to code the block (choosing from among alternatives similar to those in the $p \times 64$ standard) is also transmitted using an adaptive arithmetic coder. If the quantized transform coefficients are transmitted, this is done using the run-length/Huffman coding method from the $p \times 64$ standard. The counts for the adaptive arithmetic coder are updated once at the end of each frame.

The quadtree coding structure just described has several nice properties that make a dynamic-programming solution possible for finding an optimal set of motion vectors that minimizes the sum of the codelengths needed to encode the motion vectors and the transform-coded prediction error. Since the code used for the motion vector differences at each node doesn't change during the coding of a particular frame, the optimum number of bits to code the motion vector differences for any subtree is independent of the coding of any other disjoint subtree. Similarly, the transform coding of the prediction errors is independent for disjoint subtrees.

We now describe a dynamic-programming algorithm for choosing an optimal quadtree. For each node in the tree, we store a table indexed by the (absolute) motion vector of the node's parent. This table, for each possible motion vector $\vec{v}$ of the parent, gives the minimum codelength to code the subtree rooted at the current node given that the parent's motion vector is $\vec{v}$. Also stored with each table entry is a motion vector giving the minimum codelength. Construction of the tables is performed in a bottom-up fashion, starting at the 8×8 block level. For a node p, the table is constructed by finding, for each possible motion vector $\vec{v}'$ of the parent of p, a motion vector for p that results in the minimum codelength for the subtree rooted at p. If p is at the 8×8 block level, this is done by computing the transform codelength of the prediction error for each motion vector in the search range S and noting the minimum codelength and the corresponding motion vector. Otherwise if p is not an 8×8 block, we consider for each motion vector $\vec{v}$ in S the codelength needed to transform-code the prediction errors if the quadtree is pruned at p. (This quantity can be computed in a preprocessing step.) We also consider the codelength if the quadtree is not pruned at p. This codelength is computed by indexing the tables of children of p with $\vec{v}$ and summing. The minimum of these two quantities is added to the number of bits to code $\vec{v}' - \vec{v}$. The result is the minimum codelength required to code the subtree rooted at p given motion $\vec{v}'$ at p's parent node.

Once the minimum codelength is computed for the root of the quadtree, the motion vectors for each node in the tree are determined by going back down the tree,

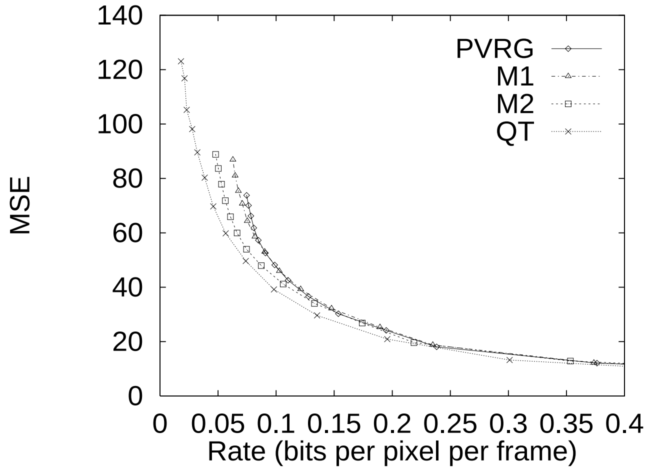


Figure 3: MSE vs. Rate for Trevor

using the tables constructed on the way up. The optimal motion vector for the root node is made known to its children. Each child uses this to index its table to find its optimal motion vector. Pruning of the tree is also performed as a result.

The dynamic-programming algorithm requires $O(N|S|^2)$ time, where N is the number of 8×8 blocks in the frame and S is the search area for block-matching. The space requirement is $O(N|S|)$.

We performed experiments using 50 frames of the grayscale 256×256 "Trevor" sequence. The $p \times 64$ coders were modified to accept input at 256×256 resolution. A rate-distortion plot for the quadtree and $p \times 64$ coders is given in Figure 3.

4 Ongoing research

The current quadtree algorithm uses a bottom-up dynamic programming approach, except that the "bottom" level is defined as the 8×8 block level used for DCT coding. More efficient encoding can be obtained by allowing the quadtree decomposition to go all the way to the pixel level. The most promising approach within reason is to "freeze" the quadtree resulting from the "bottom-up" approach starting at the 8×8 level, and then use a heuristic to expand non-pruned nodes at the 8×8 level. Another use for the top-down heuristic is as a fast approximation to the bottom-up dynamic programming approach.

The same motion estimation methods we discussed in this paper should work as well for MPEG, in which interframe coding can be both forwards and backwards. The MPEG approach is more suitable for computationally intensive encoding methods, as long as decoding is fast, as in the case at hand.

We are also interested in adapting the optic flow algorithms of Shvaytser [12], which use the Occam paradigm of minimizing code length in order to get better estimates of the motion. However, the algorithms in [12] only approximately minimize code length, since they ignore the coding of the error signal. The approach we advocate in this paper, when combined with the prediction framework in [12] should yield better motion estimates.

References

[1] M. Bierling. Displacement estimation by hierarchical blockmatching. *SPIE Vol. 1001 Visual Communications and Image Processing*, pages 942–951, 1988.

[2] B. Carpentieri and J. Storer. A split-merge parallel block matching algorithm for video displacement estimation. In *Proceedings of the 1992 IEEE Data Compression Conference*, pages 239–248, Snowbird, UT, April 1992.

[3] CCITT. Video codec for audiovisual services at $p \times 64$ kbit/s, 1990. Study group XV – Report R 37.

[4] M.H. Chan, Y.B. Yu, and A.G. Constantinides. Variable size block matching motion compensation with applications to video coding. *IEE proceedings*, 137(4):205–212, 1990.

[5] F. Dufaux and M. Kunt. Multigrid block matching motion estimation with an adaptive local mesh refinement. *SPIE Vol. 1818 Visual Communications and Image Processing*, pages 97–109, 1992.

[6] Portable Video Research Group. Pvrg-p64 codec 1.1, 1993. Available from Stanford University by anonymous ftp.

[7] P. G. Howard and J. S. Vitter. Analysis of arithmetic coding for data compression. *Information Processing and Management*, 28(6):749–763, 1992. Special issue on data compression for images and texts.

[8] J.R. Jain and A.K. Jain. Displacement measurement and its application in interframe coding. *IEEE Transactions on Communications*, COM-29(12):1799–1808, 1981.

[9] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro. Motion-compensated interframe coding for video conferencing. *Proceedings of the NTC 81*, pages G5.3.1–G5.3.5, 1981.

[10] M. Liou. Overview of the $p \times 64$ kbit/s video coding standard. *Communications of the ACM*, 34(4):60–63, 1991.

[11] A. Puri and H.-M. Hang. Adaptive schemes for motion-compensated coding. *SPIE Vol. 1001 Visual Communications and Image Processing*, pages 925–935, 1988.

[12] H. Shvaytser. Occam algorithms for computing visual motion. In *Proceedings of ICCV*, 1993.

[13] A. Singh. *Optic Flow Computation.* IEEE Computer Science Press, 1991.

[14] P. Strobach. Quadtree-structured interframe coding of hdtv sequences. *SPIE Vol. 1001 Visual Communications and Image Processing*, pages 812–820, 1988.

[15] X. Zhang, M.C. Cavenor, and J.F. Arnold. Adaptive quadtree coding of motion-compensated image sequences for use on the broadband isdn. *IEEE Transactions on Circuits and Systems for Video Technology*, 3(3):222–229, 1993.