Scalable Coding of HDTV Pictures Using the MPEG Coder

by

Adnan Husain Lawai

B.S., Massacusetts Institute of Technology (1992)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1994

Author ....[illegible]..
Department of Electrical Engineering and Computer Science
May 18, 1994

Certified by.....[illegible]..
V. Michael Bove, Jr.
Associate Professor
Thesis Supervisor

Accepted by ....[illegible]..
Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

Scalable Coding of HDTV Pictures Using the MPEG Coder

by

Adnan Husain Lawai

Submitted to the Department of Electrical Engineering and Computer Science
on May 18, 1994, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Science and Engineering

Abstract

The MPEG-2 coding standard, soon to be put into commercial use, uses DCTs and motion compensation for coding digital images at variable sizes and bitrates. Performing motion estimation at HDTV picture sizes is, however, computationally very expensive. This thesis explores ways of coding High Definition Television images by using the MPEG coder for coding a low resolution image extracted from the original and a simpler coding scheme for the high resolution components of the original not dealt with by the MPEG coder. The advantages of such an implementation are low computational cost and the availability of two bitstreams capable of simultaneously displaying the same image at different resolutions. Two distinct approaches to the problem are proposed and implemented. Experiments on images of differing characteristics show that although the methods developed here are predictably not as efficient as MPEG itself, they do perform well in terms of providing two levels of quality images at high compression rates.

This research was supported by the Television of Tomorrow research consortium at the MIT Media Laboratory.

Thesis Supervisor: V. Michael Bove, Jr.
Title: Associate Professor

Acknowledgments

My hearfelt thanks and deepest respect goes to my advisor, Mike Bove. It has been great working with him in the Media Lab as an undergraduate and graduate student. My thanks also to Andy Lippman, for his guidance and advice on the earlier part of this thesis.

Nuno Vascancelos, with others, carried out the initial experiments that became the core of this thesis. I wish to thank him for helping me to start off, and for his valuable help whenever I needed it. Roger Kermode, Henry Holzman, Joseph 'Foof' Stampleman also deserve my gratitude for fielding my many questions on MPEG and other technical issues.

To all my friends at MIT who made my seven years here so rewarding, thanks for the great times.

And to my parents and family, I couldn't have done it without your support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Simplicity of operation and scalability have recently become the focus of increasing attention within the digital image coding community. A coding method is said to be scalable if the decoder is capable of using part of the bitstream to obtain a displayable image. Using a larger portion of the bitstream, the decoder may be able to increase the resolution of the image, or add more content to it. Scalability is a key attribute in a media system where different applications are making varying demands on channel capacity. In order for such a system to be flexible for the user as well as efficient in terms of the demands it places on channel capacity, the ability to extract useful information from part of the bitsream is essential. In terms of defining a standard for High Definition Television (HDTV), the *backward compatability* requirement forces a two layer scalable structure on the encoded bitstream. Backward compatibility means that receivers/decoders that use the current technology be able to receive/decode the bitstream being encoded by the coder under development. The backward compatible encoder should be able to provide such a bitstream without incurring too great a computational cost.

The coding scheme developed in this thesis attempts to do just that. HDTV images are coded in two steps. A standard MPEG-2 coder encodes a low resolution image derived from the original. The rest of the image — a 'differential' signal consisting mainly of the higher frequency components of the original — is coded using a simple and efficient method. The result is a method of encoding the signal which is computationally not as expensive as encoding the entire image using MPEG-2, and which outputs two displayable bitstreams — one at the low and the other at the high resolution. The savings in computation is made possible by the fact that motion compensation is performed by the MPEG-2 coder only at the low resolution. Thus the very high computational requirements for performing motion compensation at HDTV resolutions incurs is avoided.

This idea is extensible to multiple scales of resolution. Thus, the low resolution signal may be in interlaced format, and the first 'enhahancement layer' might add a differential signal to convert it to progressive format. The next layer might then provide the ehancement necesary for increasing the spatial resolution, followed by a layer making a further increase in resolution possible. This structure is illustrated in figure 1-1.

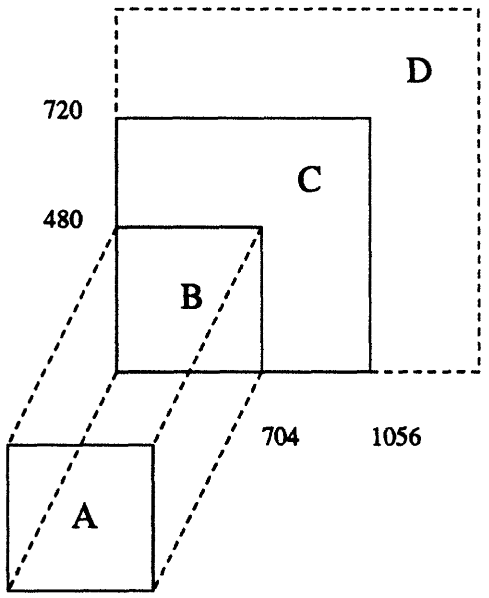


Figure 1-1: Possible structure of the image layers. Region A is at the low resolution and is interlaced. It has been coded by a complex coder such as MPEG-2. Region B extends interlace formant to progressive. Region C, the second enhancement layer extends the spatial resolution of the low resolution image, while region D adds another degree of spatial enhancement

The MPEG-2 [8] (see Chapter 3) coder is a relatively complex one and is designed to operate across a wide spectrum of digital imaging applications. It uses motion compensation to exploit temporal redundancy between frames, while block based

Discrete Cosine Transforms (DCTs) followed by quantization and entropy coding are used to exploit spatial redundancy for achieving compression. Each sequence of pictures is broken up into a series of Group of Pictures (GOPs). Within each GOP, the first frame is purely intra coded using DCTs. The intra coded frame (I frame) is followed by a series of motion compensated frames. These may be of two types — Predictive frames (or P frames) use only previous I or P frames as reference frames for motion compensation, while Bidirectionally predictive frames (or B frames) may use I or P frames which come temporally both before and after that frame as reference frames. The motion compensated prediction error for each non-intra frame is coded using DCTs as well and the DCT coefficients (both from intra and non intra frames) are quantized (using a weighting matrix based on the visual importance of each coefficient), and then coded using differential pulse code modulation and variable length coding. Within each type of frame, blocks may be not coded at all if they are deemed to have too much redundant information. Different methods of motion compensation — which can handle fields separately, or provide more or less accurate predictions depending on user requirements also exist. The MPEG-2 coder provides a high level of compression, and is an easily available, standardized coder. It is ideal for use as the workhorse of our coding scheme at a low resolution, on top of which one or more high resolution 'enhancement' layers may be added. In addition, the current HDTV proposal being standardized closely follows the MPEG-2 syntax. Thus, a proposal developed using this syntax is likely to be practically useful.

The idea of coding a high resolution picture using a complex coding mechanism at a low resolution *base* layer followed by simpler coding at a higher resolution *enhancement* layer has been tried before. Tsunashima et. al. [37] used a two level subband structure with motion compensation at the low and mid levels of resolution to obtain a scalable and efficient coder. The MPEG-2 standard contains three types of 'scalability extensions': spatial, temporal and SNR. In the spatial scalability extension, a motion compensated coded representation of the same image is provided at a lower resolution. This may then be upsampled and compared with the full resolution motion compensated predictions for P or B frames — if the former prediction is better, it may be used instead of the full resolution prediction, or it may be combined with full resolution prediction [8]. In the temporal scalability extension, the enhancement layer is capable of adding temporal resolution (by inceasing the frame rate) to the base layer. SNR scalability involves refining the DCT coefficients in order to improve the quality of the image.

This work builds upon the proposed ATV profile submitted to the MPEG-2 requirements discussion by Andrew Lippman and V. Michael Bove Jr. of the MIT Media Laboratory [22]. This profile proposed a generic scheme for encoding any type of video which may be required at multiple scales of resolution, and which required the following features:

- Highest efficiency at the base resolution
- Inexpensive decoder at the base resolution
- Extensible by more than one spatiotemporal resolution increment
- Satisfactory efficiency at extended resolution
- Inexpensive encoding and decoding at extended resolutions
- Scalable, without the need for simulcasting (treating the low resolution and high resolution components separately).
- Capable of supporting software decoding, low delay modes, VCR features, and rapid lookup.

The problem of designing a coder to meet our constraints can be broken down into distinct but interrelated parts. The first step is concerned with deriving an appropriate low resolution signal from the original which is given to the MPEG-2 coder. The choice of an appropriate method here is dictated by MPEG-2 performance characteristics at that low resolution. Once such a 'downsampling' method has been obtained, we can determine the characteristics of the 'difference' signal, i.e. that part of the original signal where the MPEG-2 coder does an inadequate job of coding (or that part of the original not coded at all by the MPEG-2 coder). Using this information, the second step of our coding scheme has to compensate for what the lower level MPEG-2 coder has left out — i.e. the 'difference' signal has to be coded so that it may provide effective enhancement for the lower layer signal.

Two methods have been been tried for the 'downsampling' part of the coding system. The first method downsamples the image in the 'standard' way [28]. That is, given a high resolution frame, this method separably downsamples and filters along each dimension in order to obtain a frame at the low resolution. Thus, this method downsamples from a progressive format to a progressive format. The second method downsamples from a progressive to interlaced format. It takes a pair of frames from the original and extracts a *field* of a low resolution frame from either frame. This is also done by separable downsampling and filtering, but in this case the downsampling factor for each frame is twice that used in the first case. In either case, the higher spatial frequency components of the full resolution image are not present in the lower resolution image, and need to be compensated for.

The second stage of the system codes these frequency componets, and those portions of the image spectrum which are badly coded by the MPEG-2 coder. This part of the coding has been performed using subband analysis. Subbands seem to be the natural choice, since they provide us with different portions of the spatiotemporal frequency spectrum which we can then quantize according to how good a job the MPEG coder has done in coding each subband. In the experiments for this thesis, our lower level resolution was two thirds of the full resolution. This suggests the use of a nine subband decomposition (three in each direction) in the spatial frequency domain. Thus, theoretically (assuming perfect subband analysis and synthesis filters as well as perfect anti-aliasing and interpolation filters) the bottom four subbands of the high resolution pictures' spatial frequency spectrum are the only ones included in the low resolution picture encoded by MPEG-2. The top five subbands need to be coded, along with the visually significant data that has been corrupted by the MPEG-2 Coder in the bottom four subbands (if there is any). Two approaches have been proposed for this part of our system, which we will refer to as the openloop and the closedloop coders.

The openloop coder works on the premise that the MPEG coder does a sufficiently good job at the lower level, so that we need not compensate for any errors made in coding the information contained in the lower subbands. Consequently, only the higher frequency subbands need to be coded. The closedloop coder, on the other hand, is designed to be more robust. It can compensate for errors made by the MPEG coder as well as code the high frequency details not handled by the lower level coding. It does this at the cost of coder complexity, with an associated cost in terms of coding delay. The coded lower resolution picture is decoded at the encoder — upsampled to full resolution and subtracted from the original to yield an 'error' image. It is this error image which is analyzed into subbands. Thus mistakes made at the lower resolution — such as motion compensation artifacts — can be compensated for.

The subbands are coded using scalar quantization followed by run-length and entropy coding. Bit allocation amongst subbands is based on the the visual importance of the data in each subband as well as the variance of a subband — which is considered a measure of the energy of the data contained in a subband. An algorithm for rate control has also been developed.

This thesis is organised in the following manner. Chapter 2 discusses the image compression techniques that were used in this project. Chapter 3 describes the salient features of the MPEG-2 coding syntax, and presents an analysis of the computaional complexity of the MPEG-2 coder. Chapter 3 describes the technical details of our coding methods. Chapter 4 presents experimental data and their analysis, while Chapter 5 ends with suggestions for future work and conclusions.

Chapter 2

Background: Image Compression

Compression is the key to doing almost anything with images. Existing storage and transmission technologies simply cannot meet the huge demands placed on them by full motion, full colour, digital video. The original sequences used for this project were in RGB colour at 24 bits per pixel, 720 by 1056 pixels per frame, 60 frames per second. To transmit this as uncompressed video would require a transmission 'bandwidth' of 1094860800 or about 1.1 Gigabits/second. Storing an hour's worth of uncompressed video would require about $4 * 10^9$ Gbits of storage space. The goal in this thesis was to code this data such that the final, compressed image would require only about 20-25 Megabits/second i.e. we were aiming at a compression factor of about 40-50:1. The goal of this section is to describe the techniques that were used in this project.

Compression techniques are of two types. Lossless coders exploit redundancy in data to devise a rendition of the data such that all of the coded information can be recovered by the decoder. Lossy coders, on the other hand, lose information. The designer's job is to devise a lossy coder such that the perceptually most insignificant information is lost. The codec used in this experiment is overall a lossy one.

In terms of image compression, quantization is usually (and certainly in this case) the most significant portion of the coder. We begin with a discussion of scalar quantization in Section 2.1. Section 2.2 discusses transform coding in the context of the Discrete Cosine Transform (DCT). Section 2.3 describes subband analysis. Section 2.4 describes, motion compensation, a temporally predictive technique for achieving compression. Finally, Section 2.5 explains lossless codes, in particular, it discusses Variable Length Codes (VLCs), with emphasis on Huffman Coding and Arithmetic Coding.

2.1 Quantization

A *quantizer* maps a large (possibly infinite) set of input values into a smaller set of *quantized* (or reconstructed) values. Amplitude quantization is required when converting continuous analog signals to discrete digital ones. In image compression systems, quantization provides a significant portion of the compression. Quantizers are of two types: *scalar quantizers* process one input sample at a time. *Vector quantizers*, on the other hand, process a collection (vector) of input samples at a time.

Mathematically, a scalar quantizer's operation may be characterized as follows: If f is the input, and $\hat{f}$ represents the quantized output:

$$\hat{f} = Q(f) = r_i \quad d_{i-1} < f \leq di \tag{2.1}$$

where Q represents the quantization operation, r_i for $1 \leq i \leq L$ denotes L reconstruction levels, and d_i for $0 \leq i \leq L$ denotes L decision boundaries. If the value of f falls in the region between d_{i-1} and d_i, it is assigned the quantized value r_i. We may also express $\hat{f}$ in 2.1 as

$$\hat{f} = Q(f) = f + e_Q \tag{2.2}$$

such that, e_Q, is the quantization error given by

$$e_Q = \hat{f} - f \tag{2.3}$$

Since quantization is a lossy operation, the role of the scalar quantizer's designer is to make the *distortion* of the original signal as visually insignificant as possible. This is a very difficult problem to deal with quantitatively since (1) the characteristics of the *Human Visual System* (HVS) are not fully understood and (2) they cannot as yet be quantified in a mathematically tractable way. In the absence of a criterion with a good physiological basis, mathematical tractability is emphasized and the Mean Squared Error (MSE) *distortion measure* is used almost universally. This defines the distortion between f and $\hat{f}$ to be:

$$d(f,\hat{f}) = |\hat{f} - f|^2 \tag{2.4}$$

The quantity e_Q^2 is thus seen to be the MSE distortion measure.

Now, given any general distortion measure $d(\hat{f}, f)$, the total distortion D in the input is, for an L level quantizer, given by

$$D = \sum_{i=1}^{L} \int_{di-1}^{d_i} d(f_0, \hat{f}_i) p_f(f_0) df_0 \tag{2.5}$$

where $p_f(f_0)$ is the probability distribution of the input. Thus, given a probability distribution, it should be possible to find a quantizer that minimizes this quantity. In practice, probability distributions for image amplitudes are not known, but can be approximated from a histogram of image amplitudes or a known image model.

Minimizing equation 2.5 with respect to $\hat{f}_k$ for $1 \leq k \leq L$ and d_k for $1 \leq k \leq L-1$, we get (using the MSE distortion criterion) [21]:

$$\begin{aligned} r_k &= \frac{\int_{f_0=d_{k-1}}^{d_k} f_0 p_f(f_0) df_0}{\int_{f_0=d_{k-1}}^{d_k} p_f(f_0) df_0}, 1 \leq k \leq L && (2.6) \\ d_k &= \frac{\hat{f}_k + \hat{f}_{k+1}}{2}, 1 \leq k \leq L && (2.7) \end{aligned}$$

This set of equations states that a reconstruction level $\hat{f}_k$ is the centroid of $p_f(f_0)$ over the interval $d_{k-1} \leq f_0 \leq d_k$, and that the decision level d_k (except for d_0 and d_L) is the midpoint between the two reconstruction levels $\hat{f}_k$ and $\hat{f}_{k+1}$.

The most straightforward method of quantization is uniform quantization, in which the reconstruction and decision levels are uniformly spaced. For a uniform quantizer:

$$d_i - d_{i-1} = \Delta, \quad 1 \le i \le L \tag{2.8}$$

and,

$$r_i = \frac{d_i + d_{i-1}}{2}, \quad 1 \le i \le L \tag{2.9}$$

Where Δ is the step size (the spacing between decision boundaries). An example of a uniform quantizer is shown in figure 2-1(a). It turns out that a uniform quantizer is the optimal quantizer (in terms of the MSE distortion measure) given an input sequence with a uniform probability distribution. Optimal quantizers have been computed for other probability distributions [16], and an example of a nonuniform quantizer is shown in figure 2-1(b).

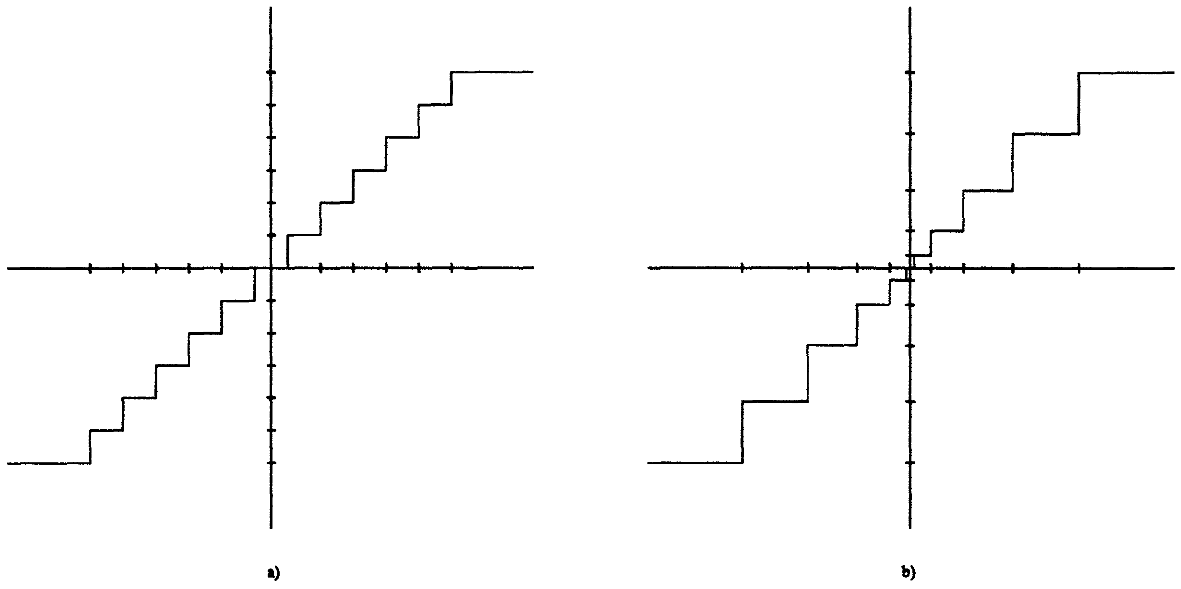


Figure 2-1: Quantizer transfer function: a) uniform, b) non-uniform.

2.2 Transform Coding: The Discrete Cosine Transform

The previous section discussed some of the issues related with quantizing a scalar source. The main limitation of a scalar quantizer is that, processing one sample at a time, it is incapable of exploiting the redundancies in a typical image. A pixel in a typical scene has a value closely related to the values of the surrounding pixels, and this correlation can be exploited by taking account of the vector nature of the source image. In transform image coding, a group of pixels (a vector) is transformed to a domain significantly different from the image intensity domain. By doing so, we wish to exploit two properties:

- The transform coefficients reduce the correlation that exists amongst pixel intensities. Because of this *correlation reduction property*, redundant information does not have to be coded repeatedly.

- For typical images, a large amount of energy is contained in a few transform coefficients. This *energy compaction property* enables us to code a small fraction of the transform coefficients without having to sacrifice much image quality.

The correlation reduction property can be illustrated by means of the following example, originally presented in [24].

Suppose a source generates two dimensional random vectors $\mathbf{x} = [x_1\, x_2]^T$ characterized by the joint pdf $p_{x_1,x_2}(x_1, x_2)$ represented in figure 2-2.

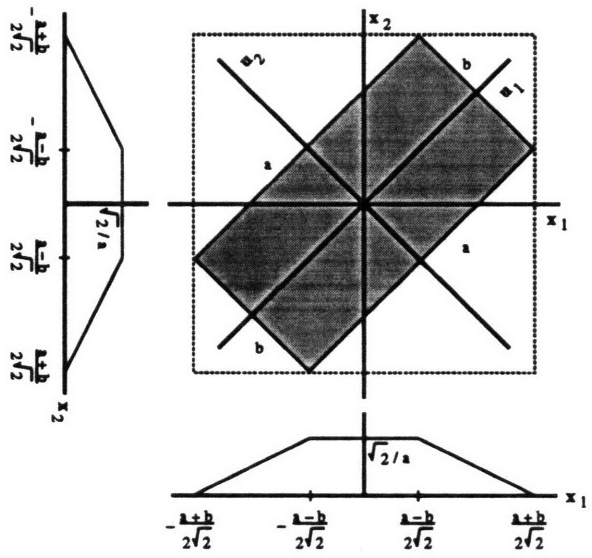


Figure 2-2: Example of a two dimensional pdf, $p_{x_1,x_2}(x_1, x_2)$, uniform in the shaded area and null on the outside. The marginal pdfs, $p_{x_1}(x_1)$ and $p_{x_2}(x_2)$, are also represented. From [24], as it appears in [39].

It is clear from the figure that $p_{(x_1,x_2)}(x_1, x_2) \neq p_{x_1}(x_1)p_{x_2}(x_2)$, i.e. that x_1 and x_2 are not independent. If we notice that the pdf is oriented in the direction of the line $x_1 = x_2$, it is clear that the two components are more likely to have similar amplitudes. Thus there is some redundancy between the two components.

Figure 2-3 shows the optimal partitioning of the two dimensional space if both components are scalar quantized separately (for an optimal 8 level quantizer along each component). This partition is clearly sub-optimal, since large regions with zero probability also have a reconstruction value associated with them, and it would be clearly better to have all of the reconstructed values inside the shaded region. Much better results can be obtained by a simple rotation of the coordinate axes. Figure 2-4

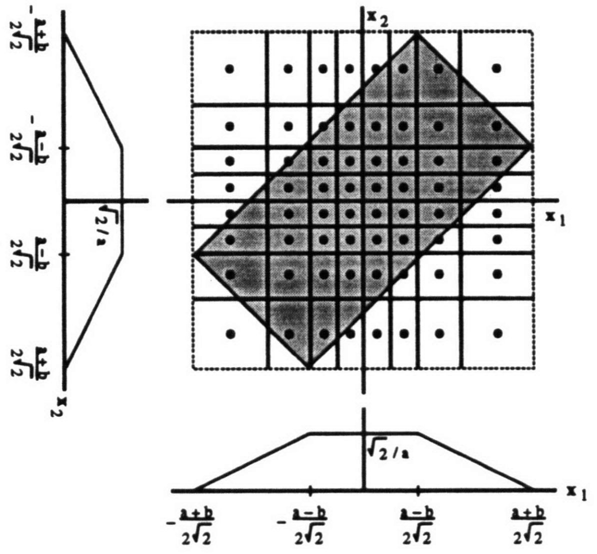


Figure 2-3: Partition of the 2-D space with scalar quantization. The dots represent the reconstruction value associated with each region of the partition. From [39]

shows the same joint pdf, but rotated such that the new coordinate axes are $u_1 = x_1 + x_2$ and $u_2 = x_2 - x_1$. After the rotation, the two components become independent and it is possible to do a much more efficient job of scalar quantization by avoiding zero probability partitions.

When, as in this example, the input random variables can be made independent by a linear transformation, they are said to have linear dependencies. In practice, image intensities cannot in general be made independent via a linear transform. They can, however, always be decorrelated, and that, followed by scalar quantization, provides good compression.

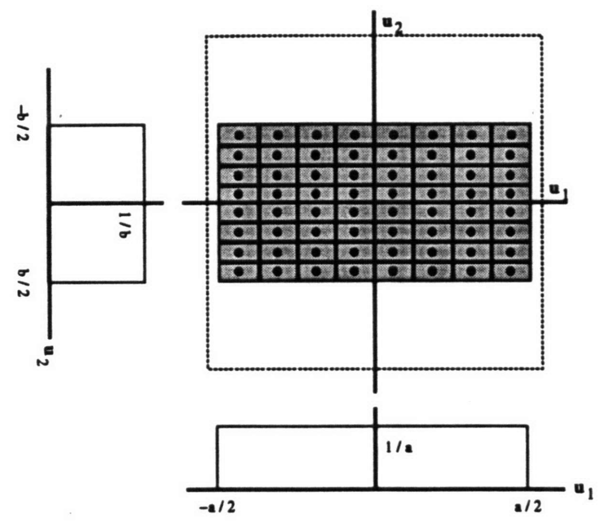


Figure 2-4: Joint pdf after rotation of the coordinate axis. The marginal densities and the partition regions associated with separable scalar quantization with 8-levels are also shown. The rotation makes the random variables independent, and the distortion is minimized. The dashed line is included for comparison with figures 2-2 and 2-3. From [39]

Suppose input samples at the transmitter are grouped together into vectors of dimension K, linearly transformed, quantized, and then transformed back to the original domain (presumably at the decoder, after some sort of channel transmission). Such a set of operations is known as transform coding.

The linear transformation maps the input vector $\mathbf{x} = [x_1, ..., x_K]^T$ into a vector $\mathbf{u} = [u_1, ..., u_K]^T$ according to

$$\mathbf{u} = \mathbf{T}\mathbf{x}. \tag{2.10}$$

The row vectors $\mathbf{t_i}^T$ of $\mathbf{T}$, are generally known as the *basis functions* of the transform, and since

$$u_i = \mathbf{t_i}^T\mathbf{u} = \sum_{j=1}^{K} t_{ik}\, u_k,\ k = 1, \ldots, K, \tag{2.11}$$

the components of the transformed vector $\mathbf{u}$ (known as the *transform coefficients*) are nothing more than the projections of $\mathbf{x}$ onto these basis functions. It is, therefore, a necessary condition that the set of basis functions can span the entire K-dimensional space, i.e. the basis vectors have to be *linearly independent.*

A desirable property of any transform is that the basis vectors be *orthonormal,* i.e. they satisfy

$$\mathbf{t_i}^T\mathbf{t_j} = \begin{cases} 0, & \text{if } i \neq j \\ 1, & \text{if } i = j, \end{cases} \tag{2.12}$$

In this case, the inverse transformation is very easy to compute. From 2.12, since

$$\mathbf{T}\mathbf{T}^T = \mathbf{I}, \tag{2.13}$$

$$\mathbf{T}^{-1} = \mathbf{T}^T \tag{2.14}$$

It is possible, given the covariance matrix of the image, to compute the transform which is optimum in terms of the correlation reduction and energy compaction properties. This is known as the *Karhunen-Loeve Transform* [21]. However, the covariance matrix of an image is in general not available, and is difficult to estimate. Even if the covariance matrix is known, there is no efficient way of computing the transform coefficients. For this reason, the Karhunen-Loeve transform, though interesting theoretically, is hardly ever used in practice.

The most commonly used transform for image coding is the *Two Dimensional Discrete Cosine Transform* (the DCT). The One Dimensional DCT is closely related to the more commonly known Discrete Fourier Transform (DFT) [28]. The main distinction between the two is that while the DFT uses complex exponentials as basis functions, the DCT uses real (co)sinusoids as its basis functions. This works well, since image intensities are real valued and thus we need not have imaginary sinusoids as our basis functions. In addition, the DCT is much better than the DFT in terms of the energy compaction property (see [21] for a discussion of why this is true). A number of well known algorithms exist for efficient computation of both the DFT and DCT [28]. The Two Dimensional DCT, uses two dimensional (co)sinusoids as its basis functions.

If $x(m,n)$ in an N by N input, the 2D DCT coefficients $F(u,v)$ are defined by:

$$F(u,v) = \frac{2}{N}\alpha(u)\alpha(v)\left[\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} x(m,n).\cos\frac{(2m+1)u\pi}{2N}\cos\frac{(2n+1)v\pi}{2N}\right] \tag{2.15}$$

where,

$$\alpha_k = \begin{cases} 1/\sqrt{2}, & \text{if } k = 0 \\ 1, & \text{if } k \neq 0. \end{cases} \tag{2.16}$$

The 2D *Inverse DCT (IDCT)*, which given N by N transformed coefficients, maps them back to the original domain, is defined by:

$$X(m,n) = \frac{2}{N}\left[\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} \alpha(u)\alpha(v)F(u,v).\cos\frac{(2m+1)u\pi}{2N}\cos\frac{(2n+1)v\pi}{2N}\right] \tag{2.17}$$

where $\alpha(k)$ is as defined in 2.16. The basis functions of the 2D DCT are shown in figure 2-5.

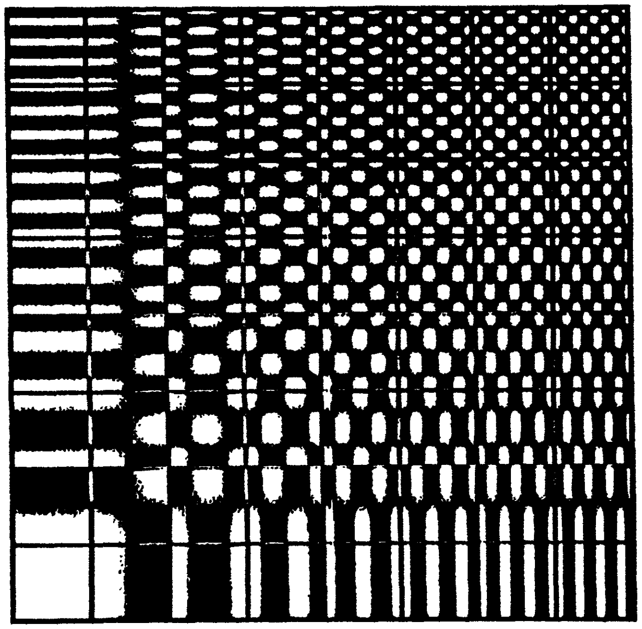


Figure 2-5: Basis vectors of a 2D 8 x 8 Discrete Cosine Transform (DCT). From [17].

An important feature of the 2D DCT is that it is *separable*. That is, each basis function (which is a function of two variables) can be represented as a product of two functions of one variable:

$$b_{i,j}(m,n) = b_i(m)b_j(n) \tag{2.18}$$

This property enables us to obtain the transform of an image by first performing a 1D DCT separately on all rows of the image and then applying a 1D DCT to all the columns of the result (*row column decomposition*). We can now exploit the efficient algorithms which exist for taking 1D DCTs. Thus the 2D DCT can be computed very efficiently.

Often the image to be coded is broken up into a number of 'subimages' and these are transform coded separately. This is done for a number of reasons. Coding different portions of the image separately enables us to code the image adaptively. Thus, in a uniform region, we need not code the transform coefficients corresponding to the the higher frequencies as well as we would need to in a region with a lot of detail. Since the entire image is not needed at one time, memory requirements are reduced, Subimage coding also reduces computational requirements in most implementations, which use efficient FFT like procedures.

The subimages cannot be made too small though, since decreasing image size decreases the amount of redundancy within each block that can be exploited. Large subimages, on the other hand, result in decreased adaptivity and greater memory and computational requirements. Typical subimage sizes are 8x8 (the size used by the MPEG coder) and 16x16.

2.2.1 Quantization in the Transform Domain

The idea behind transform coding is obtain transform coefficients with special properties that may be exploited by a quantizer to get more compression. An important property of any transform is the energy compaction property. For the DCT, this means that the 'lower frequency' transform coefficients contain most of the energy in the image. Thus, the higher frequency components may be quantized very coarsely, or not coded at all. Another consideration is the sensitivity of the Human Visual System (HVS) to each of the coefficients. Fortunately, for the DCT, the HVS is more sensitive to the lower frequency coefficients, particularly the DC coefficient.

Mathematically rigorous derivations of optimal bitrates depend on the criterion of optimality being employed, none of which is the definitively 'correct' one. Minimizing the MSE and keeping the average bitrate constant gives us

$$b_i = R + \frac{1}{2}\log_2 \frac{\sigma_i^2}{\left[\prod_{k=0}^{N-1}\sigma_k^2\right]^{1/N}}, \tag{2.19}$$

where b_i and σ_i^2 are, respectively, the bitrate allocated to and the variance of coefficient i, R is the average target bit rate, and N is the number of coefficients per block [16]. This result, originally derived by Huang and Schultheiss in 1963 [10], confirms the intuitive notion that the bitrate assigned to a coefficient should be proportional to the image energy contained in that coefficient as a fraction of the total image energy. It is easy to incorporate further perceptual considerations into this result by substituting all the σ_i^2's by $\omega_i\sigma_i^2$, where ω_i is the perceptual weight of each band, and $\sum_{i=0}^{N-1}\omega_i = 1$.

Equation 2.19 provides the theoretically optimal bit allocation, but is not always usable in practice since it does not satisfy the constraint that the individual bitrates b_i must be integers. In practice, the bitrates derived from 2.19 can be used as an initial estimate for bit allocation, followed by an algorithm which adjusts these rates according to additional constraints.

Two heuristic methods commonly used for quantization of transformed coefficients are *Zonal coding* and *Threshold coding* [21]. In Zonal coding only coefficients in a predetermined zone are quantized — the rest are discarded. Within the zone, the coefficients are typically assigned different bitrates based on 2.19 or other more ad hoc considerations. In Threshold coding, the coefficient magnitudes are compared with some threshold, and only those which are larger than the threshold are kept. The rest are discarded. From an energy compaction point of view, Threshold coding is preferable, since in Zonal coding some coefficients with small magnitudes are kept while other coefficients with large magnitudes may be thrown away. However, the positions of the coefficients which are kept are not known a priori and need to transmitted to the decoder. Threshold coding is an adaptive procedure — the number of coefficients kept and thus the total bitrate required to code them are not known. A method of assigning bits amongst the coefficients (again, some method based on equation 2.19 may be used here) which are kept and for controlling the total bitrate is required.

Once bit allocation has been performed, there still remains the problem of quantizing the coefficients. The human eye is in particular very sensitive to errors in the DC coefficient, i.e. the average brightness of the (sub)image. Coarse quantization of the coefficient results in the 'blocking effect', which exposes the block based structure of the coding in the decoded image, and which looks particularly unpleasant. The DC coefficient is thus very finely quantized, using (usually) a uniform scalar quantizer, since the pdf for this coefficient has empirically been found to be approximately uniform for most images. The AC coefficients are not as crucial to picture quality and are thus not so finely quantized. Their pdfs are approximately gaussian and thus scalar quantizers optimized for gaussian pdfs can be used to quantize them.

2.3 Subband Coding

Subband coding of signals is another incarnation of linear transform coding, first used to code speech by Crochiere et. al [6]. Subband analysis of a signal involves separating the spectral components of signals into different subbands, which may then be coded. The signal is convolved with an array of bandpass filters, and the result of each convolution is then downsampled to its Nyquist rate. The resulting smaller images are known as *subbands.* Each subband may then be coded separately.

The original image can be recovered from its subbands by first zero-padding each subband by the amount it was downsampled with and then applying an appropriate interpolation filter to isolate each subband at the original size. The results of this operation applied to all the subbands are added back together to obtain the final reconstructed image. This procedure is illustrated in figure 2-6.

Figure 2-7 shows the frequency domain effect of splitting the image spectrum using ideal 'brick wall' bandpass filters. For image coding subbands analysis/synthesis is typically applied separably along each dimension.

Subband coding originally emerged as a way of coding speech signals. The impetus for this development was the human auditory system's nonlinear response to different

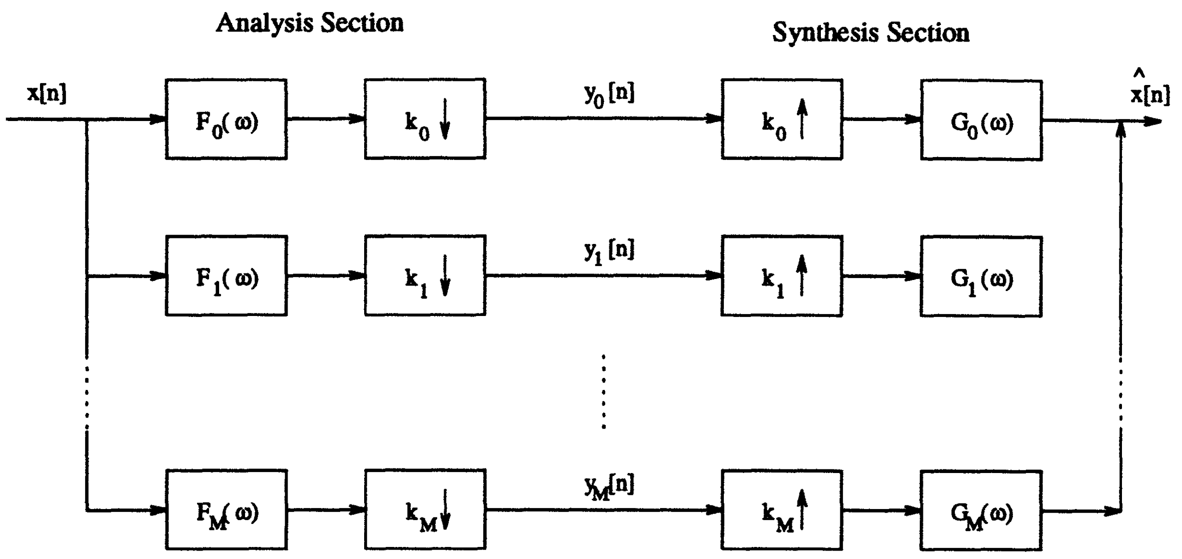


Figure 2-6: Arbitrary M band analysis/synthesis system

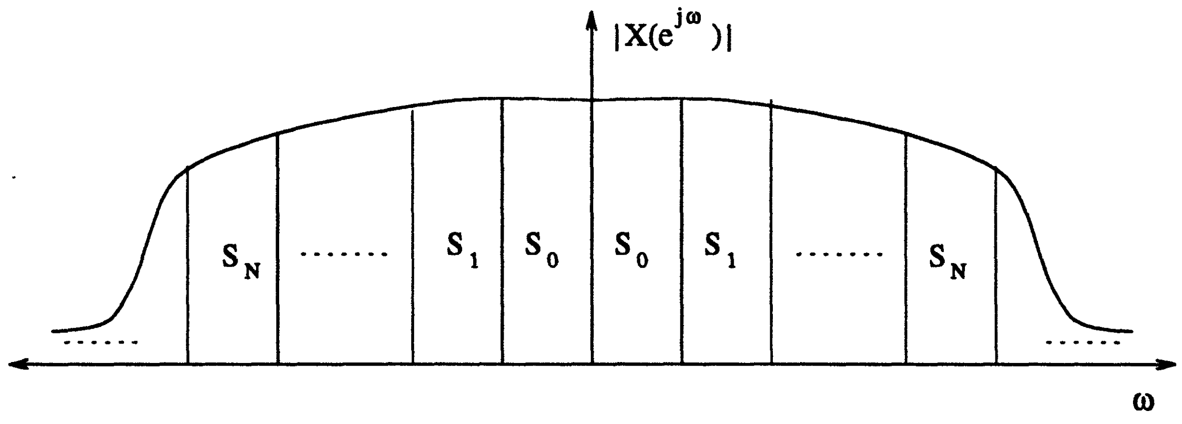


Figure 2-7: Frequency domain effect of subband coding using 'brick wall' filters

frequencies. Two significant advantages of this approach were identified. First, since the subbands were coded separately, the quantization noise was localized to each band and did not affect the other bands. Second, the bit allocation could be varied across the bands so as to more closely model the energy distribution and perceptual significance across the bands.

The qualities deemed advantageous for subband coding of speech are equally important for the coding of images. The human visual system has a nonuniform response across different portions of the spatio-temporal spectrum. Schreiber [32], Troxel et. al [5], and Glenn [4] all discuss the reduced distortion sensitivity of the HVS to high frequency spatial detail, in particular when coupled with high frequency temporal detail. The HVS is extremely difficult to model overall, but some good approximations exist for particular regions of the spatio-temporal spectrum (see Butera [1]). Temporally the HVS functions approximately as a differentiator at low frequencies and an integrator at high frequencies. Therefore, a simple halfband filter applied along the temporal axis captures the essence of this bimodal response. Spatially, available evidence suggests that the subbands should be arranged in a series of oriented bandpass channels as discussed by Marr [25], Hildreth [9], and Hubel [11]. In particular, for a fixed retinal position, there are well known components of the early visual pathway that can be well approximated using filter banks so that the bandwidths vary in increments of roughly singly octaves.

'Brick wall' bandpass filters are, however, impossible to realize in practice (according to theory, they would have to be infinitely long in the spatial domain). Very close approximations to 'brick wall' filters are also undesirable, since they cause 'ringing' in the spatial domain according to the well known 'Gibbs' phenomenon. Given this limitation, we have two choices:

- We may employ filters which fall off gradually but do not overlap at all. This results in no aliasing between the subbands but there is some information in the original which is not included anywhere in our subbands.

- We may use filters which do overlap, so that each subband contains some information from the next subband, and all the information in the original is contained in the subbands. However, it is possible to design analysis/synthesis filters such that the aliasing (overlap) cancels out at the output, and, in some cases, perfect reconstruction is achieved.

The latter approach is the one used in practice, and the analysis/synthesis filters commonly used are known as *Quadrature Mirror Filters (QMFs).* In this project, because of the peculiar nature of the problem, 3 band QMFs were chosen. The design of QMFs such that perfect reconstruction is guaranteed is dealt with is the subject of the next section.

It would be useful at this point to elaborate on the fact that subband analysis and linear transforms are essentially variations on the same theme . A block based transform coder (using a block size of N by N) can be thought of as a subband coder by considering the N basis functions of the (forward) transform to be the analysis filters of a subband coder, and using a decimation factor of N. This is followed by interpolation by a factor of N using the (inverse) transform basis functions as the synthesis filters. Admittedly, when treated as bandpass filters, DCT basis functions do not provide very good frequency localization. They are not meant to, however. The DCT basis functions are chosen for their energy compaction and correlation reduction properties, while subband analysis/synthesis filters are chosen for their ability to isolate different regions of the spectral domain, which may then be coded with psychovisual considerations in mind.

2.3.1 Design of Perfect Reconstruction Quadrature Mirror Filters

The problem here is to design analysis/synthesis filters, such that $\hat{x}[n]$ in figure 2-6 is a perfect (or as perfect as possible) replica of $x[n]$. Downsampling corresponds to a 'stretching' in the frequency domain [28], such that the frequency response of the intermediate signal $y_i[n]$ is given by:

$$Y_i(\omega) = \frac{1}{k}\sum_{p=0}^{k-1} F_i\left(\frac{\omega}{k} + \frac{2\pi p}{k}\right) X\left(\frac{\omega}{k} + \frac{2\pi p}{k}\right) \tag{2.20}$$

where we have assumed that k_0, k_1 etc. are all equal to k. It is this 'stretching' that causes aliasing in the frequency domain, as illustrated in figure 2-8. Upsampling on the other hand, causes a 'compressive' rescaling of the of the frequency axis (illustrated in figure 2-8), such that the expression for the output of the system in figure 2-6 is given by:

$$\hat{X}(\omega) = \sum_{q=0}^{M-1} Y_i(k\omega)G_i(\omega) \tag{2.21}$$

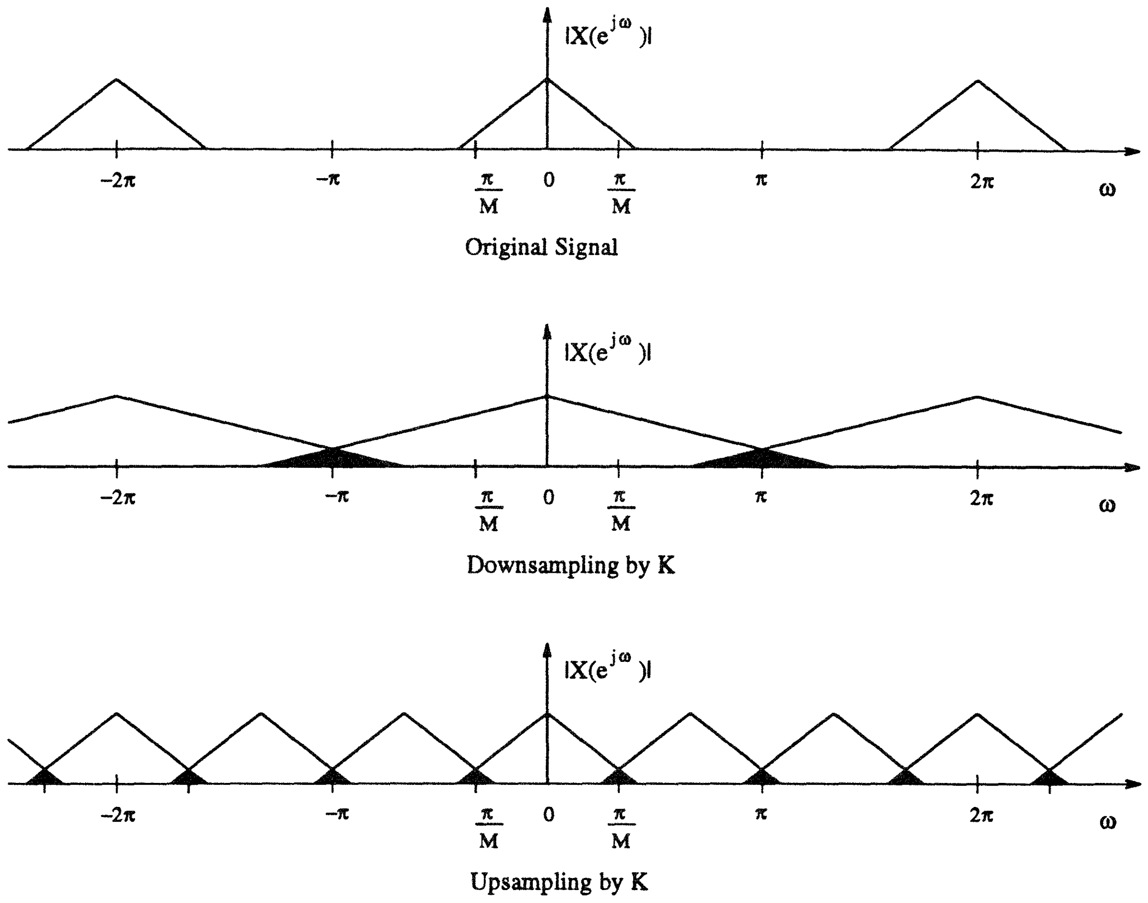


Figure 2-8: Effects of (a) downsampling by a factor of k, followed by (b) upsampling by a factor of k, without any anti-aliasing or interpolation filters. The shaded areas represent the aliasing.

Combining the two gives:

$$\begin{aligned}\hat{X}(\omega) &= \frac{1}{k}\sum_{q=0}^{M-1}\left[\sum_{p=0}^{k-1} F_i\left(\omega+\frac{2\pi p}{k}\right) X\left(\omega+\frac{2\pi p}{k}\right)\right] G_q(\omega) \\ &= \frac{1}{k}\sum_{q=0}^{M-1} F_i(\omega)G_i(\omega)X(\omega) \\ &\quad +\frac{1}{k}\sum_{p=1}^{k-1} X\left(\omega+\frac{2\pi p}{k}\right)\sum_{q=0}^{M-1} F_i\left(\omega+\frac{2\pi p}{k}\right) G_q(\omega) \qquad (2.22)\end{aligned}$$

The first term corresponds to a linear shift-invariant system response, and the second contains the system aliasing [34]. It is this second term which is to be eliminated by appropriate design of the analysis and synthesis filters $f_i[n]$ and $g_i[n]$.

A further constraint we would like to have on our analysis/synthesis system is orthogonality i.e. we would like the 'basis functions' of the analysis section to be orthogonal to each other. This enables the transformation we have applied in the analysis section to be easily inverted. In addition, coefficients derived using an orthogonal transform represent distinct information, and can be effectively used for the kind of psychovisual weighting that subband transform coeffiecients are useful for. When combined with the perfect reconstruction constraint, orthogonality yields the constraint that the synthesis filters must be *time reversed* versions of the analysis filters, i.e.

$$g_i[n] = f_i[-n], \text{for all i} \tag{2.23}$$

Furthermore, orthogonality by itself yields the constraints that,

$$\sum_{n=0}^{N-1} f_i[n] f_i[n - mk_i] = 0, \text{for all } m \neq 0 \tag{2.24}$$

and,

$$\sum_{n=0}^{N-1} f_i[n] f_j[n - mk_j] = 0, \text{for all } i \neq j \tag{2.25}$$

where the filter sample locations are computed modulo N [34].

A number of techniques exist for designing orthogonal analysis/synthesis filters for perfect reconstruction in a M band filter bank [38]. In this project the techniques developed by Simoncelli [34] were employed. Using the most basic method, we select M filters with the approximate characteristics we want (in terms of band cutoffs, orthogonality etc.) and we derive a square, full rank *analysis matrix* **A**, such that

$$y = \mathbf{A}x \tag{2.26}$$

where x is the vector of inputs and y is the vector of subband transformed outputs. We then perform an iteration on the analysis matrix according to the following algorithm:

$$\mathbf{A}' \leftarrow \frac{1}{2}\left[\mathbf{A} + (\mathbf{A}^{-1})^t\right] \tag{2.27}$$

It can be shown that starting from an appropriate matrix **A**, this method will yield an orthogonal analysis system. The problem with this method is that it depends on initial conditions and that we have no control over the frequency response of the output. This can be improved upon by using a frequency sampling technique that satisfies the perfect reconstruction constraint in the frequency domain. Both methods, however, have the disadvantage that filters with sharp transitions in frequency are difficult to realize, and thus the M band subband coder using these filters does have considerable information 'leaking' across bands.

2.4 Temporally Predictive Coding: Motion Estimation

All the methods described so far in this chapter have relied on exploiting the redundancy amongst the pixels of an image to achieve compression. When an image sequence needs to be coded, another source of redundancy becomes available: the picture does not change much from frame to frame. This leads to the idea of attempting to predict a given frame from its immediately surrounding frames. This is a specific case of a more general procedure known as *predictive coding.*

Predictive coding is based on the notion that it is more efficient to predict a sample (which may be a frame, or just a pixel value) and code the error. Already coded values from the sequence from which the sample is derived may be used for forming a prediction. It is necessary in this case that the encoder be able to make the same prediction that the decoder would make i.e. the encoder must have part of the decoder built into it. Once the prediction has been formed at the encoder, it subtracts the prediction from the original value, and then codes the error signal (using, perhaps, one of the methods described earlier). The decoder takes a previously decoded value and (possibly) combines it with additional information to form a prediction for the current value. The error is then decoded and and added to the prediction to give the decoded value of the sample. This process is illustrated in figure 2-9.

For predicting a frame from its surrounding frames, a procedure known as *motion estimation* is used. This is based on the premise that most motion in a typical scene is

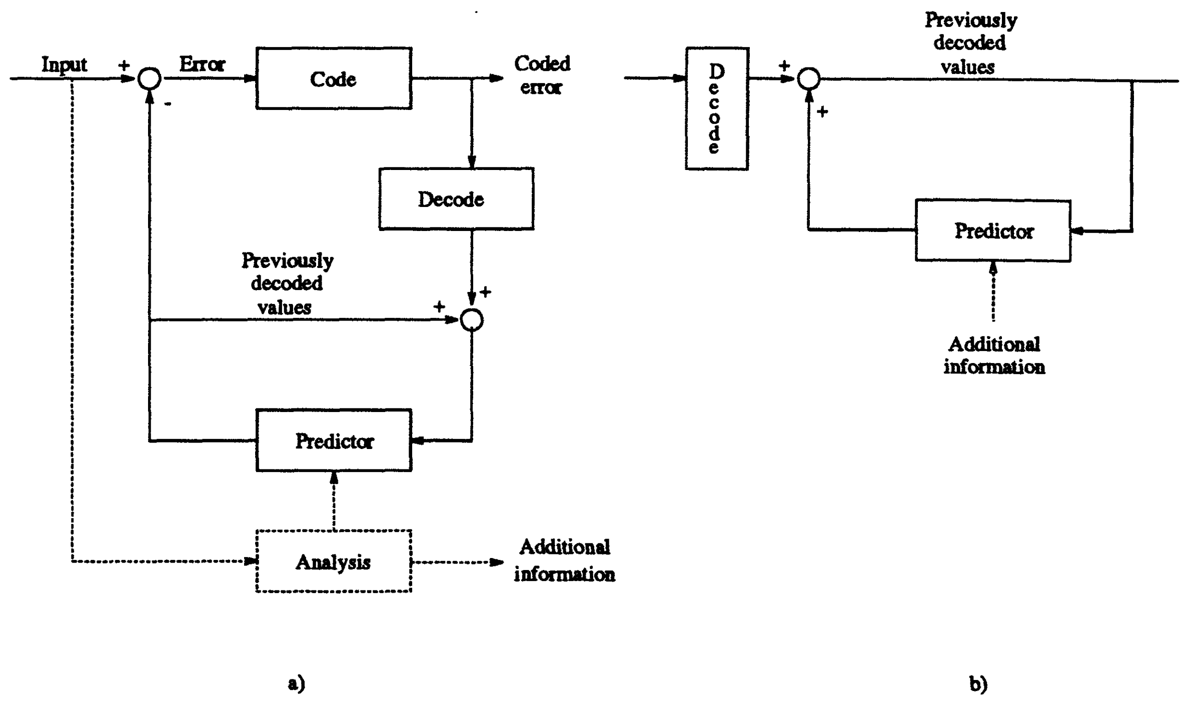


Figure 2-9: Block diagram of a predictive a) encoder and b) decoder.

translational. If we can obtain a vector field describing the motion from one frame to the next, then one frame, along with the vector motion field (which is typically source coded before transmission and decoded at the decoder), can be used to form a good prediction for other frames. This procedure uses the 'analysis' section as indicated in figure 2-9. Predictive coding schemes in general, such as *Differential Pulse Code Modulation (DPCM)* do not always use such analysis. In DPCM, the previous value, or a weighted sum of the previous few values is used as a predictor for the current value, and the difference between the prediction and the actual value is transmitted.

Estimating the motion of objects in a scene is a very complex one. A number of different approaches exist, including spatio-temporal constraint methods and region matching methods [21]. Spatio-temporal constraint methods attempt to derive some mathematical constraint on the pixel values of the image based on some image model — such as that all motion in a image is translational with uniform velocity, to take a simple case. Numerical methods for the solution of differential equations are then used to obtain a solution satisfying the constraint. The most commonly used methods in practice though, are region matching methods. These consider a small region in the image and attempt to estimate the motion of that region by finding a region in the next frame which is the 'best match' — based on some error criterion.

Segmenting the image into appropriate regions, though, is a non-trivial procedure and no 'optimal', or even 'correct' way of doing it is known. In practice, most region matching estimators simply segment the image into blocks (ignoring the content) of pixels. Typical block sizes are 8 by 8 or 16 by 16. *Block matching* consists of finding the closest match to a block from the blocks of the previous image. Though ad hoc, this method has the virtue of being conceptually simple and of working well with the other compression schemes described earlier.

In general, a small area of pre-specified size in the previous frame, usually referred to as the *search window* is used in searching for the 'best match'. The error expression to be minimized is typically the based on the MSE or *Mean Absolute Error (MAE)*. Under the MAE criterion, for example, block matching finds, for each block, the motion vector (d_x, d_y) which minimizes

$$\sum_{x,y \in \mathcal{R}} |I(x,y,t) - I(x - d_x, y - d_y, t-1)|, \tag{2.28}$$

where $I(x,y,t)$ is the intensity of frame t for the pixel coordinates (x,y), and $\mathcal{R}$ is the search window. This process is illustrated in figure 2-10.

Block matching is based on three assumptions:

- all the motion in the scene is translational and parallel to the plane of the camera.
- nearby pixels of the same object have equal motion.
- the size of revealed or occluded areas is small relative to the image size.

Since these assumptions hold approximately for most scenes (especially if the block size is small relative to the size of the objects in the scene, and the sequence has a reasonably high frame rate), block matching is a reasonably robust method for performing motion estimation.

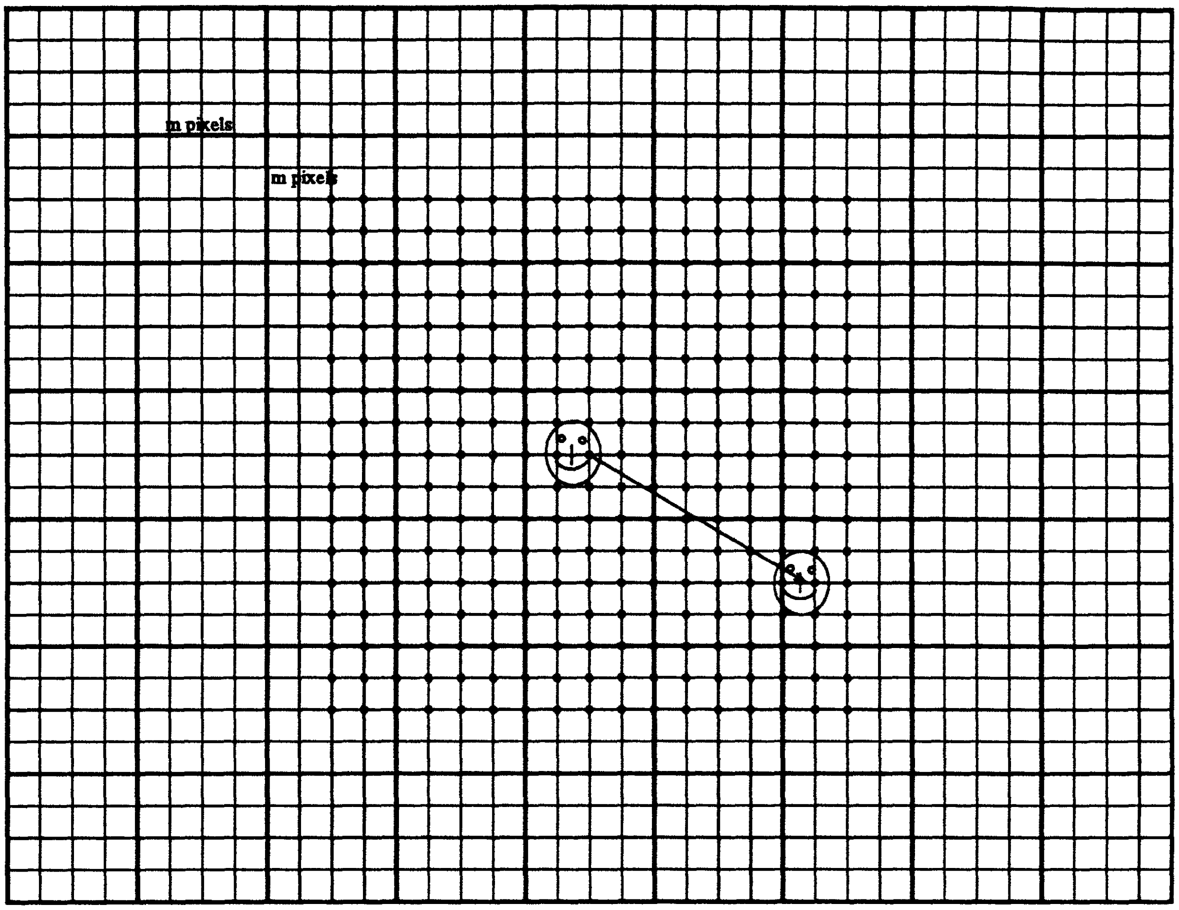


Figure 2-10: An illustration of the block matching procedure. The image has been segmented into blocks of m by m pixels. The block in the middle of the image in this figure belongs to a previous frame. This is shifted and centered around all the dots, which consitute the search window. The vector (d_x, d_y) (indicated by the arrow) corresponds to the 'best match' amongst the positions searched.

Since the error has to be calculated for every point in the search window, motion estimation by block matching is computationally a very expensive procedure — and it grows as $\mathcal{O}(n^2)$ as either the search window or the dimensions of the image are increased in size. The size of the search window depends on image characteristics — specifically the amount of motion in the scene under consideration, and the frame rate of the sequence. If the sequence has been shot at a low frame rate, then the amount of motion between frames will be large and a large search window will be needed.

A number of approaches have been tried to reduce the computational burden of the block matching. One commonly used method is to perform an n step search over the search window, instead of an exhaustive search as shown in figure 2-10. In the n step search we begin with doing block matching over n^2 equally spaced blocks spaced as widely apart as possible in the search window. We than do a similar search over a smaller search window centered around the best match from the first step, and so on. Though not as accurate as the exhaustive search, this method does yield reasonable results and is widely used in practice because of its reduced computational requirements. One thing to notice about the block matching problem in general is that it is very appropriate for parallel processing implementations — error expressions throughout the search window can all be separately and simultaneously computed in parallel, and this fact is beginning to be exploited in practice. As noted earlier, the high computational requirements for motion estimation by block matching for large image sizes was one of the motivations for this project.

The encoder in a coding scheme based on motion compensation periodically encodes frames without any motion estimation, using the methods described in earlier sections (this is known as *intra*frame coding), which is to be used for predicting other frames after decoding. For each of the frames to be coded using motion estimation (*inter*frame coding), the encoder transmits a set of coded *motion vectors* and the encoded error signal. The decoder uses the motion vectors and the error signal along with the previous intra frame to form the motion compensated frames. Although the description of the coding in this section has assumed that predictions may be formed only from previous frames, this need not be true. In fact the MPEG coder uses future frames (as well as previous ones) for prediction. This increases the storage requirements and coding delay, but enables the coder to form better estimates.

2.5 Lossless Coding

All the coding methods discussed so far have been based on some form of lossy quantization. For some applications, the distortion resulting from such lossy coding is not desirable, and a different form of compression, usually referred to as *lossless coding* is used. Lossless coding may also be applied after quantization, to exploit any redundancy in the quantized data.

Lossless coders perform a one to one mapping between the original data and the coded data, so that all the original data can be uniquely reconstructed and no information is lost in the process. They do so by exploiting the statistical properties of the data. The main idea is that more frequently occuring symbols are represented with a smaller codeword, while less frequently occuring symbols use a larger codeword. This reduces the average bitrate required. One feature of such a scheme is that it is impossible to know a priori (without knowing the data) the total number of bits which will be required to encode some data, and thus the average bitrate is not known. For this reason, lossless codes are also known as *Variable Length Codes (VLCs).*

A *source code* C for a random variable X^1 is a mapping from X, the range of X to D, the set of finite length strings of symbols from a D-ary alphabet. Let $\mathcal{X} = (\omega_1, \dots, \omega_N)$, p_i be the probability of each symbol ω_i, and l_i the length of each codeword. The theoretically optimal coder is the one which minimizes the average rate per symbol of the encoded output

$$b = \sum_{i=1}^{N} p_i l_i. \tag{2.29}$$

In addition to minimizing the bitrate, the optimal code also has to satisfy the requirement of *unique decodability.* One way to make a code uniquely decodable is to ensure that no codeword is a prefix of any other codeword. In this way there is no ambiguity about the decoded symbol. Such codes are known as *prefix codes.*

It can be shown [3] that any prefix code over an alpahabet of size D, the codeword lengths $l_1, \dots, l_N$ must satisfy the *Kraft Inequality*

$$\sum_{i=1}^{N} 2^{-l_i} \leq 1, \tag{2.30}$$

and, that if this inequality is satisfied, it is possible to construct a prefix code. Thus the optimal lossless code is one which minimizes equation 2.29 subject to the con-

[1] X might be, for example, a random variable describing the probability law for all the possible values at the output of a scalar quantizer

straint 2.30. We may use the method of Lagrange multipliers to show that 2.29 is minimized when the optimal codelengths are:

$$l_i^* = -log_D p_i. \tag{2.31}$$

This non-integer choice of codeword lengths yields the average bitrate:

$$b^* = H_D(X) = \sum_{i=1}^{N} p_i log_D \frac{1}{p_i}, \tag{2.32}$$

where $H_D(X)$ is known as the *entropy* of the source. In most cases, the codewords are constructed from a binary alphabet (0 and 1) and thus $D = 2$. Equation 2.32 states one of the fundamental results of information theory, first establised by Shannon [33]: the minimum possible rate achievable by lossless coding of a source is given by the entropy of that source.

Notice that equation 2.31 states that the entropy bound can be achieved only if $p_i = 2^{-l_i}$, (for $D = 2$). Now, if we code each symbol separately, the codeword length l_i must be an integer. This implies that the entropy bound can only be achieved if the symbol probabilities are a power of 2. Thus the entropy bound cannot in general be achieved for a scalar coder. However, it can be shown [3] that a code can always be constructed such that the average rate of that code is within one bit of the entropy bound, i.e:

$$H(X) \leq b \leq H(X) + 1, \tag{2.33}$$

where $H(X)$ is the entropy for $D = 2$.

If we code more than one symbol at a time, though, we can get closer to entropy. Suppose, we code K symbols together. Let the probability of each such K-vector be $\mathbf{p_i}$, and the codeword length associated with each symbol be $\mathbf{l_i}$. Then the average bitrate to be minimized is given by:

$$\mathbf{b} = \sum_{i=1}^{M} \mathbf{p_i l_i}, \tag{2.34}$$

Where M is the size of the symbol alphabet. Following the same analysis as before for the vector case, we find that

$$\mathbf{H(X)} \leq \mathbf{b} \leq \mathbf{H(X)} + 1. \tag{2.35}$$

If we assume that the K symbols forming the vector are i.i.d drawn from $\mathcal{X}$ according to p_i, then the *average bitrate per symbol* and the *entropy per symbol* are given by $\frac{b}{K}$ and $\frac{\mathbf{H(X)}}{\mathbf{K}}$. Thus,

$$H(X) \leq b \leq H(X) + \frac{1}{K} \tag{2.36}$$

Thus it is theoretically possible to achieve average bit rates arbitrarily close to the entropy of the source by coding larger and larger blocks of symbols.

2.5.1 Huffman Coding

The optimal prefix code for a given distribution can be constructed by a simple algorithm discovered by Huffman [12], [3]. The following is a description of the *Huffman coding* algorithm:

1. List all possible messages and consider these messages the leaves of a binary tree. Associate with each message its probability.

2. Pick the two messages with the least probability and create a parent node with a probabaility associated with it which is the sum of the two leaves. Label one of the branches with a one and the other with a zero.

3. If the probability associated with this node is one, go to step 4. Otherwise treat this node as a leaf and go to step 2.

4. Follow the path from the root to each leaf. The sequence of bits associated with the path to each message is the codeword associated with that message.

The encoder needs to transmit its model of the Huffman tree as well as the codewords to the decoder. The above algorithm depends on the fact that the probabaility distribution of the source is known. In practice this is seldom true. When Huffman coding typical image data, it is customary to first go through the data and construct a histogram of all the symbol occurences in order to obtain a probability mass function for the data. If such a procedure is not desirable for the application being considered (for example if it introduces too much coding delay), a procedure known as *Dynamic Huffman coding* [36] can be used. In Dynamic Huffman coding, the process begins with both the encoder and the decoder using a Huffman tree based based on an arbitrary probability distribution for the data. As both the encoder and the decoder receive symbols from the source, they modify their model the of the source's probability distribution and hence their Huffman trees in a known manner. As the length of the entire message approaches infinity, this procedure yields the correct Huffman tree and thus the optimal coding scheme.

As noted above, because of the restriction of using integer length codewords, the entropy bound can only be achieved if block coding is performed. Huffman coding yields suboptimal performance perticularly when the probabiltiy distribution is higly skewed. For example if one symbol has a probability of 0.9 associated with it, it should take no more that $log_2 0.9 = 0.15$ bits to encode it. However, we use one bit, which is more than six times as much to encode this bit, and since this symbol will occur frequently, good performance will not result. Block coding also presents a problem though, since the probability laws become more complicated and the size of the encoding table inceases exponentially with block length, increasing coding delay. For these reasons, another approach, known as *arithmetic coding* is often used.

2.5.2 Arithmetic Coding

Arithmetic coding is another form of lossless coding whose performance approaches the entropy bound. However, instead of assigning each symbol or group of symbols in the message a codeword as in Huffman coding, arithmetic coding assigns a unique codeword to the entire message. The restriction of integer length codewords per symbol is therefore done away with and, especially in cases where Huffman coding works poorly, arithmetic coding can more closely approach the entropy bound.

The basic algorithm for arithmetic coding is as follows:

1. Divide the interval [0,1) on the real line into a set of subintervals each of length equal to the probability of each symbol in the input alphabet. This is the main interval.

2. To code the next input symbol, choose the subinterval associated with that symbol. Make this the new main interval and divide it into subintervals according to the input probabilities, as in 1.

3. Indicate to the decoder the boundaries of the current main interval. If there are more symbols in the message, go to step 2. Othewise stop.

A sequence of symbols is thus translated into a successively smaller interval which *uniquely* identifies the input sequence. In fact, both ends of the interval need not be transmitted — any number inside the interval may be used to uniquely identify the input message. The compression is provided by the fact that symbols with higher probabilities have longer intervals associated with them, and it takes fewer bits to describe a number in a longer interval. Note that the decoder need not wait for the encoder to be done before it can start decoding. As the interval becomes narrower and narrower, more and more digits of a number that falls within this interval become set, and the decoder can use this information to begin decoding the sequence.

As an illustration of the efficacy of arithmetic coding consider an example with only two symbols: A 0 with probability 16382/16383 and an end of file character with probability 1/16383. Suppose a message has 100,000 0's and one end of file character. Using arithmetic coding, this message can be compressed using three bytes, while Huffman, or fixed length coding would take 12,501 bytes to compress this message [27]. Though obviously very contrived, this example does illustrate the kind of situation in which arithmetic coding provides a big win over Huffman coding.

In practice, implementations of arithmetic coding are not as simple as described above since the arithmetic precision required to code a message keeps growing with the length of the message. Thus, periodic renormalizations of the interval are required in real implementations. Recently, the Binary Arithmetic coder or *Q coder*, has proven to be very popular. The Q coder begins with converting the K letter alphabet into a string of binary values, perhaps by using some inefficient fixed length code. These values are then coded using arithmetic coding. One of the reasons why the Q coder is so popular is the simplicity of its implementation, since it involves binary arithmetic. Good explanations of arithmetic coding intricacies and implementation details are provided in [30] and [40].

2.5.3 Run Length Coding

Many types of data have long runs of the same value as a form of redundancy. For example two tone facsimile images have large regions of either black or white. Another example is the quantized pixel value data from the high subband of an image which does not have a lot of high resolution detail. In such a case, most of the pixel values will be zero, and a lot of runs of zeros will be expected. Run lengths are a particularly simple form of redundancy to compress — one needs to code only the value and the length of a run of each value.

It can be shown [16] that if the source can be modeled as a first order Markov process (i.e. if one value of the source depends only on the previous value), then the maximum compression that can be achieved by run length coding is equal to the reciprocal of $H(X)$, where $H(X)$ is the entropy of the source. This makes intuitive sense, since the more skewed a probabaility distribution is, the lower entropy. And a more skewed distribution implies longer runs — which allows run length coding to perform better.

In practice, run length coding is usually followed by a Huffman coder in order to exploit the remaining redundancy in the data. Using this combination it is possible to bypass the main weakness of Huffman coding alone — that it does not work well with highly skewed distributions beacuse of the integer length codeword constraint. As an illustration of this consider again the example at the end of the last section. A source has two symbols: 0 with probability 16382/16383 and an end of file character with probability 1/16383. A message with 100,000 0s and an end of file character can be run length coded simply by indicating a run of 100,000, which can be done in 4 bytes. So arithmetic coding no longer has a particular advantage over Huffman coding if the Huffman coder is preceded by a run length coder. Popat [31] cites evidence that this is particularly true at low bitrates (below 1.6 bits per sample for a gaussian source). However, at higher bitrates and for large alphabet sizes the codebook for run length coding becomes very large, since all possible run/magnitude pairs need to be accounted for. The conceptual and implementational simplicity of run length coding leads it to be often preferred over arithmetic coding at low bitrates.

For more on run length coding see Jayant and Noll [16] or Huang et. al. [26].

2.5.4 Entropy coded quantization

In section 2.1, the optimum scalar quantizer for a given pdf was presented (equation 2.7). Quantized data is often entropy coded. The reason for this is that using fixed length codes for transmitting quantized data (i.e. essentially performing no further coding) is optimal only for uniformly distributed data, and the output of a quantizer is often not uniformly distributed, even though that may be the aim of such a quantizer.

The introduction of variable length coding after quantization introduces a further design parameter into our problem of data compression with minimum distortion, however, and the analysis of section 2.1 is no longer valid in this case. It can be shown [16] that for a given distortion, the uniform quantizer will achieve the minimum entropy when quantization is followed by entropy coding (under the assumption of high bitrate coding). This result depends on the fact that we can (theoretically) have an arbitrarily large number of reconstruction levels. The reason why we can have a large number of reconstruction levels and still achieve a small average bitrate is that quantization intervals which are not very likely contribute very little to the average bitrate after being entropy coded. In fact we can (theoretically) have an infinite number of quantization levels and still have a finite average bitrate.

The advantage of using a uniform quantizer for such an *entropy coded quantizer* can be intuitively explained in this way. Suppose we have two quantizers $\mathcal{Q}_1$ and $\mathcal{Q}_2$. $\mathcal{Q}_1$ has been optimized to achieve minimum distortion according to equation 2.7, while $\mathcal{Q}_2$ is a uniform quantizer with an unlimited number of levels. Since $\mathcal{Q}_1$ minimizes distortion, quantization intervals are closer together in regions of high probability density and further apart in regions of low probability. Thus the total area under the pdf covered by each reconstruction level is approximately the same and the output of the quantizer will have approximately uniform probability distribution. For Q_2, though, reconstruction levels in regions of high probability will have a greater likelihood of occurence than levels from regions of low probability density. Since the more uniform a distribution, the higher its entropy, the output of Q_2 will have a lower entropy than the output of Q_1, which an intelligently designed entropy coder can exploit. The average distortion for Q_2 may be higher, but it may be decreased by increasing the number of quantization intervals. In addition, because of an unlimited (or in practice, large) number of levels, Q_2 does not suffer from overload distortion[2]. The result of all these effects is that Q_2 is able to achieve the average distortion of Q_1 with a lower codeword entropy.

For this reason uniform quantizers followed by entropy coders are a very popular component of practical image coding systems.

[2]this type of distortion results from the inability of a quantizer to cope with high values of input data. Such values are not very probable and are thus part of a large quantization interval, with a relatively low reconstruction value assigned to them

Chapter 3

The MPEG-2 Coding Standard

The MPEG (Motion Picture Experts Group) coding standard is a *generic* coding scheme designed to deal with digital video compression over a wide variety of applications. Generic means that the standard is independent of a particular application; it does not, however mean that it ignores the requirements of the applications. MPEG developed in response to the need for a standard in a field that was seeing a rapid growth in compression schemes for video for different applications. Storage media such as CD-ROMs, tape drives and writable optical drives as well as telecommunication channels such as ISDNs and local area networks were being used to carry digital video. The need for a standard was sorely felt in order to facilitate interoperability.

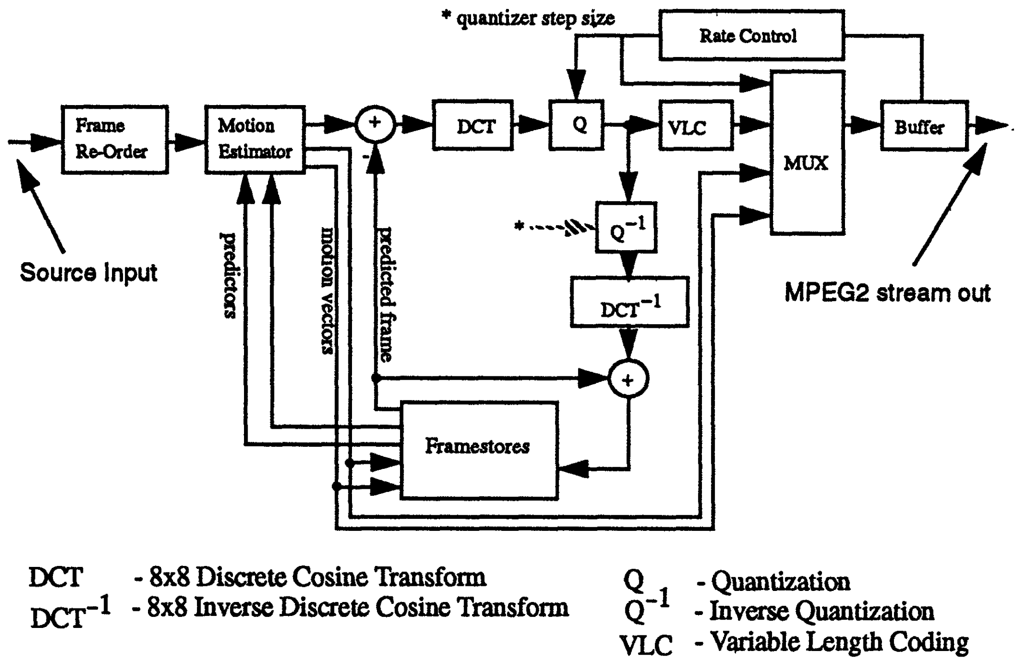
The original MPEG proposal was standardized by the ISO (International Organization for Standardization) in 1990 and was meant to address the compression of video signals at about 1.5 Mbits per second (including audio). It was developed with a number of desirable characteristics for such a generic coder in mind. These included the ability to perform random access, fast forward and reverse searches, reverse playback, audio visual synchronization, robustness to errors, low coding decoding delay for applications such a videotelephony, editability and the ability to perform real time encoding/decoding without too exorbitant a cost [7].

MPEG-2 is a modification of MPEG, in order to deal with a wider, and more demanding, variety of applications — most importantly HDTV and regular television broadcast over satellite and cable. It is meant to address video compression for rates upto about 40 Mbits. In addition, the MPEG-2 bitstream may be scalable A *base layer* carries a video signal at a low resolution (in terms of spatial or temporal resolution or SNR) which can be encoded and decoded by itself. This may be supplemented by an *enhancement layer*, which, when appropriately decoded and added to the base layer provides extra resolution.

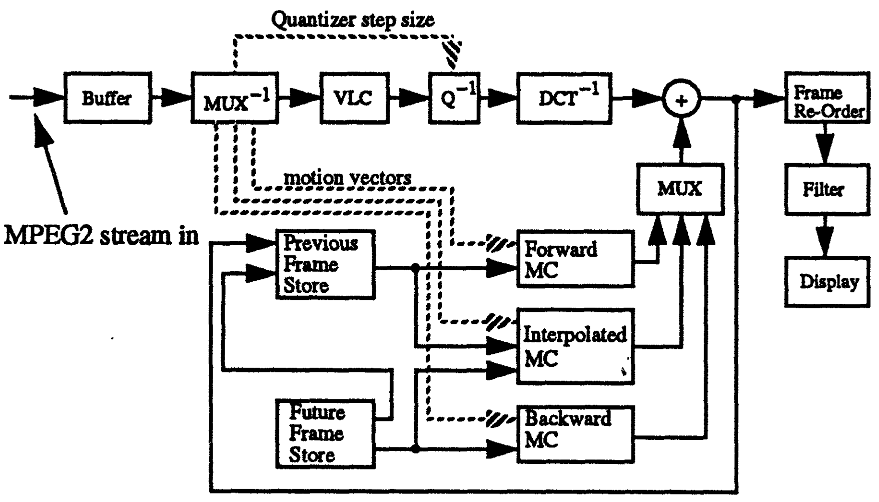
Since MPEG is meant to be a generic standard for a variety of applications it defines a *syntax* for the bitstream i.e. it defines the types of operations which are allowed under the standard, the sequence they must occur in and the type of data they can contribute to the bitstream. It leaves the implementations of the different operations to the discretion of the user, who may use different algorithms depending on the application and the state of the art in terms of technology. MPEG is organized as a set of different *profiles*. Each profile is composed of a subset of the syntax and supports a specific functionality. A profile can have different *levels*, which put constraints on the parameters and are usually associated with different resolution input formats (SIF, CCIR, 601, HDTV, etc.). The intent behind this organization was to have a measure of flexibility within the standard and to facilitate interoperability between different encoders and decoders. This chapter will describe the *main* profile [14] — the core of the codec around which the other profiles are built.

3.1 Overview of the Codec

As mentioned previously, the MPEG-2 algorithm uses motion compensation for temporal redundancy reduction and block based DCTs for spatial coding. The quantized data (DCT coefficients) is further compressed using variable length coding — MPEG outputs a variable length bitstream. Rate control is achieved by adaptively controlling the quantization step size. These main operations of the codec are summarized in the simplified block diagrams of the encoder and decoder in figure 3-1 a) and b) respectively.



a)



b)

Figure 3-1: Simplified block diagram of MPEG-2 codec: a) encoder, b) decoder. From [20].

3.2 Video data structures

A source or reconstructed picture in MPEG-2 consists of three rectangular matrices of integers. A luminance matrix (Y) and two chrominance matrices (Cb and Cr). The luminance and chrominance matrices are obtained from the RGB representation by applying a linear transform as defined in the CCIR 601 Recommendation [15], or by a transformation specified in the bitstream.

MPEG-2 represents video in a hierarchical manner as shown below

Sequence

{Group of Pictures}

Picture

Slice

Macroblock

Block

A sequence may consist of one or more groups of pictures (GOPs) or one or more pictures. The GOP layer is optional and consists of one or more pictures grouped together in a sequence meaningful for the motion compensator (to be discussed later). Pictures correspond to a single frame of the image sequence. The main function of the slice layer is to provide a means of resynchronizing the decoder after an error has occured in the bitstream.

Each slice consists of an arbitrary number of macroblocks in the same horizontal position. A macroblock is in turn made up of blocks. A block is an 8 by 8 pixel region of the picture. Each macroblock consists of four luminance (Y) blocks and a fixed number of chrominance blocks depending on the format of the video being coded (4:4:4, 4:2:2 or 4:2:0). Most applications use the 4:2:0 format in which the chrominance channels are both downsampled by a factor of two in either dimension. For the 4:2:0 format, each macroblock consists of 4 Y blocks and one block each of the Cb and Cr components, giving a total of 6 blocks per macroblock. In the 4:4:4 format, the chrominance components are not subsampled — this corresponds to 12 blocks per macroblock. In the 4:2:2 format, the chrominance components a sampled by 2 in the horizontal direction only, corresponding to 8 blocks per macroblock. The choice of format depends on the quality of input available and output desired. A macroblock that is not coded is referred to as a skipped macroblock. Each slice may begin and end only with a non-skipped macroblock. Figure 3-2 illustrates this structure.

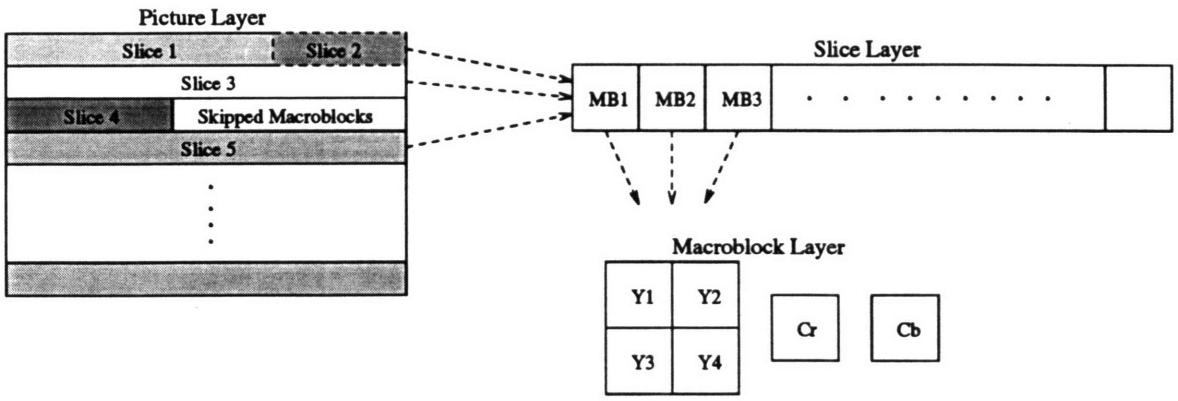


Figure 3-2: Spatial data structures within MPEG-2

MPEG-2 supports both progressive and interlaced sequences, and the type of coding algorithm depends on the input type. Two picture structures are defined for interlaced sequences: in *field pictures* each field is considered a separate entity, while *frame pictures*, obtained by merging two fields together in a single frame, each frame is considered a unique entity. Macroblocks in frame pictures contain both fields, while macroblocks in field pictures contain only one field. The way motion compensation and spatial processing is performed on an interlaced sequence varies according to whether the picture is of frame or field type.

3.3 Temporal Processing

There are three common picture types in the MPEG-2 bitstream:

- *Intra (I)* frames are coded without any temporal processing. Only Intraframe coding, based on DCTs is allowed.

- *Predictive (P)* frames are coded by motion compensation using a temporally previous frame, and

- *Bidirectional (B)* frames are coded by motion compensation using both a temporally previous frame and a frame in the future.

The ordering of these frames within a sequence is specified by two parameters: M and N. M describes the relative frequency of P frames. Every Mth frame that is not an I frame must be a P frame. If M = 1 then there are no B frames — every frame is a P frame except for the I frames, whose frequency is described by the N parameter. Every Nth frame is an I frame. A P frame can only be predicted from the previous I or P frame, and a B frame can only be predicted using the nearest I or P frames that follow or precede it. These features are demonstrated in figure 3-3, where M = 3, and N = 2.

A group of pictures (GOP) is typically an I frame followed by a succession of P and B frames before the next I frame. A GOP header is used to distinguish between GOPs, whose structure may be changed in the bitstream. GOPs are meant to facilitate random access, reverse playback and editability. Applications requiring these features will typically have short GOPs.

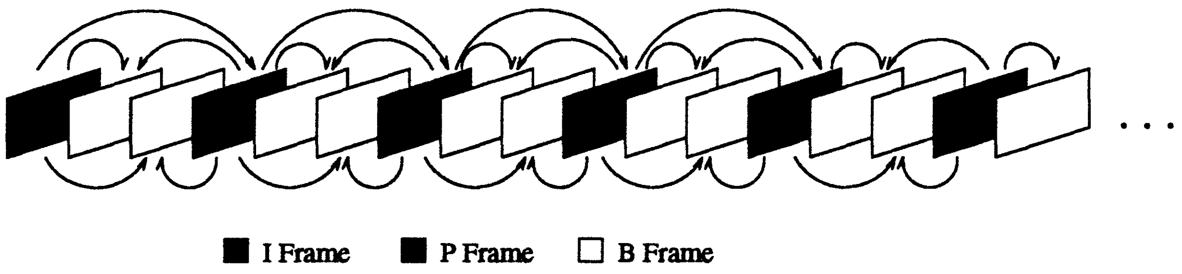


Figure 3-3: Example of a GOP structure with M = 3, N = 15. The arrows indicate the frames from which each P or B frame was predicted.

The non causal character of the B frames requires that a frame store (of M-1 frames) be kept for both encoding and decoding. The natural order of the frames (referred to as the *display order*) cannot be used for coding. A *coding order* requires that the the I and P frames which are being used to form predictions come before the P and B frames which will be predicted from them. For example, if the display order is:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
I	B	B	P	B	B	P	B	B	I	B	B	P	B	B	P	...

then the coding order is:

1	4	2	3	7	5	6	10	8	9	13	11	12	16	14	15	...
I	P	B	B	P	B	B	I	B	B	P	B	B	P	B	B	...

Fortunately, frame reordering may be achieved simply my moving pointers to frame buffers.

Predictions are formed by block matching (described in section 2.4) on the luminance component of each macroblock (i.e. a 16x16 block size is used), and the motion vectors so obtained are appropriately scaled (depending on the input format) and used for the chrominance blocks. Typically the error criterion used is the MAE 2.28, since it is the simplest and computationally cheapest to implement. The motion vectors are computed to half pel accuracy. This is done by predicting an extra pixel in between any two pixels in the image (being used to form a prediction) by simple linear interpolation from the immediately surrounding pixels, as demonstrated by the following equations.

$$\begin{aligned} P(x+0.5,y) &= (P(x,y)+P(x+1,y))/2 \\ P(x,y+0.5) &= (P(x,y)+P(x,y+1))/2 \\ P(x+0.5,y+0.5) &= (P(x,y)+P(x+1,y)+P(x,y+1)+P(x+1,y+1))/4 \end{aligned} \tag{3.1}$$

Once a motion vector has been computed to single pel accuracy using the traditional techniques, an exhaustive search can be carried out on the eight surrounding vectors within half a pel of the full pel motion vector.

The way in which predictions are formed depends on the picture type and one of the three prediction modes associated with each type. For frame-type pictures the three modes are,

- *frame mode*, in which motion vectors (one for P macroblocks, two for B macroblocks) are generated for the entire macroblock by block matching using the full 16x16 luminance block.

- *field mode*, in which a vector is generated separately for each field in the macroblock from the two most recently decoded fields. A motion vector may be generated from a field of the same parity or of different parity, whichever yields better results. Thus, two vectors per macroblock are generated for P macroblocks, and four for B macroblocks.

- *dual-prime mode* tries to combine the reduced overhead of frame mode prediction with the coding gain typical of B frames. One vector is generated per macroblock. However, each field in a macroblock is predicted from both fields of the prediction frame, and a single motion vector is created by interpolation between the vectors for the two fields. The motion vector thus derived indicates a pixel to be used for prediction of a particular pixel in a field of the same parity. This motion vector is then scaled, corrected in the vertical direction to reflect the vertical difference between fields of opposite parity, and added to a differential value of small amplitude (restricted to be 0 or ± 1) in order to point to a second pel in the field of opposite parity. These two pels are then averaged to form a prediction. This process is repeated in order to predict a value in the next field, using the same vector and differential value. Dual-prime mode is only used in P pictures when there are not B pictures in between the predicted and reference fields or frames.

Typically, the field and frame modes are combined into a *field/frame adaptive* mode, where the best of these two prediction methods is chosen at the macroblock level. The field mode is more efficient in areas of strong motion, where the two fields can be displaced by different amounts, whereas frame mode works more efficiently in less active areas, since it requires only half the number of vectors. Field type pictures have three modes as well.

- In *field mode* a vector is generated for each macroblock by searching over the same region in reference fields of both types (using a 16x16 block size), and using the best prediction. One vector per macroblock is generated for P type macroblocks and two for B type macroblocks.

- *16x8* mode partitions each block into an upper and a lower region and generates motion vectors for each sub-block in the same way as in field mode (except that it uses 16x8 blocks). The rationale behind this is that a 16x16 block in field pictures represents a 16x32 block in a real picture, and a 32 pixel height is too much for efficient motion compensation.

- In *dual-prime* mode, a macroblock from each reference field is used for predicting a macroblock. The two motion vectors so obtained are linearly interpolated to get a single motion vector. This motion vector, along with a differential value, is used to provide two motion vectors for predicting a macroblock from a macroblock of either parity by linear interpolation. As for frame pictures, dual prime mode is used only for P pictures when there are no intervening B pictures between the reference and the prediction.

The number of arithmetic operations required to compute a full pel motion vector per macroblock is proportional to the square of the range of the search window. Assuming that the search window is $\pm range$ pels in both width and height, the number of computations required is:

Possible number of vectors $= (2 \times range + 1)^2$
Number of arithmetic operations to calculate MAE/vector $= 2 \times 16^2$
Total number of operations $= 512 \times (2 \times range + 1)^2$

When more than one vector has to be generated per macroblock, the computational burden is increased. For example, in field/frame adaptive mode (for frame type pictures), three vectors need to be generated. In this case, however, we can use the fact that the total MAE per macroblock is the sum of the MAE between two odd fields and the MAE between two even fields shifted by the same amount. So for field/frame adaptive mode we need to perform four searches amongst 4 possible combinations reference field/predicted field pairs. The number of computations required is then:

Possible number of vectors = $4 \times (2 \times range + 1)^2$

Number of arithmetic operations to calculate MAE/vector = $2 \times 16 \times 8$

Total number of operations = $256 \times 4 \times (2 \times range + 1)^2$

In order to refine each computed full pel vector to half pel accuracy, three new points need to be calculated by interpolation for every point in the original frame. These three points require a total of five 2 point interpolations. Each 2 point interpolation consists of one addition and one division operation. Another eight macroblock size compares (involving calculation of the MAE) must then be performed. Thus:

Number of 2 point interpolations per macroblock = 5×16^2

Number of operations for half pel refinement/ frame vector = $8 \times 2 \times 16^2$

Number of operations for half pel refinement/pair of field vectors = $2 \times 8 \times 2 \times 16 \times 8$

Number of operations for half pel refinement/3 field/frame vectors = $2 \times 8 \times 16^2$,

Once the required motion vectors have been calculated for a particular macroblock, it remains to be decided which vector(s) give the best prediction and whether this prediction is sufficiently close to be useful. The best vector(s) are chosen in terms of whichever one minimizes the chosen error criterion. This requires only a few additions and compares per macroblock. Several *macroblock types* are possible, depending on the frame type.

- P frame macroblocks can be predicted from a past frame (*forward predicted macroblocks*), or not predicted at all (*intraframe macroblocks*).
- B frame macroblocks can be predicted from a past frame (*forward prediction*), from a future frame (*backward prediction*), from an interpolation from both past

and future frames (*interpolative prediction*), or not predicted at all)*intraframe mode*).

Having chosen the best vector, a decision needs to be made about whether transmitting the motion vectors will result in a coding gain if a motion vector set to zero is used for prediction. This decision is made according to figure 3-4, used for a coder using the MAE error criterion [20].

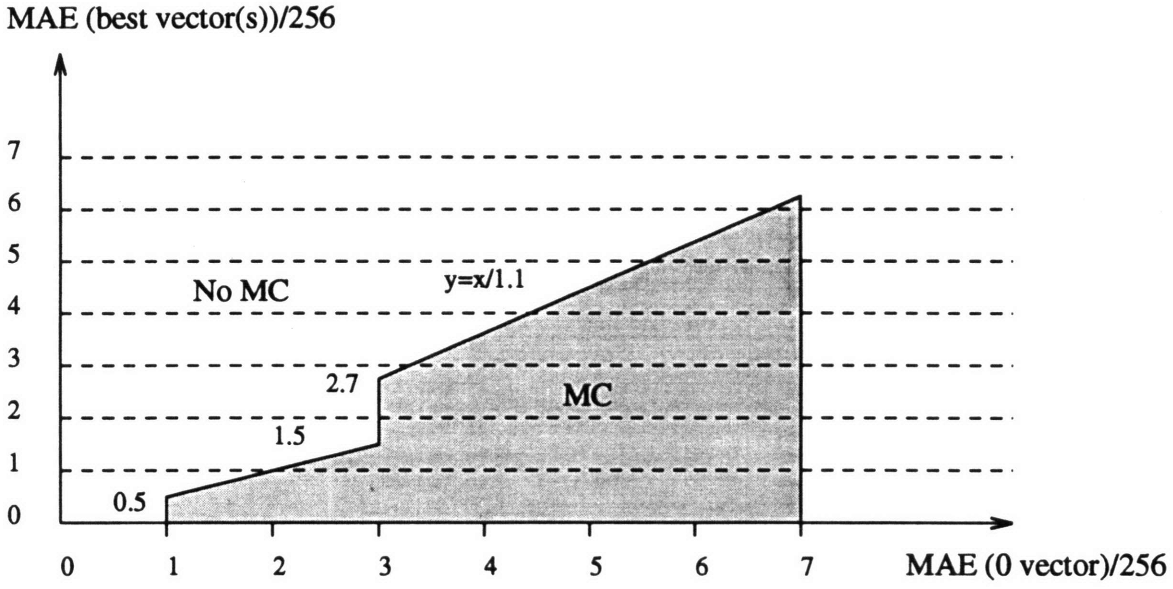


Figure 3-4: Motion Compensated or No Motion Compensation decision characteristic.

Deciding whether the best motion vectors calculated above is/are better than just coding the block in INTRA mode (using only intraframe prediction) is done by comparing the variance of the residual luminance component and the variance of the luminance of the original macroblock, using the characteristic given in figure 3-5.

Having decided the type of the macroblock to be transmitted, the macroblock is subjected to a DCT and then quantized.

Motion compensation at the decoder consists of decoding the macroblock vectors and then using them as offsets for macroblocks from the previously decoded frames into a prediction store. The dequantized residual values are then added to the prediction to form the decoded image. Thus, in terms of arithmetic operations, motion compensation involves 64 additions per block.

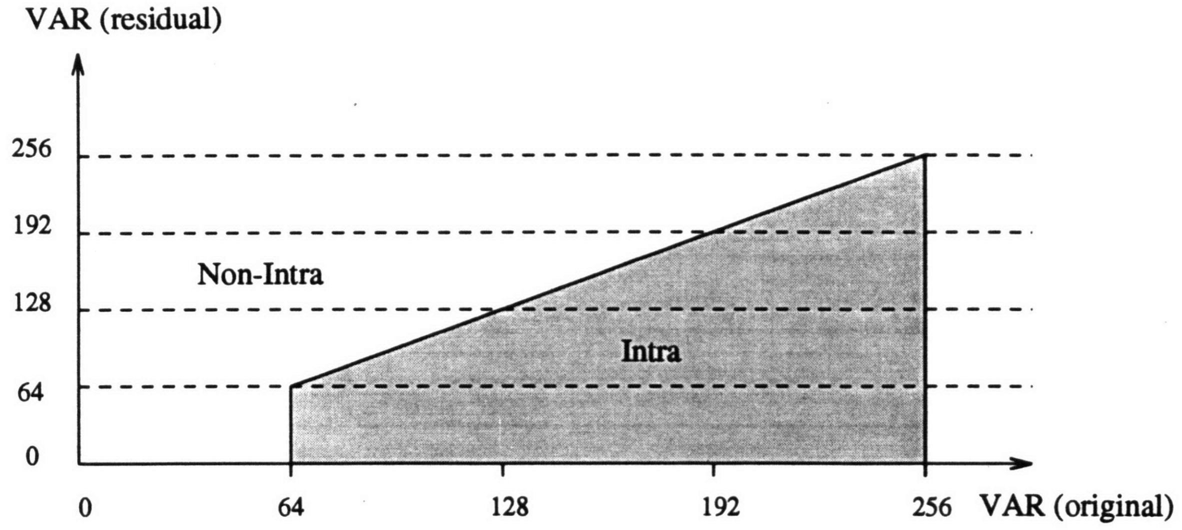


Figure 3-5: Intra / Non-intra coding decision.

3.4 Spatial Processing

After motion compensation has been performed, spatial redundancy within a frame is minimized by using the 8x8 block DCTs on each of the luminance and chrominance blocks. The DCT is applied to the pixel values directly for Intra macroblocks, while the motion compensated macroblocks (P and B types) have DCTs applied to the residual data (the difference between the prediction and the actual data).

There are two different modes for DCT coding:

- In *frame DCT mode*, each block contains lines from both fields interleaved together and the DCT is applied without any consideration of the interlaced nature of the macroblock.
- In *Field DCT mode*, lines from each field are used to form different blocks, and an 8x8 DCT is performed on these.

Both these procedures are explained in figure 3-6.

Field type DCTs are more effective when there is a lot of motion between fields and significant vertical detail. This causes vertical edges to appear serrated when fields are interleaved, introducing high vertical frequencies which are difficult to code. Frame DCTs are more efficient when there is not a lot of motion or vertical detail,

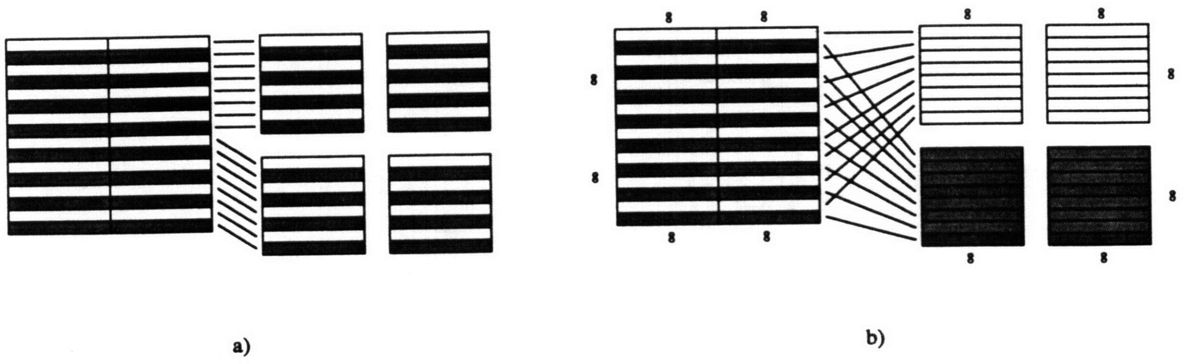


Figure 3-6: Macroblock structure for a) frame DCT, b) field DCT.

since the higher vertical resolution allows us to compress the vertical redundancy more effectively. The decision about which DCT mode to use is based on comparing the total error between successive lines in a macroblock with the total error between every other line in a macroblock. If the former is higher, it indicates a large inter-field error, and field mode is used.

We can take advantage of the separable nature of the DCT and employ row column decompositions (see section 2.2) to reduce the computational requirements of taking a DCT or IDCT. We can deduce from looking at equations 2.15 and 2.17 that direct computation would require (for a N by N block) $N^2(N^2-1)$ additions and N^4 multiplications. However, using row column decomposition, we need to perform N 1D N point DCTs (along the rows) followed by N 1D N point DCTs (along the columns). Since each 1D N point DCT takes about N^2 multiplications and $N(N-1)$ additions, we need, for each $N \times N$ (or 8×8) block:

$$\text{Number of multiplications} = 2 \times N^3 = 2 \times 8^3$$
$$\text{Number of additions} = 2 \times (N-1)N^2 = 2 \times 7 \times 8^2$$

Computation of the IDCT simply uses a different set of basis functions and uses the same number of operations per block. Direct computation of the 1D DCT/IDCT has been used in our implementation. More efficient implementations, using FFT (Fast Fourier Transfrom) algorithms to derive the DCT (there is a simple relationship between N point DFTs and DCTs) are possible [21].

After application of the DCT, the transformed coefficients are quantized. Varying the quantization parameters is the main method of achieving rate control within MPEG-2. A larger quantization step size results in more information being lost and degradation of image quality, but the picture can be coded using fewer bits.

Quantization of transform coefficients is a two step procedure. First a prespecified quantization weighting matrix, common to all blocks in a frame is applied to the block coefficients. Each modified coefficient is then scalar quantized, with a quantization step size that is determined by the local image activity and the available transmission rate.

Typical quantization matrices weigh the coefficients unevenly to reflect both the subjective (visual) and objective (in terms of total amount of energy represented) importance of the lower frequency coefficients. They also provide an easy way to perform threshold coding — any coefficient of amplitude less than the quantizer step size is automatically zero, and by dividing high frequency components by a large number, quantization matrices increase the effective step size of the quantizer for these components. The quantization matrix is different for intra and non-intra macroblocks. This is due to the fact that non-intra macroblocks contain residual energy, which is highpass in nature, and both the subjective and objective importance of the resulting DCT coefficients is roughly equivalent. Quantizing the higher frequency coefficients much more coarsely would throw away a lot more of the macroblock energy for the non-intra case than for the intra case. The default quantization matrices for the two cases are shown in figure 3-7. These matrices can be changed and specified in the bitstream.

The next step in the quantization process is to find an appropriate quantization step size. This is done by setting the value of the quantization parameter *mquant*, which is done in three steps [1].

1. Before a frame is coded, a target bitrate is set by computing the number of bits

[1]For this project the rate control mechanism described in the MPEG Test Model 5 [2] was implemented. As far as the MPEG syntax is concerned we may substitute any method for determining *mquant*.

$w_I(i,j)$							
8	16	19	22	26	27	29	34
16	16	22	24	27	29	34	37
19	22	26	27	29	34	34	38
22	22	26	27	29	34	37	40
22	26	27	29	32	35	40	48
26	27	29	32	35	40	48	58
26	27	29	34	38	46	56	69
27	29	35	38	46	56	69	83

a)

$w_N(i,j)$							
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16

b)

Figure 3-7: Quantization matrix for a) intra, b) non-intra macroblocks.

available for the frame. This is done by means of a set of formulas that attempt to keep the ratio of bits allocated to I, P and B frames approximately constant.

2. By means of a 'buffer fullness' of a 'virtual buffer' a reference quantization value is calculated per macroblock. The buffer fullness is a measure of the deviance of the coder from the target bitrate per macroblock, and this portion might be viewed as a kind of feedback loop. The macroblock to be coded is assigned a large quantization step size if the previous macroblocks have been taking more bits than they should have taken to maintain the target bitrate calculated in step 1.

3. The reference value of the quantization parameter calculated in step 2 is modulated according to the spatial activity in the macroblock. If the variance of the macroblock pixels is high, then presumably the macroblock contains a lot of detail that can mask noise — allowing the coder to increase the quantization parameter. The activity measure, based on variance, is normalized with respect to the average activity measure in the previous frame. This normalized measure is multiplied with the reference parameter obtained in step 2 to yield the value of *mquant*.

The exact equations for determination of *mquant* are given in [2].

The main computational burden incurred in doing the rate control is in computing the variance of the macroblocks. According to the TM5 specifications, the variance over 8 luminance blocks per macroblock is calculated and the minimum variance is used for the final value. The 8 luminance blocks are the 4 luminance blocks treated as if they were part of a field picture (i.e. with different fields in different blocks) and the 4 blocks treated as if they were part of a frame picture (i.e with two fields interleaved together in the same block). For each block, calculation of the mean is done first. This requires 63 additions and 1 division. Calculation of the variance requires a further 64 multiplications, 127 addition/subtractions, and 1 division per block. In addition, another 4 additions/subtractions, 5 divisions, and 4 multiplications are required per macroblock. Thus for rate control, the total number of operations (excluding a small overhead of operations per frame) is:

Function	Multiplications	Divisions	Additions
Rate Control	516	13	1020

Since the DC transform coefficient (which represents the average value) is extremely important both subjectively and objectively, it is quantized using a fixed scalar quantizer, whose step size may be 8, 4 or 2, according to the desired accuracy specified in the bitstream. For intra blocks, the AC coefficients are quantized using *mquant* (which is calculated for every macroblock) according to the following equations.

$$\begin{aligned} QDC &= dc//\{8,4,2\}, \qquad \text{depending on the required accuracy,} \\ ac\sim(i,j) &= (16 \times ac(i,j))//w_I(i,j), \\ QAC(i,j) &= [ac\sim(i,j) + sign(ac\sim(i,j)) \times ((p \times mquant)//q)]/(2 \times mquant), \end{aligned} \tag{3.2}$$

Here QDC and QAC are the quantized dc and the quantized ac coefficients at position (i,j) respectively, and $//$ represents division followed by rounding off to the nearest integer. The two parameters p and q are fixed at 3 and 4 respectively for this implementation, w_I is the intra quantization matrix whose default value is specified in figure 3-7, and

$$sign(x) = \begin{cases} +1, & \text{if } x > 0 \\ -1, & \text{if } x < 0 \\ 0, & \text{if } x = 0. \end{cases} \tag{3.3}$$

The quantized coefficients are reconstructed according to:

$$\begin{aligned} recon\_dc &= QDC \times 8, 4, 2 \\ recon\_ac(i,j) &= [2 \times mquant \times mquant \times QAC(i,j) \times w_I(i,j)]/16 \end{aligned} \tag{3.4}$$

A similar procedure is followed for quantizing non-intra coefficients. Specifically, quantization is performed according to:

$$\begin{aligned} ac \sim (i,j) &= (16 \times ac(i,j))//w_N(i,j), \\ QAC(i,j) &= ac \sim (i,j)/(2 \times mquant). \end{aligned} \tag{3.5}$$

Here $ac(i,j)$ refers to all the coefficients, including the dc value. w_N is the non-intra quantization matrix.

Dequantization is then performed according to the following equation.

$$\begin{aligned} recon\_ac &= [(2 \times QAC(i,j) + sign(QAC(i,j)) \times mquant \times w_N(i,j)]/16, \\ recon\_block(i,j) &= \begin{cases} recon_a c(i,j), & \text{if } QAC(i,j) \neq 0 \\ 0, & \text{if } QAC(i,j) = 0 \end{cases} \end{aligned} \tag{3.6}$$

Note that the last of the above equations implies that the quantizer has a deadzone i.e. that a larger than usual range of values around zero get quantized to zero.

Based on these expressions for quantization/inverse quantization, the number of arithmetic operations required per macroblock are given below.

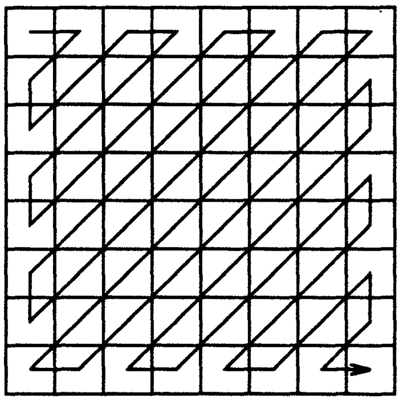
Function	Multiplications	Divisions	Additions
Intra Quantization	128	128	63
Non-intra Quantization	65	128	0
Intra Dequantization	129	63	0
Non-intra Dequantization	192	64	64

3.5 Variable Length Coding

After quantization has been performed on the transformed macroblock, the non-zero coefficients, along with the motion vectors, quantizer and other parameters are entropy coded to further increase the compression. The vlc encoded data is then incorporated into the bitstream according to the MPEG-2 syntax. To reduce overhead, the contents of each block are examined for non-zero coefficients. If none are found the the block is deemed *non-coded*, otherwise it is classified as *coded*. If a block is deemed to have been coded then the quantizer value is examined to see if it has changed from the previous macroblock. If it has then the quantizer must be resent. To indicate the blocks that are non-coded, and thus not sent, a *coded block pattern* needs to be transmitted for each macroblock

Since neighbouring blocks typically have similar means, the dc coefficients from the DCT are coded using DPCM — the value of the previous block's dc component is used as a predictor for the current dc value and the difference is losslessly coded according to a prespecified table in the MPEG-2 specifications.

In coding the ac coefficients, the coder takes advantage of the fact that neighbouring coefficient values have similar amplitudes by using an intelligently chosen scanning order to encode the coefficients. The lower frequency coefficients have relatively high amplitudes, and many of the higher frequency components are zeroed out. Zero runs amongst the high frequency components are fairly common, and this fact is exploited by employing run length coding. Two scanning orders are specified: *zig zag scan* and *alternate scan*. These are shown in figure 3-8. Zig zag scan is the default



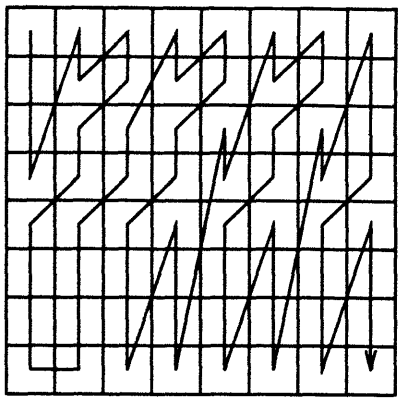


Figure 3-8: Scanning orders for DCT coefficient coding: a) Zig Zag Scan, and b) Alternate Scan.

scanning order and works well with progressive pictures, while alternate scan works better with interlaced pictures. The scanned vector of coefficients is then run length coded. MPEG-2 specifies a table of vlc's for commonly occuring run-amplitude pairs. Uncommon run-amplitude pairs are escape coded — i.e. an escape symbol followed by fixed length coded run and amplitude values is used.

Motion vectors are also differentially coded. Four separate predictors are kept, which are used to form the four different types of predictions that may be required for each of the modes in each picture type. In Test Model 5, the following rules are obeyed for encoding motion vectors:

- Every forward and backward motion vector is coded relative to the last vector of the same type. Each component of the vector is coded independently, the horizontal component first and then the vertical component.

- The prediction motion vector is set to zero in the macroblocks at the start of a macroblock slice, or if the last macroblock was coded in intra mode. A no motion compensation decision (according to figure 3-4 also corresponds to setting setting a predictor to zero.

- Only vectors used for the selected prediction mode are coded. Only vectors that have been coded are used as predictors.

Vectors are coded in raster scan order. The differential components are then variable length coded using a vlc table derived from statistical experimentation. Tables of vlc's are also used to encode other MPEG parameters and variables, such as coded block pattern, or macroblock type [2].

3.6 Summary of Computational Requirements

Tabulated in this section is a summary of computational requirements for encoding CCIR 601 (480×704) and HDTV (720×1056) resolution pictures using MPEG-2. For these calculations, it is assumed that 4:2:0 input format, and field/frame adaptive motion compensation is being used. The figures presented here are for one frame.

In addition to arithmetic operations, memory copies for the purpose of motion estimation, half pel buffer construction, motion compensation etc. will also be required and these will take time to do. No computational requirements are presented for variable length coding. It is assumed that this can mostly be done using table lookups.

HDTV Resolution

60 frames per second	
upto 45×66 macroblocks per frame	$\Rightarrow$ 178,200 macroblocks/second
6 blocks per macroblock	$\Rightarrow$ 1,069,200 blocks/second

CCIR 601 Resolution

30 frames per second	
upto 30×44 macroblocks per frame	$\Rightarrow$ 39,600 macroblocks/second
6 blocks per macroblock	$\Rightarrow$ 237,600 blocks/second

[2]This parameter in the MPEG-2 stream, in addition to specifying the type of motion prediction, also contains information about whether the macroblock in coded or non-coded and whether the quantizer needs to be reset.

HDTV Resolution

Functional Block	**Mult/Div**	**Add/Sub**
Frame Reorder (encoder)		
Frame Reorder (decoder)		
Motion Estimation (full pel, full search, *range* = 15)		2,922,670,080
Motion Estimation (full pel, full search, *range* = 45)		25,184,839,680
Half Pel Buffer Construction	3,801,600	3,801,600
Half Pel Refinement		24,330,240
Motion Compensation		1,140,480
DCT	18,247,680	15,966,720
IDCT	18,247,680	15,966,720
Quantization (Intra)	4,561,920	1,122,660
Quantization (Non-intra)	3,439,260	
Dequantization (Intra)	3,403,620	
Dequantization (Non-intra)	4,561,920	1,140,480
Rate Control	1,571,130	3,029,400
VLC		
VLD		

CCIR 601 Resolution

Functional Block	Mult/Div	Add/Sub
Frame Reorder (encoder)		
Frame Reorder (decoder)		
Motion Estimation (full pel, full search, *range* = 15)		1,298,964,480
Motion Estimation (full pel, full search, *range* = 45)		11,193,262,080
Half Pel Buffer Construction	1,689,600	1,689,600
Half Pel Refinement		10,813,440
Motion Compensation		506,880
DCT	8,110,080	7,096,320
IDCT	8,110,080	7,096,320
Quantization (Intra)	2,027,520	498,960
Quantization (Non-intra)	1,528,560	
Dequantization (Intra)	1,512,720	
Dequantization (Non-intra)	2,027,520	506,880
Rate Control	698,280	1,346,400
VLC		
VLD		

HDTV Resolution

	Mult/Div	Add/Sub
I Frame Encode		
Half Pel Buffer Construction	3,801,600	3,801,600
DCT	18,247,680	15,966,720
Quantization (Intra)	4,561,920	1,122,660
Rate Control	1,571,130	3,029,400
VLC		
P Frame Encode		
Motion Estimation (full pel, full search, *range* = 15)		2,922,670,080
Half Pel Refinement		24,330,240
Half Pel Buffer Construction	3,801,600	3,801,600
Motion Compensation		1,140,480
DCT	18,247,680	15,966,720
Quantization (Non-intra)	3,439,260	
Rate Control	1,571,130	3,029,400
VLC		
B Frame Encode		
Fwd. Motion Estimation (full pel, full search, *range* = 15)		2,922,670,080
Fwd. Half Pel Refinement		24,330,240
Bwd. Motion Estimation (full pel, full search, *range* = 15)		2,922,670,080
Bwd. Half Pel Refinement		
DCT	18,247,680	15,966,720
Quantization (Non-intra)	3,439,260	
Rate Control	1,571,130	3,029,400
VLC		

I Frame Decode		
Half Pel Buffer Construction	3,801,600	3,801,600
IDCT	18,247,680	15,966,720
Dequantization (Intra)	3,403,620	
VLD		
P Frame Decode		
Half Pel Buffer Construction	3,801,600	3,801,600
Motion Compensation		1,140,480
IDCT	18,247,680	15,966,720
Dequantization (Non-intra)	4,561,920	1,140,480
VLD		
B Frame Decode		
Motion Compensation		1,140,480
IDCT	18,247,680	15,966,720
Dequantization (Non-intra)	4,561,920	1,140,480
VLD		

CCIR 601 Resolution

	Mult/Div	Add/Sub
I Frame Encode		
Half Pel Buffer Construction	1,689,600	1,689,600
DCT	8,110,080	7,096,320
Quantization (Intra)	2,027,520	498,960
Rate Control	698,280	1,346,400
VLC		
P Frame Encode		
Motion Estimation (full pel, full search, *range* = 15)		1,298,964,480
Half Pel Refinement		10,813,440
Half Pel Buffer Construction	1,689,600	1,689,600
Motion Compensation		506,880
DCT	8,110,080	7,096,320
Quantization (Non-intra)	1,528,560	
Rate Control	698,280	1,346,400
VLC		
B Frame Encode		
Fwd. Motion Estimation (full pel, full search, *range* = 15)		1,298,964,480
Fwd. Half Pel Refinement		10,813,440
Bwd. Motion Estimation (full pel, full search, *range* = 15)		1,298,964,480
Bwd. Half Pel Refinement		10,813,440
DCT	8,110,080	7,096,320
Quantization (Non-intra)	3,439,260	
Rate Control	698,280	1,346,400
VLC		

<u>I Frame Decode</u>		
Half Pel Buffer Construction	1,689,600	1,689,600
IDCT	8,110,080	7,096,320
Dequantization (Intra)	1,512,720	
VLD		
<u>P Frame Decode</u>		
Half Pel Buffer Construction	1,689,600	1,689,600
Motion Compensation		506,880
IDCT	8,110,080	7,096,320
Dequantization (Non-intra)	2,027,520	506,880
VLD		
<u>B Frame Decode</u>		
Motion Compensation		506,880
IDCT	8,110,080	7,096,320
Dequantization (Non-intra)	2,027,520	506,880
VLD		

Chapter 4

Structure of the Coders

Two structures for the coders were suggested and implemented in this project. This chapter will describe the details of their implementations and will discuss the performance issues that were considered while designing the coders. In order to facilitate comparisons between these coding schemes and others in terms of computational complexity, the number of arithmetic operations required for the main components of the coders will also be presented. The figures presented are for the luminance portion of the image.

4.1 Overview

We will begin with a brief overview of the two coding structures and then consider the design details at every step. For the purposes of this thesis, we will assume that the lower layer image is two thirds the spatial resolution of the full resolution HDTV image. Assuming that we have sampled using the appropriate anti-aliasing filters [28], (see figure 2-8 for the effects of downsampling. Anti-aliasing filters are a necessary feature of the coding scheme, since aliasing causes very unpleasant image artifacts), only the lower portion (the lower third, or two thirds, depending on the type of downsampling employed) of the total image spectrum is available for coding at the lower layer. Thus, it is the higher frequencies of the spatial frequency spectrum in either direction that we are mainly interested in coding at the enhancement layer.

This makes subband coding using a nine subband structure the natural choice for coding the high resolution enhancement image.

4.1.1 The Open Loop coder

The openloop coder works on the premise that the MPEG-2 coder does a very good job of coding the lower resolution image. Specifically, we assume that the MPEG-2 coder does such a good job that not only is the low resolution image perceptually flawless, but when the low resolution image is interpolated back to the higher resolution, the higher resolution image does not reveal any of the MPEG-2 coding errors either. The only visually significant missing information is that contained in the high spatial frequencies of the original high resolution image.

When the low resolution sequence is in interlaced format, however, it is not simply the spatial high frequencies that are missing. The region of support for the spectrum of an intelligently derived interlaced sequence is diamond shaped in the time/vertical frequency domain [19], [29]. Thus more of the high spatial frequencies from the temporal high frequency region are missing from the original spectrum, and need to be compensated for. Thus an eighteen subband structure must be used for effective coding of the enhancement signal in the openloop structure when the base signal is interlaced. This is pictorially explained in the frequency domain in figure 4-1.

For reasons to be discussed later, in the experiments performed for this project the base layer did not have the ideal region of support shown in figure 4-1. Instead, only the bottom four subbands (vertical low/horizontal low and middle bands in both temporal bands) were fully available to MPEG-2.

Figure 4-2 is a block diagram of the openloop encoder. We begin by downsampling and interlacing (if necessary) the original image to derive a base layer image, which is given to the MPEG-2 encoder. Simultaneously, we split frame pairs from the original image into two bands of temporally high and low components. The X'ed subbands are then derived from these bands by subband decomposition (filtering followed by downsampling). The selected subbands are then coded, with bits allocated in proportion to the visual importance and the amount of signal energy in each

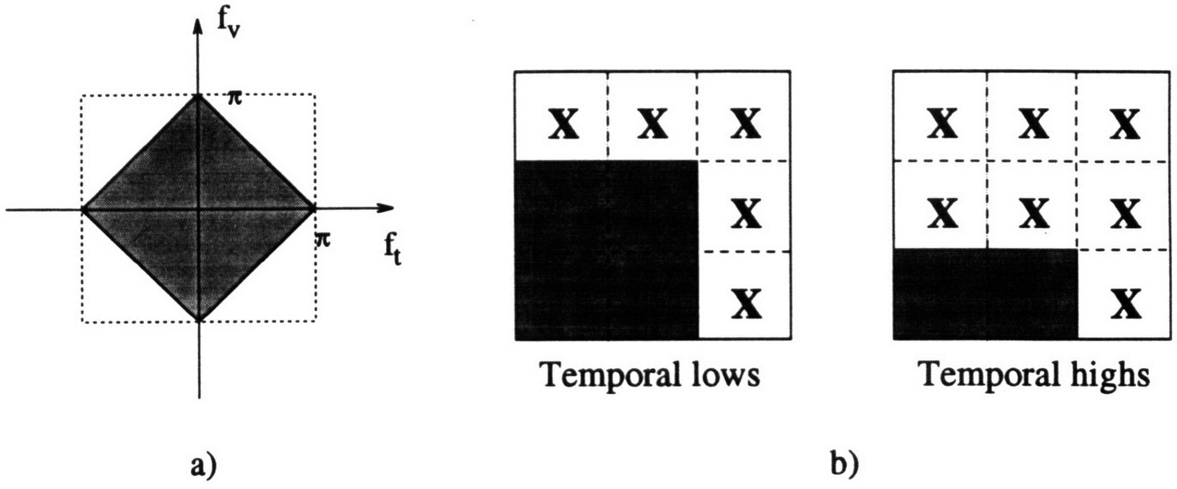


Figure 4-1: (a) Region of support for an interlaced sequence in the temporal/vertical frequency domain, (b) the shaded subbands from the spectrum of the original high resolution image are coded by MPEG. The X'ed subbands are not.

subband.

The decoding process is straightforward. The MPEG-2 decoder converts the bitstream it receives into pictures, which are then interpolated to full size. Simultaneously, the subband decoder converts the enhancement layer bitstream into decoded subbands. The subbands are added together to give the decoded residues of the two temporal bands, and converted from temporal bands to frame pairs. The decoded enhancement frames are then added to the upsampled MPEG-2 decoded images to obtain the final image. This process is illustrated in figure 4-3.

The openloop coder is meant for applications where the low resolution image is required to be of a very high quality, and where a correspondingly large and reliable channel is available to carry the base resolution signal.

4.1.2 The Closed Loop coder

The closedloop coder trades off increased computational complexity and robustness with an increase in computational requirements. In this case, it is not the original sequence which is subject to subband analysis. Instead, the MPEG-2 decoded signal is upsampled and subtracted from the original, and it is this difference signal which is subband coded as the enhancement layer. This structure may be thus be viewed as

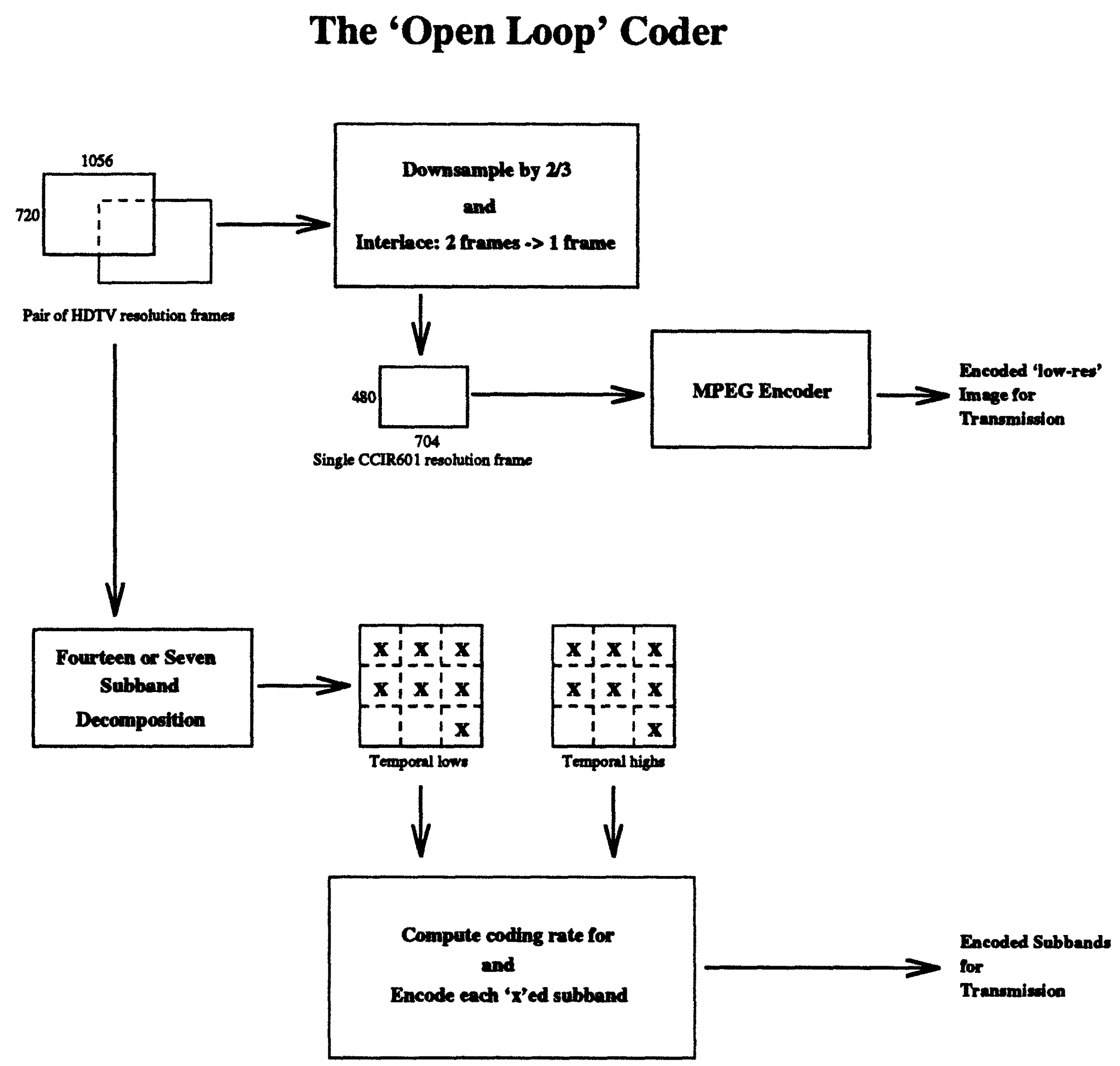


Figure 4-2: Block diagram of the 'Open Loop' encoder.

The 'Open Loop' Decoder

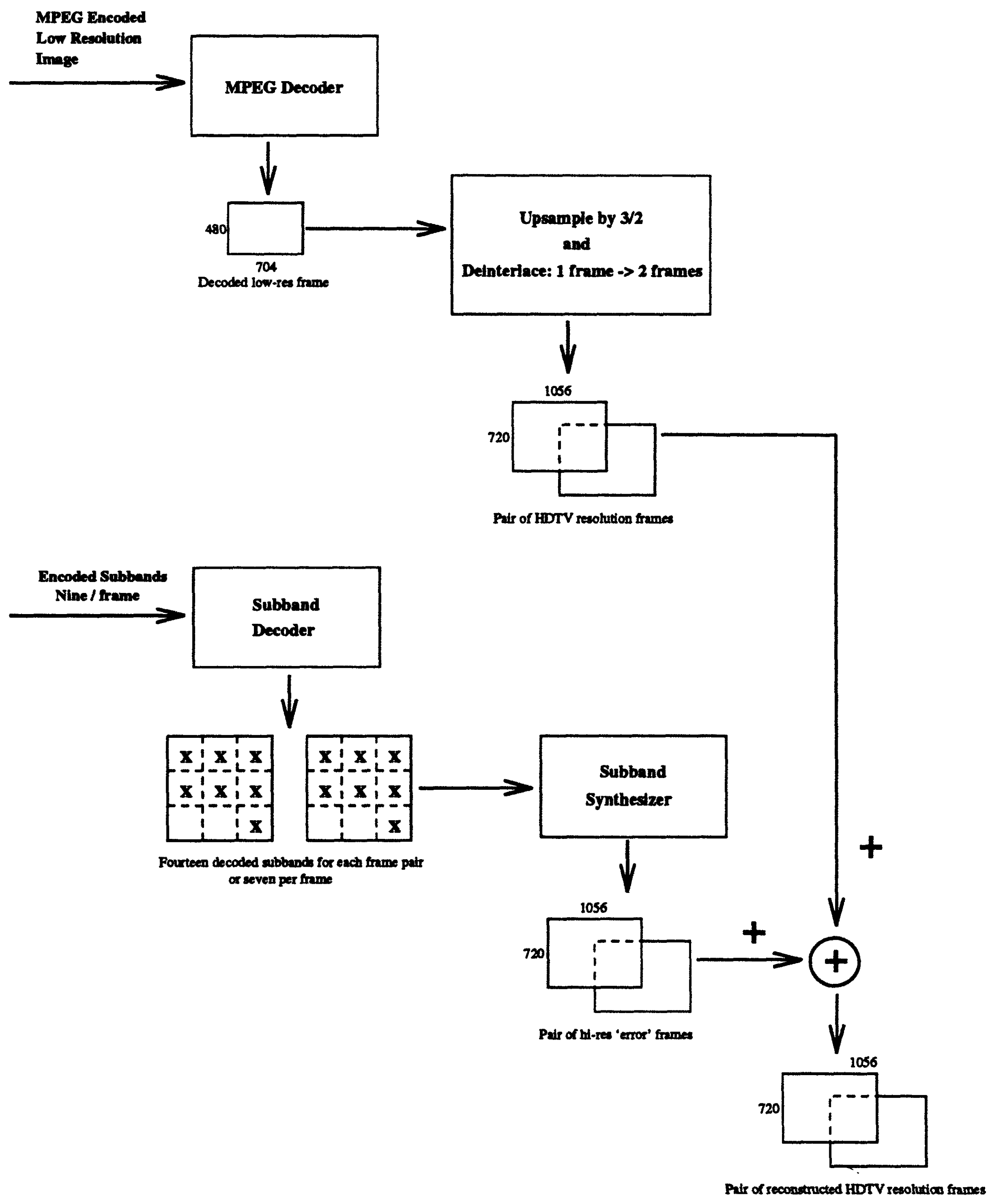


Figure 4-3: Block diagram of the 'Open Loop' decoder.

blonging to the class of predictive coders discussed in section 2.4, and illustrated in figure 2-9. Note that since the MPEG-2 coder is also a predictive coder, this scheme does not put an additional burden on the MPEG-2 encoder in terms of having to provide decoded frames at the low resolution.

Since it is the 'error' image which is coded, the base image need not be coded very well. Its bitrate needs to be just high enough to ensure that the low resolution image is of an acceptable quality. Temporal subband analysis is not required, though it may be efficient to use it nevertheless. Visually objectionable errors made by the MPEG-2 coder can be corrected for, provided enough bits are assigned to the appropriate subbands.

A block diagram of the closedloop encoder is provided in figure 4-4. The decoder is almost identical to the one in the openloop case, the only difference being in the number of subbands that is decoded, and in the fact that synthesis of temporal bands into frame pairs may not be required. Figure 4-5 is a block diagram of the closed loop decoder. Note that one essential functional block missing from all of the previous four block diagrams is the conversion unit from RGB format to the YBR 4:2:0 format (which was the format that was used within the coding blocks throughout this thesis). All the picture sizes given in these block diagrams are for the luminance component, and must be halved in each direction for the chrominance components in the 4:2:0 format.

The closedloop coder is more appropriate for situations in which the base layer channel is unreliable, the sequences to be encoded are demanding, or when a high quality picture at the high resolution is a requirement, and encoding complexity (delay) is not a problem.

4.2 The Functional Units

With the general structure of the coders in mind, this section will give the details of the functional unit implementations, and the design issues involved.

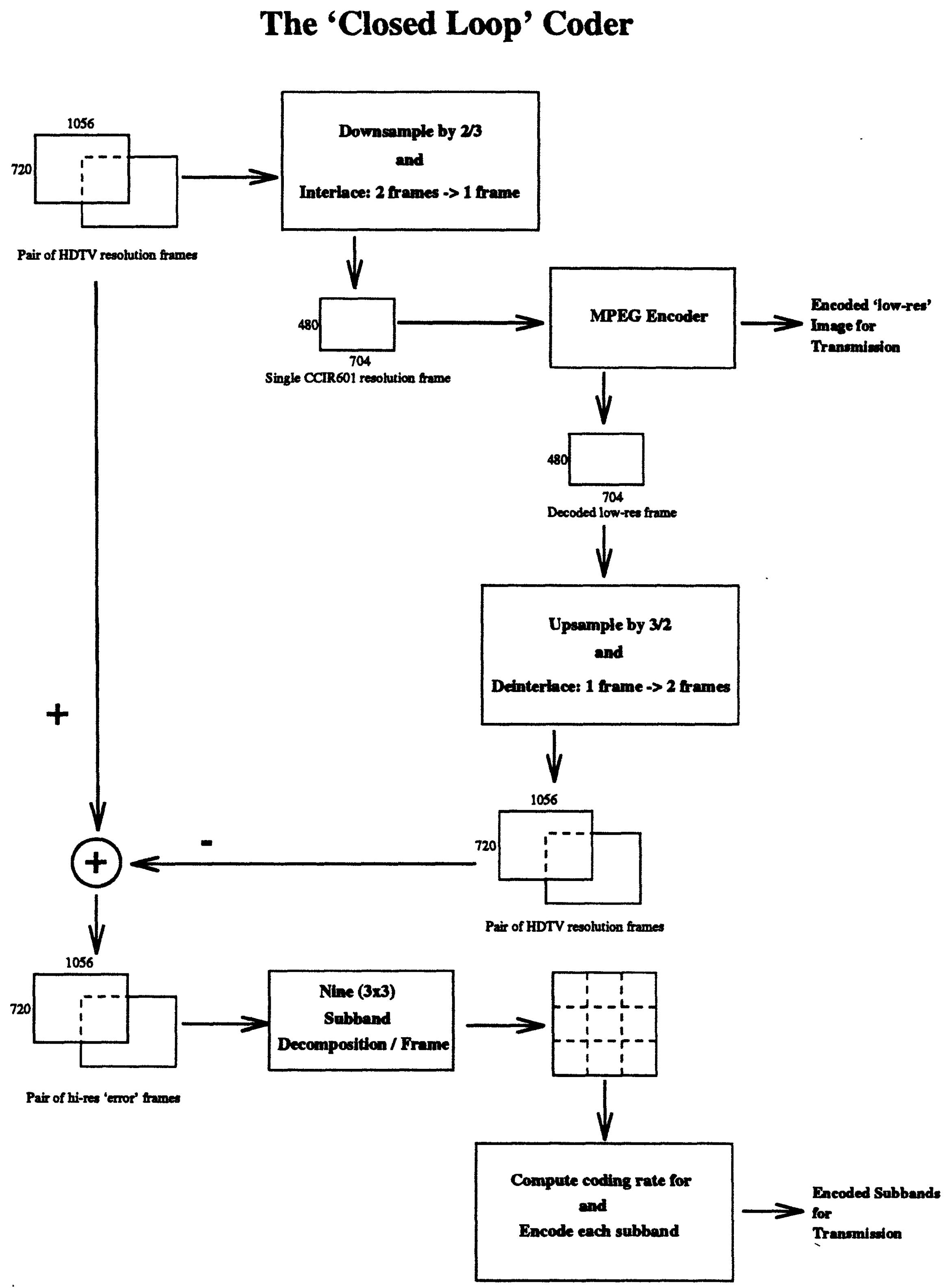


Figure 4-4: Block diagram of the 'Closed Loop' encoder.

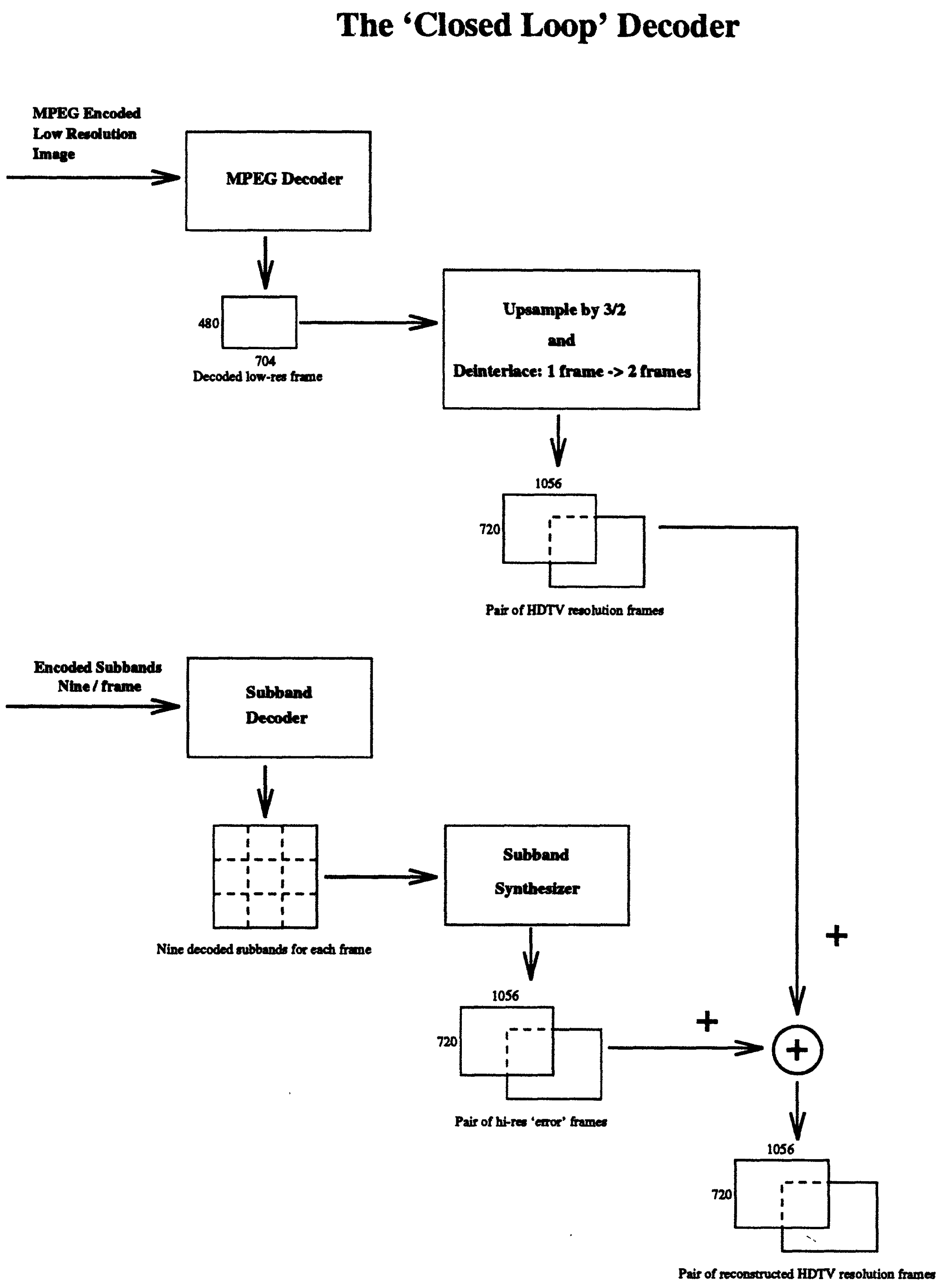

The 'Closed Loop' Decoder
MPEG Encoded Low Resolution Image
MPEG Decoder
480
704
Decoded low-res frame
Upsample by 3/2 and Deinterlace: 1 frame -> 2 frames
1056
720
Pair of HDTV resolution frames
Encoded Subbands Nine / frame
Subband Decoder
Nine decoded subbands for each frame
Subband Synthesizer
1056
720
Pair of hi-res 'error' frames
+
+
1056
720
Pair of reconstructed HDTV resolution frames

Figure 4-5: Block diagram of the 'Closed Loop' decoder.

4.2.1 The Downsampling Unit

If the base layer is in progressive format, downsampling is straightforward. It is performed in the 'standard' way [28], i.e. by upsampling by 2, applying a lowpass filter with a $\pi/3$ cutoff and then decimating by 3, separably along each dimension of the image. The only issue is designing an appropriate lowpass (anti-aliasing/interpolation) filter.

The most appropriate filter would be such that its frequency response complements the frequency response of the subband analysis filters, so that perfect reconstruction can be possible (assuming no coding, or lossless coding). Let us consider what happens as we downsample to the lower resolution and then upsample back. We first zero pad by a factor of 2 and then apply our $\pi/3$ cutoff filter (which cuts off at what would be considered $2\pi/3$ in the original spectrum). We then decimate by a factor of 3, which causes aliasing at the higher frequencies, since our filter does not have a 'brick wall' characteristic. This gives us the low resolution image. At this point, the original spectrum has been modified multiplicatively by the frequency response of the filter and, and the aliasing at the high frequencies. In order to upsample to the full resolution, we begin by zero padding our image by a factor of three. Next, we again apply our filter (with the appropriate gain). This has the effect of taking the entire spectrum of the low resolution image, normalizing it by a factor of three (so that a frequency of π now appears as $\pi/3$) and multiplying it with the frequency response of our filter. The next step is decimation by a factor of two. This 'stretches out' the spectrum (from the previous step) within each interval of 2π centered around a multiple of 2π. The net effect on the original spectrum is that frequencies higher than approximately $2\pi/3$ are cutoff. There is some aliasing energy present in the region around $2\pi/3$ in the frequency spectrum. At other frequencies the effect is precisely that of being multiplied by the response of our filter twice.

Ignoring for the moment the aliasing energy around $2\pi/3$, ideally the shape of the square of the anti-aliasing/interpolation filter's frequency response should be very similar to the shape of the sum of the frequency responses of the low and middle subband analysis filters for perfect reconstruction to be possible. Thus, if $q_l(t)$, $q_m(t)$

and $q_h(t)$ are the low, middle and high QMF filters (see section 2.3) respectively, and h(t) is the anti-aliasing/interpolation filter,

$$H^2(2j\omega) = Q_l(j\omega) + Q_m(j\omega) \tag{4.1}$$

where the equation has been written in terms of the frequency responses of the respective filters.

Given this equation, we can design the required filter by the following procedure. We sample the frequency response on the right hand side of equation 4.1 at a large number of points. We then take the point by point square root of this frequency response and add an equal number of zeros to it. We can consider this new set of values as expressing the sampled frequency response of our desired filter. Thus, if the original frequency response had a cutoff at $2\pi/3$, the response of this derived filter will have a cutoff at $\pi/3$, and the shape of the square root of the frequency response of the original filter (scaled down by two). We now need to take the inverse transform of this frequency response in order to obtain our filter. We can do this using MATLAB [23], which uses an equation error method [35] to find an approximation for the inverse Z transform, given a specified number of taps.

Unfortunately, the above algorithm does not yield good results for a reasonably low number of taps (upto 61). However, if we do not take the square root of the frequency response, we can still get a very good approximation of the filter we require. Not taking the square root also has another big advantage — the transition from the passband to cutoff is much sharper, and passband energy is lower, reducing the problem of aliasing energy error. On the other hand, the shape of frequency response is approximately the same in most of the passband and stopband regions, where the (normalized) gains are very close to unity and zero respectively.

Because of the aliasing energy which we have not accounted for, the filter specified by equation 4.1 may not be the best one for our purposes. This is particularly true when the QMF filters are such that the frequency response of their sum does not exhibit a sharp transition between the passband and cutoff regions. In this case, a large amount of aliasing energy will be included in the spectrum of the final upsampled image, which may appear as image artifacts. In the openloop structure, there is no way to correct for them, while in the closedloop structure they might require a disproportionately high number of bits to encode, reducing the bits allocated to coding other parts of the spectrum, and thus the decreasing the general quality of the image.

Another disadvantage of this method is the relatively large size of the filter required to form a reasonable approximation. Filtering is a computationally expensive operation, and large filter lengths are not desirable for our system. For these reasons it might be politic to abandon the perfect reconstruction ideal and use filters derived by using other filter design techniques, such as windowing [28]. In windowing, we begin with the impulse response of a 'brick wall' filter, which is infinite, and multiply it with a finite length window to obtain an approximation to that frequency response. The longer the window, the more accurate our approximation. This operation corresponds to convolving the perfect 'brick wall' frequency response with the frequency response of the window, which is usually close to the form of a sinc function. This has two effects: it makes the transition less sharp — the transition region is the length of the mainlobe width of the sinc like function — and it introduces ripple in the stopband and the passband, corresponding to the relative height of the sidelobes of the sinc like function. A number of windowing functions are known, which trade off these two factors, the simplest being a rectangular window.

After some experimentation (which was by no means exhaustive) , the low QMF filter shown in figure 4-13 was found to yield the best results in terms of overall SNR. The fact that it performs better than the filter derived from equation 4.1 without the square root can be explained by the fact that the response of the QMF is closer to the ideal response (as specified by 4.1).

When the base resolution sequence is interlaced, however, the situation is somewhat more complicated. We now need to convert from 60 frames per second to sixty fields per second, with successive fields covering different parity lines of the raster. Each different field at the base layer has a resolution of 240 lines, each consisting of

704 pels, and must be temporally derived from a different frame at the high resolution. This suggests that a high resolution frame must be vertically downsampled by a factor of 3 (instead of 2/3) to obtain a field. However, since the lines of successive fields need to be of different parity, and our downsampling factor is odd, we need to upsample by 2 and then do an offset downsampling by 6. This process is best described by means of an illustration, which is provided in figures 4-6 and 4-7. It is clear from these figures that since our downsampling factor is odd, the odd parity field needs to be derived from lines in between two fields, and thus simply shift downsampling by 3 will not do.

Once again, designing an appropriate anti-aliasing filter with a $\pi/6$ cutoff is an issue, and it should be done in such a way that perfect reconstruction within the subband structure is possible. If we ignore the effects of the shift required before downsampling for each odd field, we have a situation analogous to the one we have already discussed for the progressive downsampling case, the only difference being that we are now downsampling by a factor of 2/6 instead of 2/3. Thus, we would expect the spectrum of the final image to be cut off at about $\pi/3$, and this frequency response, with the exception of the aliasing energy around $\pi/3$, would be exactly like the original spectrum multiplied twice by the response of the anti-aliasing filter. Thus, for perfect reconstruction,

$$H^2(2j\omega) = Q_l(j\omega). \tag{4.2}$$

Again, we can use MATLAB to find the appropriate filter by sampling the square root of the frequency response of $q_l(t)$ and zero padding, as discussed earlier. MATLAB fails to give an accurate impulse response for small filter sizes, so we work with the frequency response without taking the square root.

For the three band QMFs that were used in this project, the low band filter had a gradual transition characteristic and had relatively low attenuation in the stopband. Thus the $\pi/6$ cutoff downsampling filter based on this filter's response allowed considerable aliasing. This showed up as stripe-like aliasing patterns in the

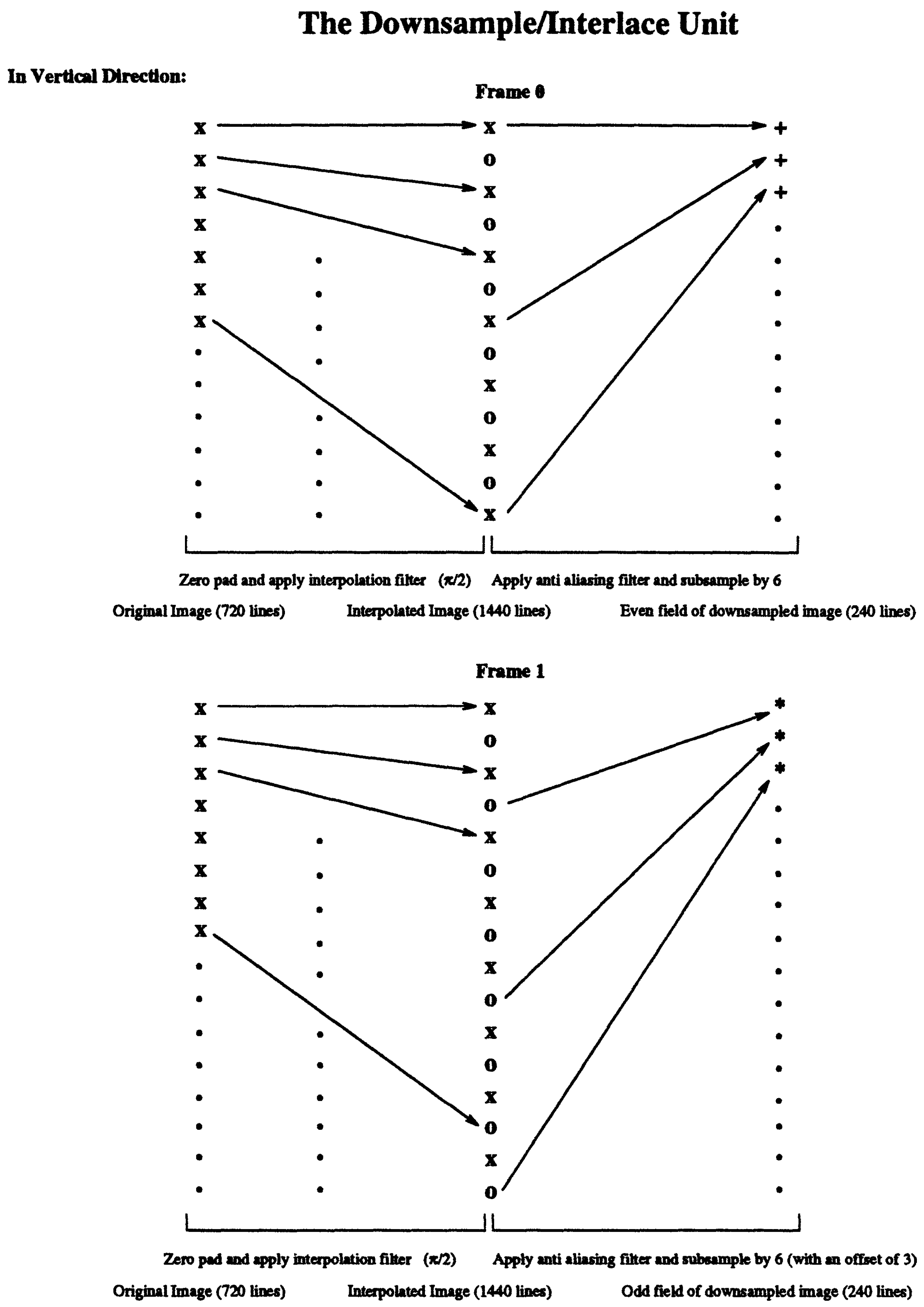

The Downsample/Interlace Unit
In Vertical Direction:
Frame 0
Zero pad and apply interpolation filter (π/2)
Apply anti aliasing filter and subsample by 6
Original Image (720 lines)
Interpolated Image (1440 lines)
Even field of downsampled image (240 lines)
Frame 1
Zero pad and apply interpolation filter (π/2)
Apply anti aliasing filter and subsample by 6 (with an offset of 3)
Original Image (720 lines)
Interpolated Image (1440 lines)
Odd field of downsampled image (240 lines)

Figure 4-6: Downsampling method used to derive the even and odd fields of the low resolution image from the progressive full resolution original. The $\pi/2$ interpolation filter is not needed since the $\pi/6$ anti aliasing filter applied immediately after it makes it redundant, but is included for completion.

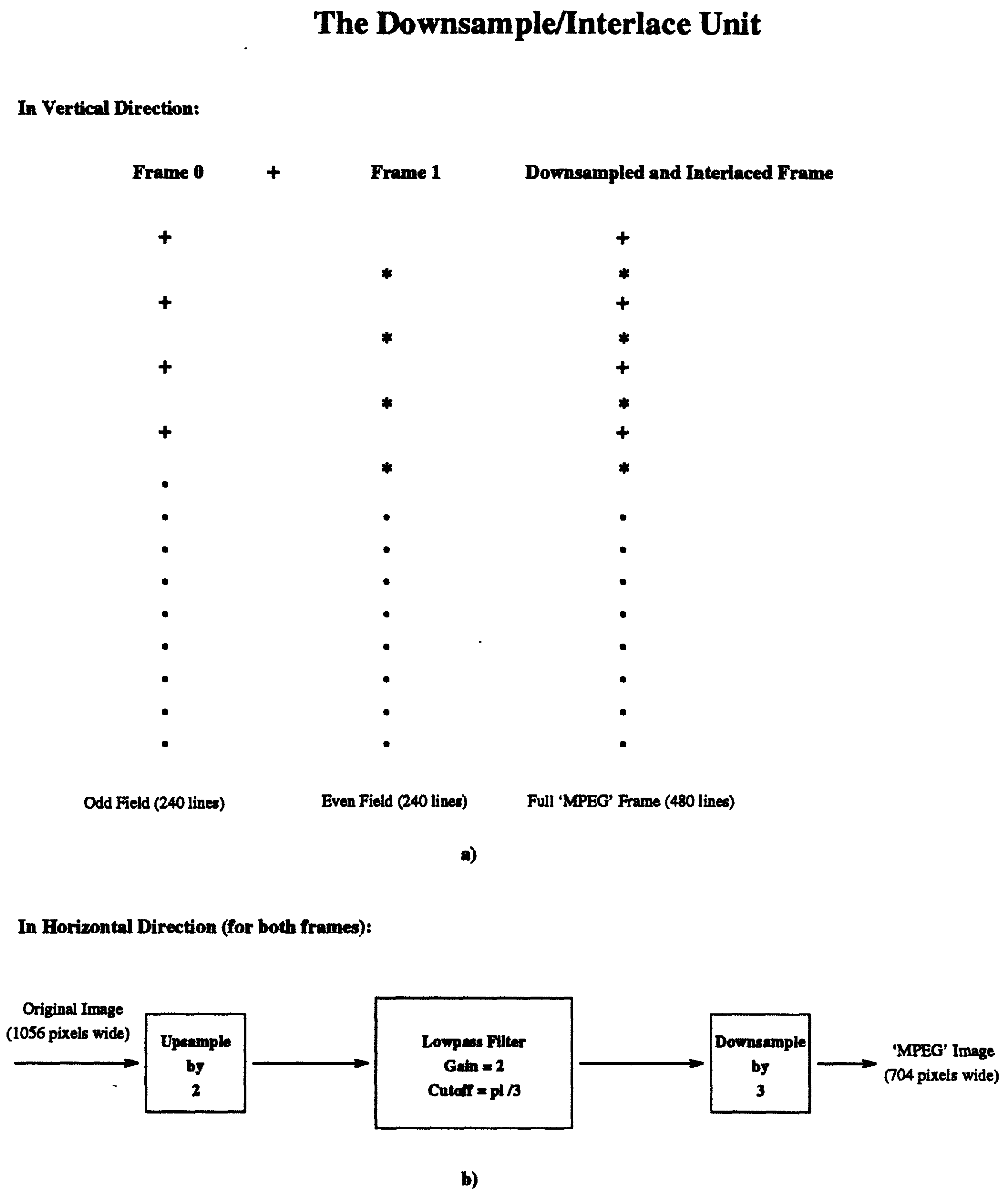

The Downsample/Interlace Unit
In Vertical Direction:
Frame 0
+
Frame 1
Downsampled and Interlaced Frame
Odd Field (240 lines)
Even Field (240 lines)
Full 'MPEG' Frame (480 lines)
a)
In Horizontal Direction (for both frames):
Original Image (1056 pixels wide)
Upsample by 2
Lowpass Filter Gain = 2 Cutoff = pi /3
Downsample by 3
'MPEG' Image (704 pixels wide)
b)

Figure 4-7: (a) The even and the odd fields derived as in figure 4-6 are merged together into a frame before being MPEG-2 encoded, (b) the horizontal rate conversion is done in the standard way.

image, particularly around edges. In the closedloop coder, bits were required to compensate for this aliasing energy, and consequently, the rest of the image suffered in terms of quality. In the openloop coder the aliasing could not be compensated for, and added to the noise in the image.

For this reason, a $\pi/6$ cutoff filter designed using a windowing method was used. The window length was 31 taps and the Hamming window [28] was used to obtain our filter. This window size has the advantage of having a relatively small mainlobe width, and very low sidelobe magnitudes. The magnitude of the frequency response of this filter is shown in figure 4-8 (a). The frequency response of the filter derived from the lowpass QMF filter is shown for comparison. We can see from this figure that the hamming filter has both a sharper transition and greater attenuation in the stopband region. It is worth noting, however, that the difference between the final images in which the two filters were used was minor — the images downsampled using the hamming filter were only about 0.3 dB better. Thus, better QMF designs might lead to coding gains in the system.

At this point it is useful to analyse what our operations do to the signal in the frequency domain as we downsample and interlace. As discussed previously, from the perspective of the full resolution image, all downsampling does (if we assume perfect lowpass filters), is limit the horizontal frequencies to $2\pi/3$ and the vertical frequencies to $\pi/3$. Of course, in reality, aliasing will extend these regions to beyond these points, and change the shape of the response within these regions. The regions of support for a full resolution image which has been downsampled and upsampled, using perfect filters is shown in figure 4-9.

Interlacing causes interesting things to happen in the vertical/temporal frequency plane. If we begin with a progressive image g[h, v, t] (with frequency response given by $G(\omega_h, \omega_v, \omega_t)$), we can obtain an interlaced image by multiplying this image by the two dimensional function,

$$\frac{1}{2} + \frac{1}{2} cos\pi[v + t]. \tag{4.3}$$

Thus, in order to get the frequency response of the interlaced image, we need to

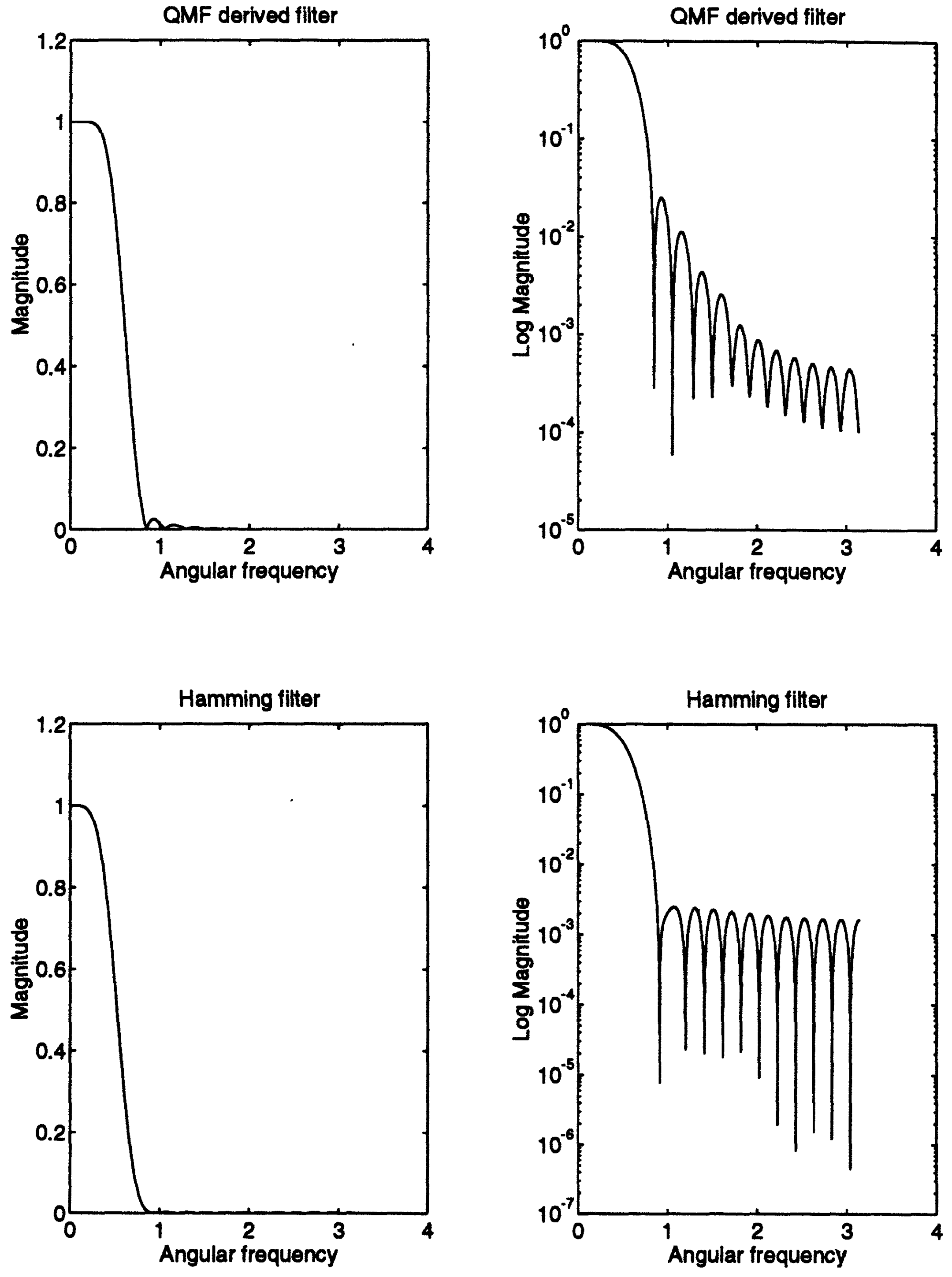


Figure 4-8: Frequency responses of the downsampling filters

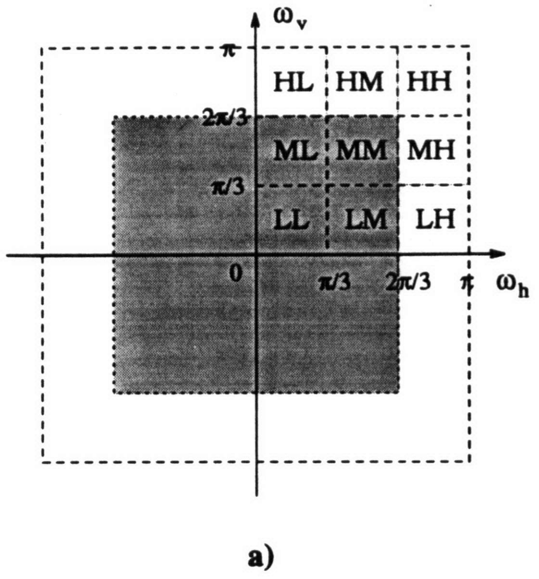


Figure 4-9: Regions of support for (a) progressively downsampled image, (b) interlaced and downsampled image, from the perspective of the upsampled full resolution image.

convolve $G(\omega_h, \omega_v, \omega_t)$ with the frequency response of 4.3, which consists of impulses at $(0, 0)$, $(+\pi, +\pi)$, and $(-\pi, -\pi)$. Furthermore, beginning with two fields with lines one pel apart, we can get an 'intermediate' progressive image simply by zero padding by a factor of two, and applying one pel shifts to alternate fields. Using the DTFT convention given in [21], this gives us,

$$I(\omega_h, \omega_v, \omega_t) = 2\pi^2 \left[H(\omega_h, 2\omega_v, \omega_t) + \frac{1}{2}H(\omega_h, 2\omega_v + \pi, \omega_t + \pi) + \frac{1}{2}H(\omega_h, 2\omega_v - \pi, \omega_t - \pi)\right], \tag{4.4}$$

where we have ignored the phase factor arising from the shift, and $H(\omega_h, \omega_v, \omega_t)$ is considered to be the spectrum of the sequence of fields we have obtained from downsampling. If we think about what this does in the frequency domain, the spectrum of the interlaced sequence consists of copies of the spectrum of the original image reflected around its mid point and added to itself. Thus the LL frequencies are aliased with the HHs, the LHs with the HLs and vice versa. Note that this operation is completely reversible if the spectrum of the original image is left intact.

In terms of computational complexity, if we perform a one dimensional convolution between two sequences of length M and N, then, if $N \leq M$, each point requires N

multiplications and $(N-1)$ additions to compute. Thus if an image of size X pixels by Y pixels is being separably filtered by filters of length N_h and N_v, we require $X \times Y \times N_h + X \times Y \times N_v$ multiplications and $X \times Y \times (N_h - 1) + X \times Y \times (N_v - 1)$ additions. In the interlaced case, since we must first zero pad by 2 and then decimate, the image size on which filtering is to be applied is 1440 by 2112. For the progressive case, filtering is applied to the full resolution image, which is 1056 by 720 pixels wide. Thus, for the two cases, using filters of length 31 and 15 for the interlaced case and 15 and 15 for the progressive case, we get the following numbers.

Downsampled Format	Multiplications	Additions
Interlaced	139,898,880	136,857,600
Progressive	22,809,600	21,288,960

4.2.2 MPEG-2 coding at the base layer

From figure 4-9, it is clear that compared to the progressive sequence, only half the number of subbands are available for MPEG-2 coding of the interlaced sequence at the base level. This makes intuitive sense, since the interlaced sequence contains half as much data as the progressive one, and the vertical resolution of a field is half that of a progressive frame.

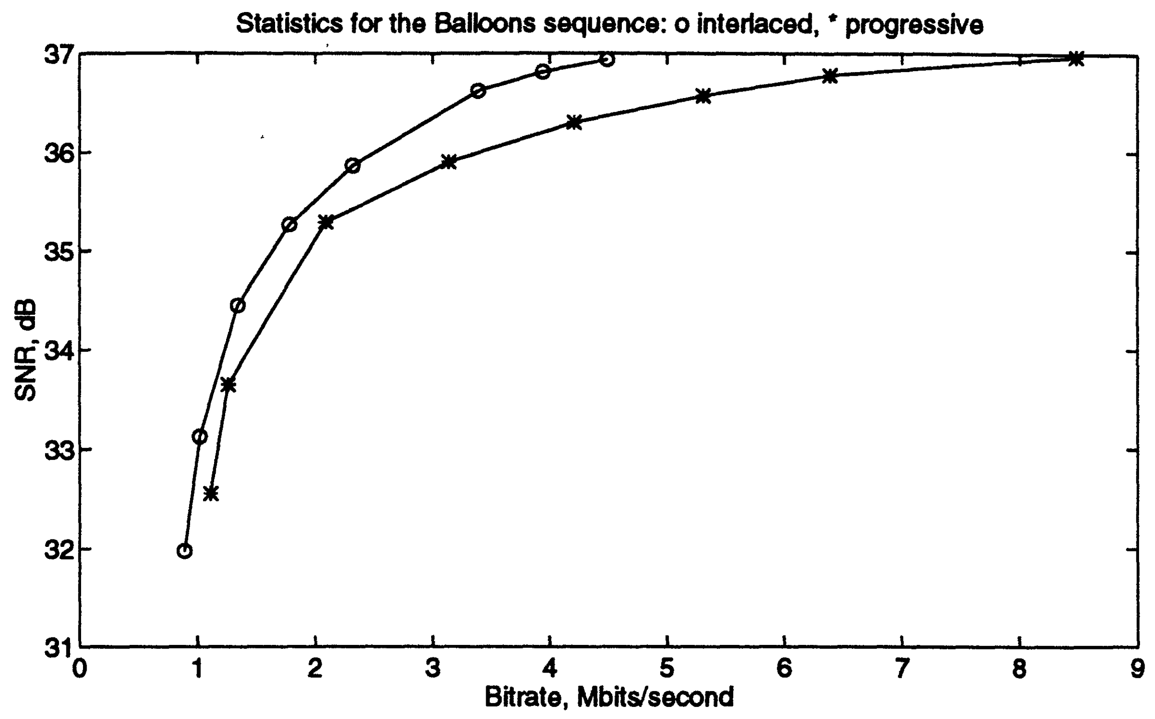
The 60 field per second interlaced sequence is converted to a 30 frame per second sequence by interleaving together successive pairs of fields into frames. These are treated as frame type pictures by MPEG-2, which performs field/frame adaptive motion compensation on them. Field/frame adaptive mode is appropriate if we consider the fact that the each field is derived from a frame of the original sequence, and thus performing motion compensation amongst neighboring fields is in a way equivalent to performing motion compensation on neighboring frames of the original sequence. However, if a region of the interlaced frame does not contain a lot of vertical detail (which may have originated as detail between the two fields), frame mode provides more efficient motion compensation, since it generates only one motion vector per macroblock (as opposed to the two generated in field mode).

Merging fields into frames can be though of as zero padding each field vertically, convolving with the anticausal filter $f[v,t] = \delta[v,t] + \delta[v,t+1]$, and then decimating by two in the temporal direction. In the frequency domain this leads to (1) a rescaling of the vertical frequency axis, such that ω_v is replaced by $\omega_v/2$, (2) a multiplication by a cosine frequency response along the temporal axis, and (3) aliasing along the temporal axis as a result of downsampling. Thus if we begin with $I(\omega_h, \omega_v, \omega_t)$, we get a great deal of temporal aliasing in the frequency domain. The net result is that the entire spectrum of the image contains aliasing energy from other parts of the spectrum. The only region which is relatively free of it is the LL area, where, in typical images, large amounts of aliasing do not occur because of the band limited nature of the input.

Throughout this project, the parameters M and N were chosen to be 3 and 15 respectively. Thus there were two B frames between any two P or I frames, and one I frame after 15 frames of the other types. These values were chosen since they provide a good compromise between the compression rate (which decreases as the number of I frames increases, and increases with the number of B frames) and computational complexity (which increases with the number of B and P frames). In addition, the values chosen for these parameters facilitate features such as random searches and reverse playback. Another parameter that needed adjustment was the search window size for block matching. The two images used for the initial tests of this coder were scanned in at 24 frames per second, and thus they contained a disproportionately large amount of motion between frames. The search window per frame was set at 16 by 16 pixels. This meant that when block matching over N frames a search window of $16N$ by $16N$ would be used. For the above values of M and N this resulted in a maximum search window size of 48 by 48, which occurred whenever a P frame was being predicted.

Above bitrates of about 6 Mbits/sec (calculated with respect to 60 frames/s), pictures of good viewing quality resulted at the lower resolution. Figure 4-10 plots the signal to noise ratios for the two test sequences against bitrate [1]. Notice that the

[1]The figures shown in figure 4-10 were computed using an older (Test Model 2) version of the



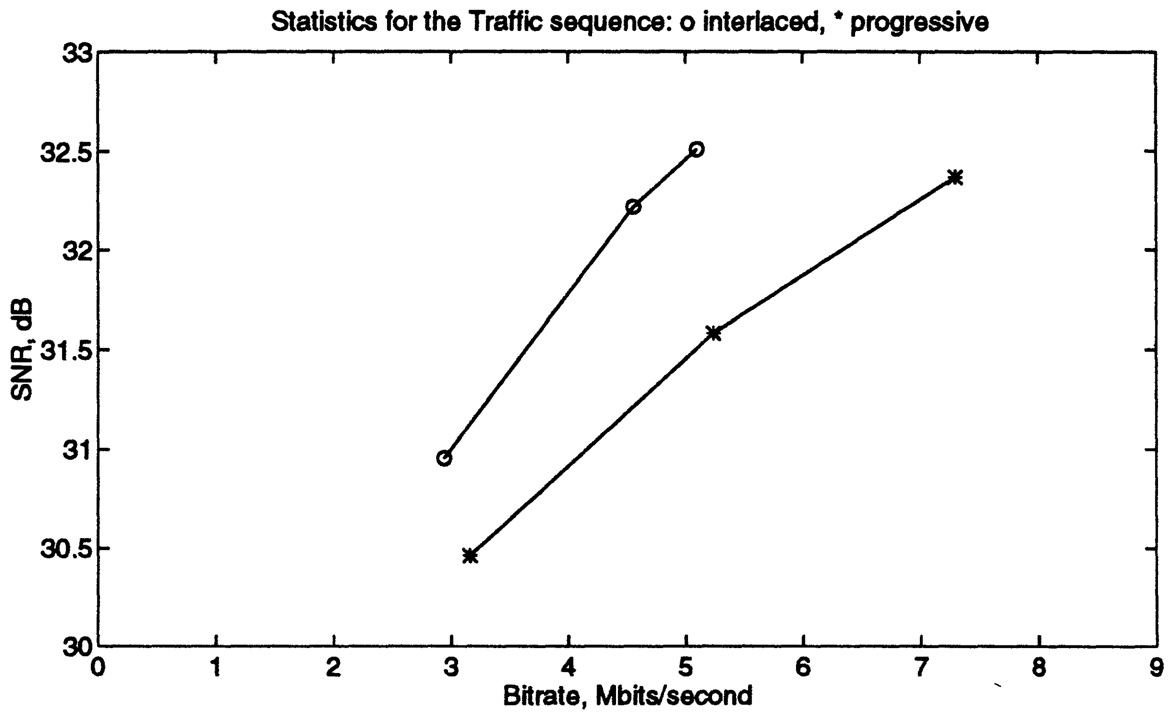


Figure 4-10: Bitrate vs. SNR for the 'Balloons' and 'Traffic' sequences.

'Traffic' sequence is a much more demanding one and results in lower SNRs at the same bitrate. This is not surprising, since 'Traffic' contains much more detail, both spatial and temporal, than the 'Balloons' sequence. These sequences are displayed in figures 5-1 and 5-2. Another observation worth making about figure 4-10 is that although the progressive sequences contain twice as much data as the interlaced ones, the bitrate required to achieve the same SNR is not twice as high. This suggests that MPEG-2 is not as efficient at coding interlaced sequences as it is in coding progressive ones. This is not surprising, given our previous discussion, where it was shown that a frame type interlaced picture contains a large amount of aliased energy, which reduces the correlation amongst DCT coefficients. Thus, the DCT and entropy coding blocks of the MPEG-2 coder cannot do as good a job of compression as they can with the more correlated energy in a progressive image.

One problem with the MPEG-2 coding which is not very noticeable at the low resolution, but is very apparent at the full resolution is the presence of blocking artifacts . Blocking artifacts are a common problem with block based DCTs. They result from coarse quantization of DCT coefficients, leading to the appearance of artificial boundaries between blocks. Since these boundaries are unnatural they are visually both noticeable and unpleasant. Not surprisingly, they are a particular problem at low bitrates. With the MPEG-2 coder used for this project, blocking artifacts were visible whenever the rate control was overloaded, even when the overall bitrate was high. This occurred in particular in regions which contained a great deal of both spatial detail and motion. A good example is the region indicated around the man's mouth in the 'Balloons' sequence, shown in figure 5-3. As in that region, blocking artifacts seem particularly unpleasant when a background with details to mask them is not available. In addition to looking unpleasant, these artifacts reduce the coding gain in the closedloop coder, which has to spend bits compensating for them. In the openloop coder, they are a serious problem if they are very apparent, since the openloop structure cannot compensate for them.

MPEG-2 coder, and are not as good as those for the coder used in our final experiments. In general there is about a 1-2 dB difference. However, the trends that are shown are preserved across models.

Blocking artifacts can be reduced by a number of methods, the simplest of which is by filtering the pixels around the block boundaries (in the decoded image) with a very smooth lowpass filter. This filter should be smooth enough that it does not adversely affect the non-spurious details in the image. This method, originally proposed by Reeve et. al. [13], works well in low bitrate applications, where picture quality is not required to be very good. Within our system, though, it was not found to be very effective. By using the filter proposed in [13], we were able to reduce the blocking, but it was still visible. Within the closedloop structure, this operation yielded a very small gain in the SNR of the final image. It was thus decided to abandon this method, and use higher bitrates at the base resolution instead. In the openloop coder, however, this method might still be useful as a way of enhancing the image at the decoder if blocking artifacts are still a problem. A much more effective solution would be to take account of this problem when choosing the MPEG-2 rate control mechanism.

Figures for the number of arithmetic operations required for MPEG-2 coding are given at the end of chapter 2.

4.2.3 The Upsampling Unit

Upsampling/deinterlacing is performed in the obvious ways. If the base image is progressive, then the standard method, shown in figure 4-11 is used. If the base layer is interlaced, then the two fields are first separated from the MPEG-2 decoded image, and then each field is upsampled to a full resolution frame, using shift upsampling, with a shift of three on every alternate field. This process is illustrated in figure 4-12. The fields are horizontally upsampled according to figure 4-12.

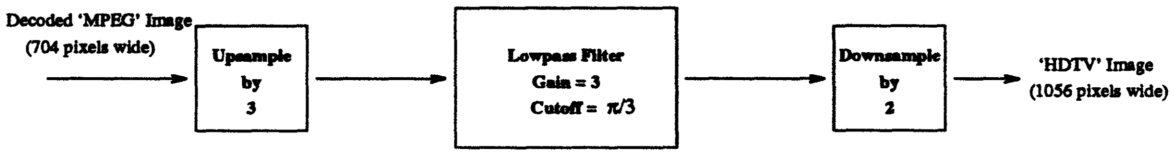


Figure 4-11: Upsampling unit for progressive sequences and horizontal lines of interlaced sequences.

The computational requirements for upsampling are identical to those of down-

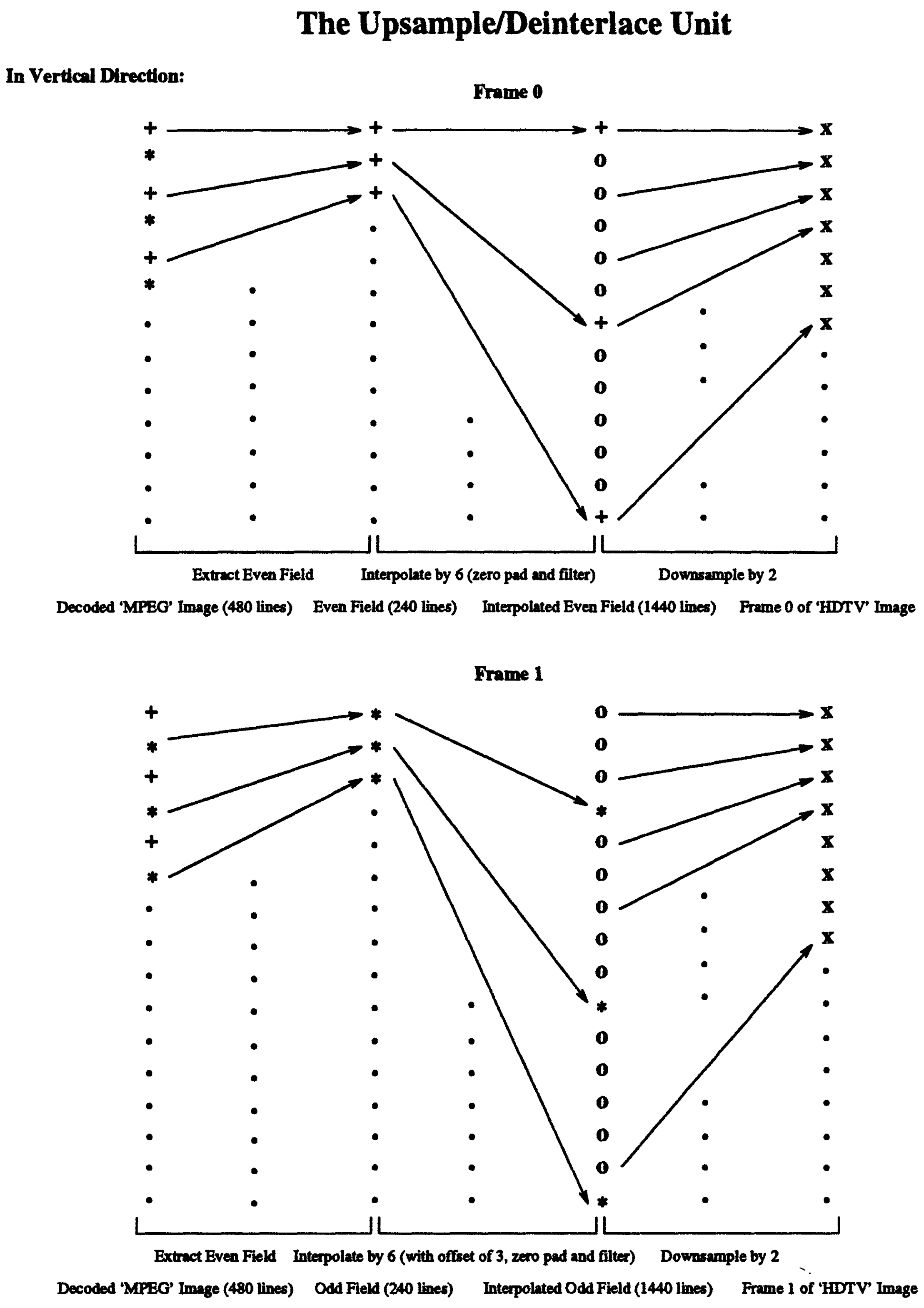

The Upsample/Deinterlace Unit
In Vertical Direction:
Frame 0
Extract Even Field
Interpolate by 6 (zero pad and filter)
Downsample by 2
Decoded 'MPEG' Image (480 lines)
Even Field (240 lines)
Interpolated Even Field (1440 lines)
Frame 0 of 'HDTV' Image
Frame 1
Extract Even Field
Interpolate by 6 (with offset of 3, zero pad and filter)
Downsample by 2
Decoded 'MPEG' Image (480 lines)
Odd Field (240 lines)
Interpolated Odd Field (1440 lines)
Frame 1 of 'HDTV' Image

Figure 4-12: Upsampling unit for the vertical lines of interlaced sequences.

sampling.

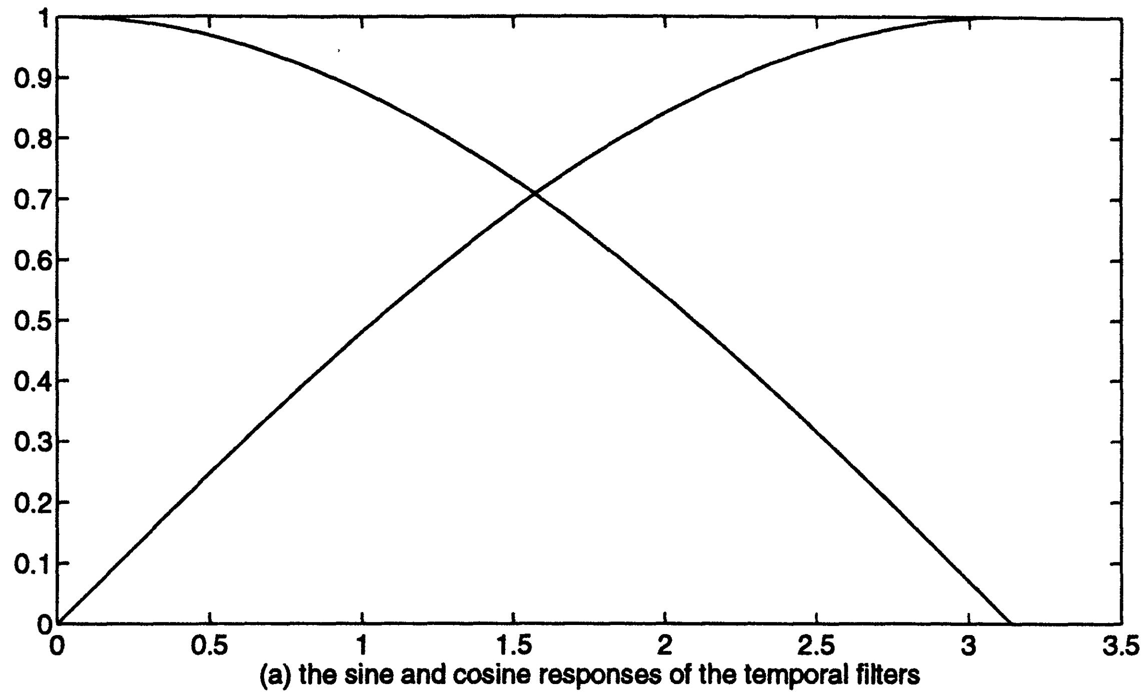
Downsampled Format	Multiplications	Additions
Interlaced	139,898,880	136,857,600
Progressive	22,809,600	21,288,960

4.2.4 The Subband Coder

Because of the 2/3 ratio between the sizes of the the base layer and full resolution images, a nine subband structure in the spatial frequency domain is a natural choice.

The first issue to be resolved within this structure is whether or not a temporal subband split will be advantageous. Theoretically, it should yield a coding gain simply because there is less energy in the high temporal frequencies than there is in the lows, and a temporal split allows us to apportion our bits appropriately. In practice, though, we need to ensure that the temporal split is a clean one. i.e. that a lot of energy does not leak across temporal subbands, and this adds significantly to the computational burden and the encoding and decoding delay. Within the closedloop structure, we tried a simple temporal subband decomposition by averaging over the sum and difference of frame pairs. This did not yield results as good as those from the simple nine band decomposition. We can understand why this happens if we consider the frequency responses of the two tap temporal filters we have applied. The magnitude of the frequency response of the averaging filter is a cosine, while that of the differencing filter is a sine. These two responses are shown in figure 4-13 (a). A lot of the middle portion of the frequency spectrum is shared by these two functions. Thus there is a large amount of duplication of energy between between subbands, which ends up being coded twice. It was experimentally found that the difference in SNRs between doing and not doing a temporal split was very small (a fraction of a dB) though. Thus using a more sophisticated temporal filer might yield a significant coding gain.

More sophisticated temporal filters, such as the 3 tap and 15 tap filters used in [18] would no doubt yield better results, but it is debatable whether this increase



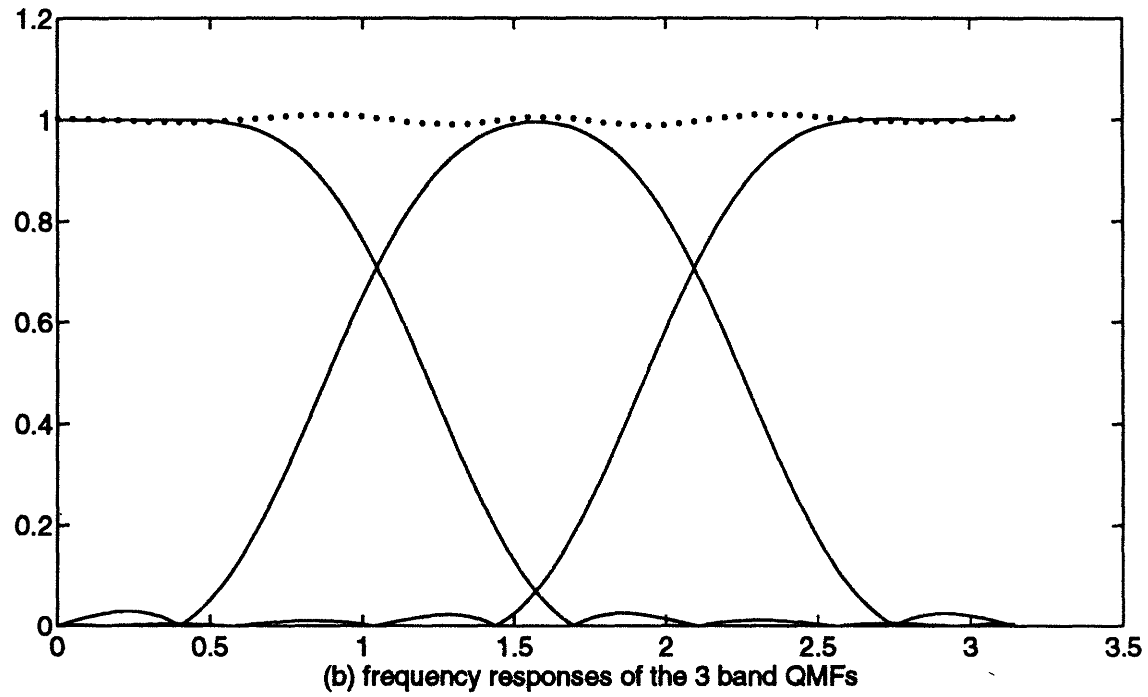


Figure 4-13: (a) Frequency responses for the averaging and difference filters, (b) frequency responses for the 3 band QMF filters

in performance merits the extra computational burden. We can get a significant improvement by using a temporal split if the image being coded does not contain a great deal of motion, since in that case the differene image will not contain a great deal of energy. In that case, even the two tap filter used here would likely yield a coding gain. Within the closedloop structure, however, MPEG-2 handles regions of low motion very effectively, and as a result temporal frequencies are more uniformly distributed for the enhancement image. This is not true in the openloop case, however, since the subbands being coded come directly from the image and MPEG-2 has not had a chance to remove the low temporal frequencies.

The next problem is coming up with a set of appropriate 3 band QMFs. Since, for optimal performance, the QMF shape must be replicated by the downsampling filters, we have to ensure that the filters we employ have relatively sharp transitions and large stopband attenuation. For purposes of this project, the 15 tap 3 band QMFs designed by Simocelli [34] were used. These were designed using the procedure outlined in section 2.3.1. Although these filters were not optimized, they produced acceptable results. The frequency responses of these filters are shown in figure 4-13 (b).

In terms of computational requirements, the full resolution image must be filtered nine times to obtain the nine subbands in the closedloop case, and and seven times *per frame* in order to get the seven spatial subbands for the openloop coder. If in addition, the simple temporal analysis described above is to be used, we need an extra 720×1056 operations *per frame*. Using the results for convolution, and assuming QMF filter lengths of 15, we get the following figures.

Coder type	Multiplications	Additions
Closedloop	205,286,400	191,600,640
Openloop-interlace	159,667,200	149,022,720
Openloop-progressive	114,048,000	106,444,800

Once the subbands have been separated, they must be coded. This is done by quantization, followed by run length coding of zeros, followed by entropy coding.

Figure 4-14 is a block diagram of the subband encoding and decoding process.

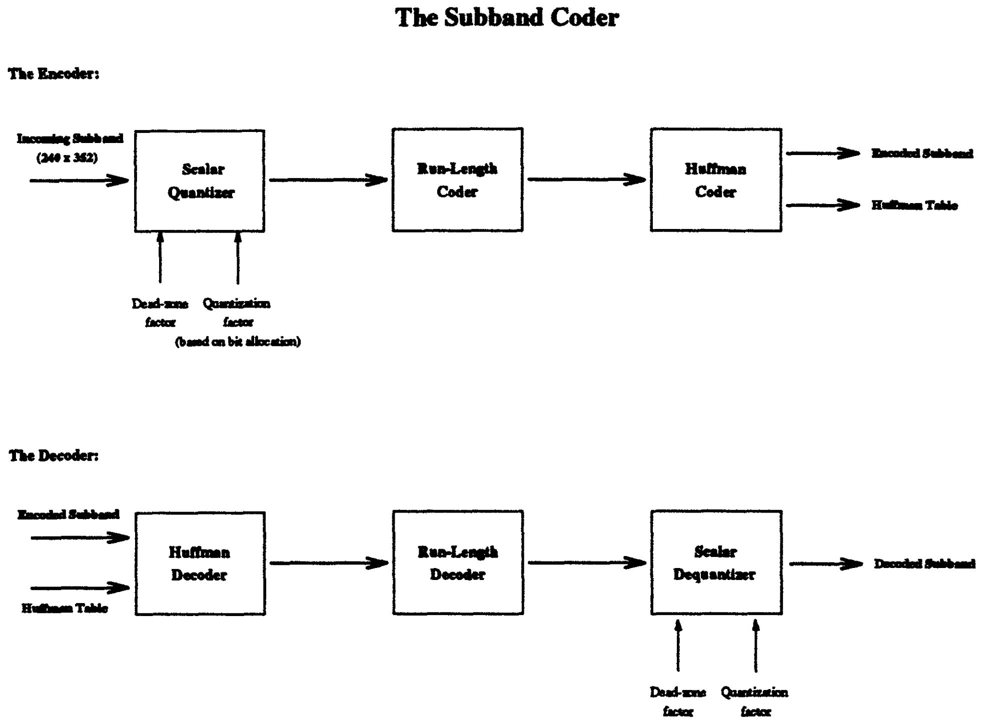


Figure 4-14: Block diagram of the subband coder

The quantizer is a uniform one and takes three inputs, the quantization factor, the dead zone factor and a mask file. The mask file contains an array of masking factors for each 8 by 8 block in the subband. Each masking factor is the the normalized sum of absolute values of all the pixels in that block over all the luminance subbands that are to be coded. The masking factor is used as a measure of image activity in a particular region of the image. The premise behind this procedure is that the more active a region of a normal image, the more detail it contains, which can mask out errors. Thus we can get away with quantizing that region more coarsely. This argument holds for the openloop case, where we are coding subbands directly from the image. Regions of high activity can indeed mask noise. However, for the closedloop case, the opposite is true, at least for the low subbands which are coded by MPEG-2, where the error between the real image and MPEG-2's prediction of it is being coded. A more active region simply means that MPEG-2 did not do a sufficiently good job of coding that region — thus in order to effectively remove MPEG-2 artifacts, active regions should be coded more finely. Thus, in MPEG-2 coded subbands, the local quantization factor for a particular region is derived by dividing the subband quantization factor by the masking factor, while in non MPEG-2 coded subbands, it is obtained by multiplying the subband quantization factor by the masking factor. Note that in terms of SNR, the argument that a more energetic region should be quantized more coarsely is counterintuitive. If increasing the SNR is the the only requirement, then most of the available bits should go to the regions with the greatest amount of energy, since those are the regions which make the largest contributions to the MSE and SNR. However, SNR is not always a good measure of human perception of image quality. The masking effect at high frequencies is a very well documented and much exploited phenomenon in the design of imaging systems.

The block size for computing masking factors is chosen to correspond to the MPEG-2 macroblock size at the the low resolution. This is because a potentially different value of *mquant* is used for each macroblock, which affects local image characteristics. Thus, a macroblock size of 16 by 16 at a resolution of 480 by 704 gives us a masking block size of 8 by 8 at a subband size of 240 by 352.

The next issue is whether or not to use a dead zone when coding a particular subband. A dead zone maps relatively large values around zero to zero and thus provides the potential for more efficient run length coding. If a large number of values in a subband is clustered around zero, a dead zone increases distortion significantly. On the other hand, it frees up bits to use in coding perceptually more important values more accurately. Thus a dead zone will yield a better looking image if the values around zero in a subband are perceptually not important. This is the case for the high subbands in both the openloop and closedloop coders. Zeroing a small but non zero coefficient contributes to background noise, but the noise is not very noticeable if more significant details are accurately rendered. In the lower subbands, especially the ones neighboring the MPEG-2 coded bands, this is no longer true. These bands contain a significant amount of aliasing energy, which may show up as small pixel values in the error image which are nevertheless important since aliasing is very noticeable. The lower bands are also visually more important compared to the higher ones, and a mistake in the lower bands carries a higher perceptual penalty.

Thus, in this project the higher subbands were quantized with a deadzone, while the MPEG-2 coded subbands and the subbands adjacent to it were not.

The number of arithmetic operations required for quantization is relatively small. Computation of the mask file requires $240 \times 352 \times (9, 7, 5)$ operations per frame, depending on the number of subbands employed. A DPCM coded version of the mask file (with a step size of 0.2) is used at both ends. Quantization involves one multiply/add per masking block to determine the quantization factor. A pixel is quantized and dequantized according to the following expressions.

$$\begin{aligned} \text{Qpix} = &\quad [\text{pix} \pm \text{quant}/2] \;//\; \text{quant}, && \text{if abs(pix)} \geq \text{deadzone} \\ &= 0 && \text{if abs(pix)} < \text{deadzone} \\ \text{DQpix} = &\quad \text{quant} \times \text{Qpix} \end{aligned}$$

Thus the number of arithmetic operations required per frame for quantization and dequantization is given as follows.

Coder type	Mults/Divs	Adds/Subs
Closedloop	1,520,640	1,521,960
Openloop-interlace	1,182,720	1,184,040
Openloop-progressive	844,800	846,120

After quantization, the pixel values are subjected to lossless coding to further increase compression. The fact that high subbands do not contain a lot of energy and the presence of a quantizer with a dead zone in the preceding stage make long zero runs fairly common, which can be efficiently compressed using run length coding of zeros. In the next stage, a Huffman coder further reduces redundancy. The Huffman coder used in this project built a table based on statistics for the entire subband before coding. Though this is theoretically the optimal way to do it (in the absence of a known probability distribution for the subband), it introduces coding delay. A quicker method might be to use dynamic huffman coding, which was discussed in section 2.5.1, This type of huffman coder dynamically builds a huffman table. This would, of course, be less efficient in terms of providing compression.

Instead to the runlength/huffman coding pair, an arithmetic coder may be used for lossless coding of the quantized data. For this project, the Q-coder described in section 2.5, and specified in [30] is used as an alternative. This Q coder uses a state machine driven probability estimation method. Numbers are decomposed into binary decisions such as zero/nonzero, positive/negative, and magnitudes and actual values are arithmetic coded using the different estimated conditional probabilities, which are dynamically computed using several state machines. Since these conditional probabilities can be estimated in an identical manner by both the decoder and the encoder, no table needs to be built or transmitted prior to coding a particular subband. The arithmetic coder did not provide as efficient compression as the runlength/huffman pair. This is not surprising given the fact that the arithmetic coder does not begin with a priori knowledge of the subband statistics, as the huffman coder does. In practical implementations, however, either such an arithmetic coder or a runlength coder followed dynamic huffman coding would be preferred for its lower coding delay.

Bit allocation

Since there is no really good metric for perceived image quality, no 'optimal' scheme for bit allocation amongst subbands exists. Even if a 'perfect' metric existed, we would have to know the probability distribution of the data we are coding in order to come up with an optimal bit allocation. For this reason, common bit allocation schemes are usually somewhat arbitrary, and are based on a combination of intuitive notions of type of information subband statistics (such as the variance) convey and experimentation with different types of images.

We may use reasonable approximations as a guide, though. Thus if the error criterion to be minimized is MSE, the basis functions of the subbands are orthogonal, each subband is independently quantized using a Lloyd Max quantizer, and d_k^2, the error distortion for the kth subband can be related to the bitrate R_k and the subband variance σ_k^2 by,

$$d_k^2 = \epsilon_*^2 2^{-2R_k} \sigma_k^2, \tag{4.5}$$

then the optimal bit allocation, subject to the constraint of constant average bitrate R, can be shown by the method of Lagrange multipliers to be given by an expression identical to equation 2.19, with DCT coefficients replaced by subbands. Including, the perceptual weighting criteria v_k, the expression is:

$$b_k = R + \frac{1}{2}\log_2 \frac{v_k\sigma_k^2}{[\prod_{j=0}^{N-1} v_k\sigma_j^2]^{1/N}}. \tag{4.6}$$

Here b_k and σ_k^2 are, respectively, the bitrate allocated to and the variance of subband k, B is the overall bit rate, and N is the number of subbands being quantized. That the expressions are identical for the DCT and subband coders should not be surprising, given the the discussion on page 30 about the equivalence of these two types of coders.

Equation 4.5, where ϵ_*^2 is a constant depending on the probability distribution of the image, has been experimentally observed to hold for most images [16]. The basis functions for the subbands are orthogonal, and the MSE is a reasonable (though sometimes untrustworthy) criterion of perceptual quality. In this setup we are not performing optimized scalar quantization, though given the fact that a uniform quantizer is the optimal entropy constrained quantizer, this might not be a bad assumption. Thus, in the absence of anything better, equation 4.6 should give us at least a reasonable guess for the bit allocation. A careful look at equation 4.6 reveals that it is possible (when the variance of a subband is low, and the bitrate to be apportioned amongst the bands is low), for the formula to give us a negative bitrate, which is impossible in practice. Clearly, in practice equation 4.6 can only serve as a guide, and should not be considered to be of fundamental importance. In the experiments for this project, if a subband was assigned negative bits according to equation 4.6, then its bit assignment was set to zero and the equation was applied to the remaining subbands, until a set of subbands with positive bit allocations resulted.

A bit allocation scheme that assigned bits to each subband in direct proportion to its variance times a weighting factor was found to yield slightly better results than using the above formula. The weighting factors work only for the images we dealt with though, and there is no guarantee that the particular values we used are generally applicable. Although equation 4.6, does not provide the best bit allocation for every image, in combination with some perceptually biased weighting factors, it does give good results. We may use our knowledge of the perceptual significance of subbands to choose the weighting factors v_i in equation 4.6. It is a well known fact that in general,

- the higher the frequency, be it horizontal, vertical, or temporal, the lower its perceptual significance,
- vertical and horizontal frequencies have approximately the same importance, however diagonal frequencies are less important.
- chrominance subbands are less important than luminance bands.

Thus, a reasonable but nevertheless arbitrary weighting scheme is the one described in figure 4-15.

3	2	2
3	3	3
3	3	3

a) Temporal lows

2	1	1
2	2	2
2	2	2

b) Temporal highs

Figure 4-15: Perceptual significance weighting for the eighteen spatiotemporal bands

Rate control

Rate control must be an essential part of any coder that uses variable length coding, since the rate of the output bitstream in such a coder is not possible to determine a priori. We must constrain the bitrate within certain parameters so that,

- if the bitrate is too high, channel capacity or decoder buffering is not exceeded
- if the bitrate is too low, channel capacity is not wasted, or image quality needlessly compromised

As we have seen with the MPEG-2 rate control algorithm, effective control should be performed adaptively at a relatively low level, such as at the macroblock or slice layers in MPEG-2. The MPEG-2 coder defines a buffer fullness variable, which it uses to control the bits assigned to each macroblock by means of the control variable *mquant*. Though real implementations of our coders should do something similar by providing rate control at the subband or block level, nothing as fancy was implemented as part of this project.

Rate control is achieved in our system by varying the quantization parameter for each subband. We initialize our quantization parameters by iterating through a number of values as follows.

1. For each subband in the first frame, choose an initial quantization parameter (quant) of either 2 or the quant resulting from the previous subband. Carry through the entire coding process (quantization, entropy coding).
2. If the resulting bitrate from the previous step, which used a quant of Q is too high compared to the target bitrate for the band, attempt a quant of $2Q$. If too low, choose a quant of $Q/2$ and carry through the coding process. If the resulting bitrate is within *tolerance* of the target, stop.
3. If the two previous quants give bitrates which are both above or both below the target, go to step 2. Otherwise try a quant which is the average of the previous two quants. If the resulting bitrate is within *tolerance* of the target, stop. Otherwise, of the last three quants, keep the two quants which give one bitrate closest to the target above the target and the one bitrate closest to the target below the target. Repeat step 3.

On average, this algorithm converged to within 0.05 bits/pixel of the target bitrate in about 5 to 7 iterations. Thus in effect we are performing the entire coding about six times in this step. Since the first frame at the base layer is always an I frame which takes a relatively small amount of computation to code, the extra computational capability will presumably be available at this stage.

The quantization factor for the same subband in the next frame is computed by taking advantage of the approximate relationship given in equation 4.5 for scalar quantized sources and the fact that if either the bitrate is high or if the probability distribution of subband coefficient values is flat outside the dead zone, then the quantization factor is proportional to the square root of the distortion [16]. This leads us to:

$$Q \propto \epsilon_* 2^{-R_k} \sigma_k, \tag{4.7}$$

where Q is the quantization parameter of a subband and the other quantities are as used in equation 4.5. Thus, whenever the above approximations are valid, then, assuming that the statistics of a subband do not change significantly from frame to frame (or frame pair to frame pair), and thus equation 4.5 remains valid, we may calculate the a new quantization parameter from the previous one by,

$$Q^* = Q^\dagger \frac{\sigma_k^\dagger}{\sigma_k} 2^{-(R_t - R^\dagger)}, \tag{4.8}$$

where Q^* is the predicted quantization factor, and $Q^\dagger$, $\sigma_k^\dagger$, and $R^\dagger$ are the quantization factor, standard deviation, and achieved bitrate for the subband in the last frame.

The main computational component of rate control is calculating the variance. In order to calculate the variance, we need the mean first, which requires 84480 additions per subband We need a further 84480 subtractions, and 84480 additions per subband to find the variance. This information is summarized for the three coders in the table below, which presents the figures per frame.

Coder type	Mults/Divs	Adds/Subs
Closedloop	760,320	1,520,640
Openloop-interlace	591,360	1,182,720
Openloop-progressive	422,400	844800

This method worked reasonably well when implemented. Occasionally, the quantization parameter needs to be set aright, and this can be done at the beginning of each GOP, while MPEG is computing an I frame, or whenever a scene change is detected. In practice about five out of six subbands coded using this method were coded at bitrates within 0.1 bits/pixel of the target.

At the end of each frame the number of bits required to code that frame is compared with the target bitrate for the frame (which is chosen such that it is constant for each frame, i.e. in our setup it is $bitrate/60$) and the number of difference bits are added to the target bitrate of the next frame This rate is used in order to allocate bits amongst subbands as described in the previous chapter. In this way, if the current frame uses too many bits, then fewer bits are assigned to the next frame, and vice versa. Adding a fraction of the difference bits (instead of all of them) to set the target rate for the next frame works better, since it ensures that the rate control converges slowly to the desired value, and does not keep oscillating around the target rate.

The purpose of providing rate control within this setup was to demonstrate an implementation of our system under real constraints, and thus a lot of time was not spent on fine tuning this mechanism. In this sense our system represents a worst case scenario. With a more elaborate system, tighter rate control as well as better performance should be possible.

Chapter 5

The Experimental Results and Conclusions

This chapter will provide the results of running the openloop and closedloop coders on the two sample sequences shown in figures 5-1 and 5-2. All calculations for bitrates are made based on an original image size of 720 by 1056 pixels and a frame rate of 60 frames per second. Since both the sequences used in this project were originally scanned in at 24 fps, they are rather demanding to code with respect to these reference values — the motion between frames for these images will be larger than that between frames of a sequence shot at 60 fps, and thus MPEG-2 will not be able to motion compensate as effectively.

The 'Balloons' sequence, shown in figure 5-1 does not contain a great amount of spatial detail. Most of the image, such as the two people's clothes and the area within the balloons, and the background consists of relatively flat regions. There is a lot of motion around the man's face, but elsewhere in the image, it is largely restricted to a not very fast camera pan. 'Traffic', on the other hand is very highly detailed spatially, as can be seen from figure 5-2, and there is also a large amount of motion due to the moving cars and camera pan. As can be seen from figure 4-10, is much more difficult to code.

Two constraints must be considered in choosing a bitrate for the low resolution image. First, the low resolution image must be of a viewable quality. Secondly, the

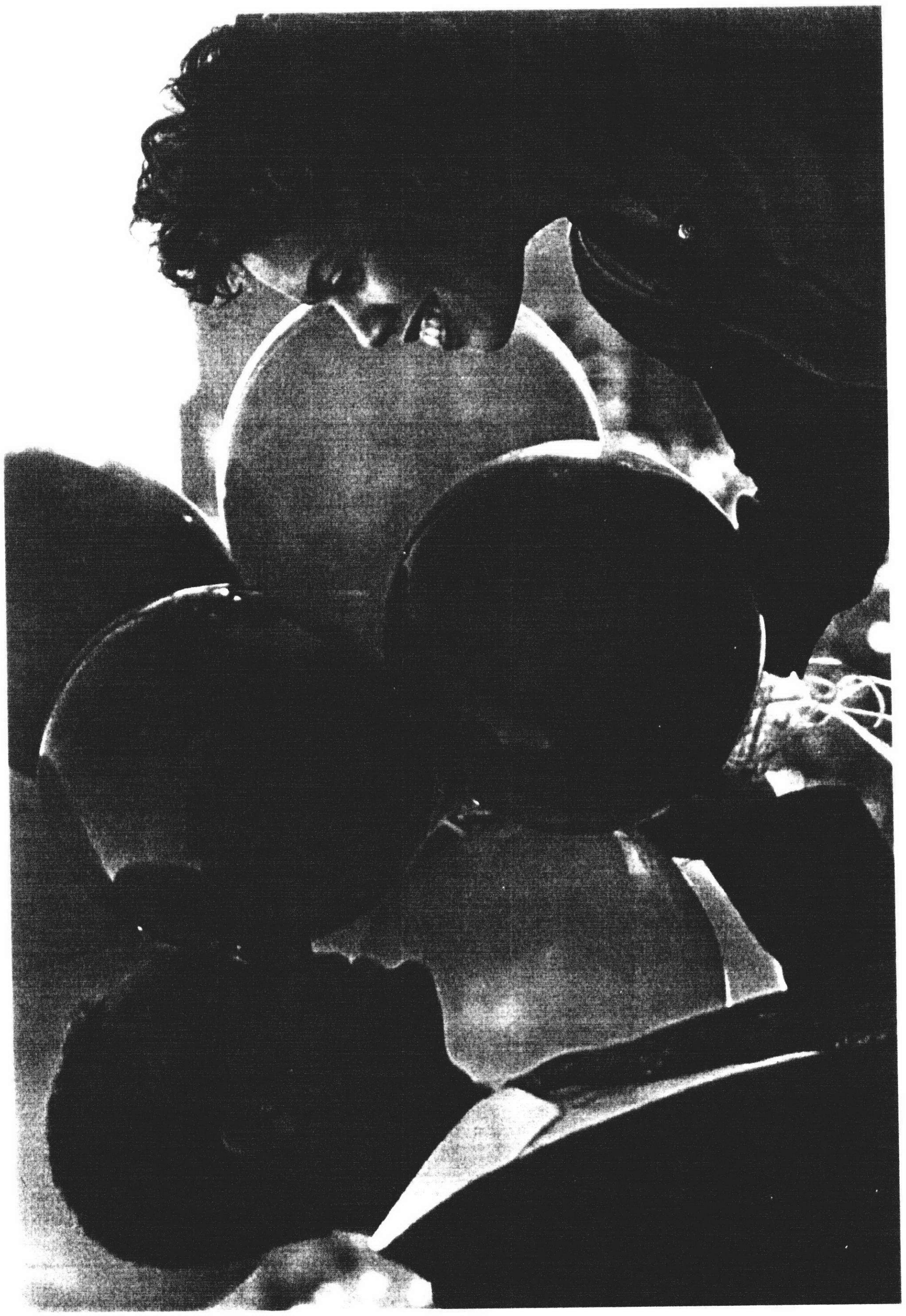


Figure 5-1: First frame of the 'Balloons' sequence

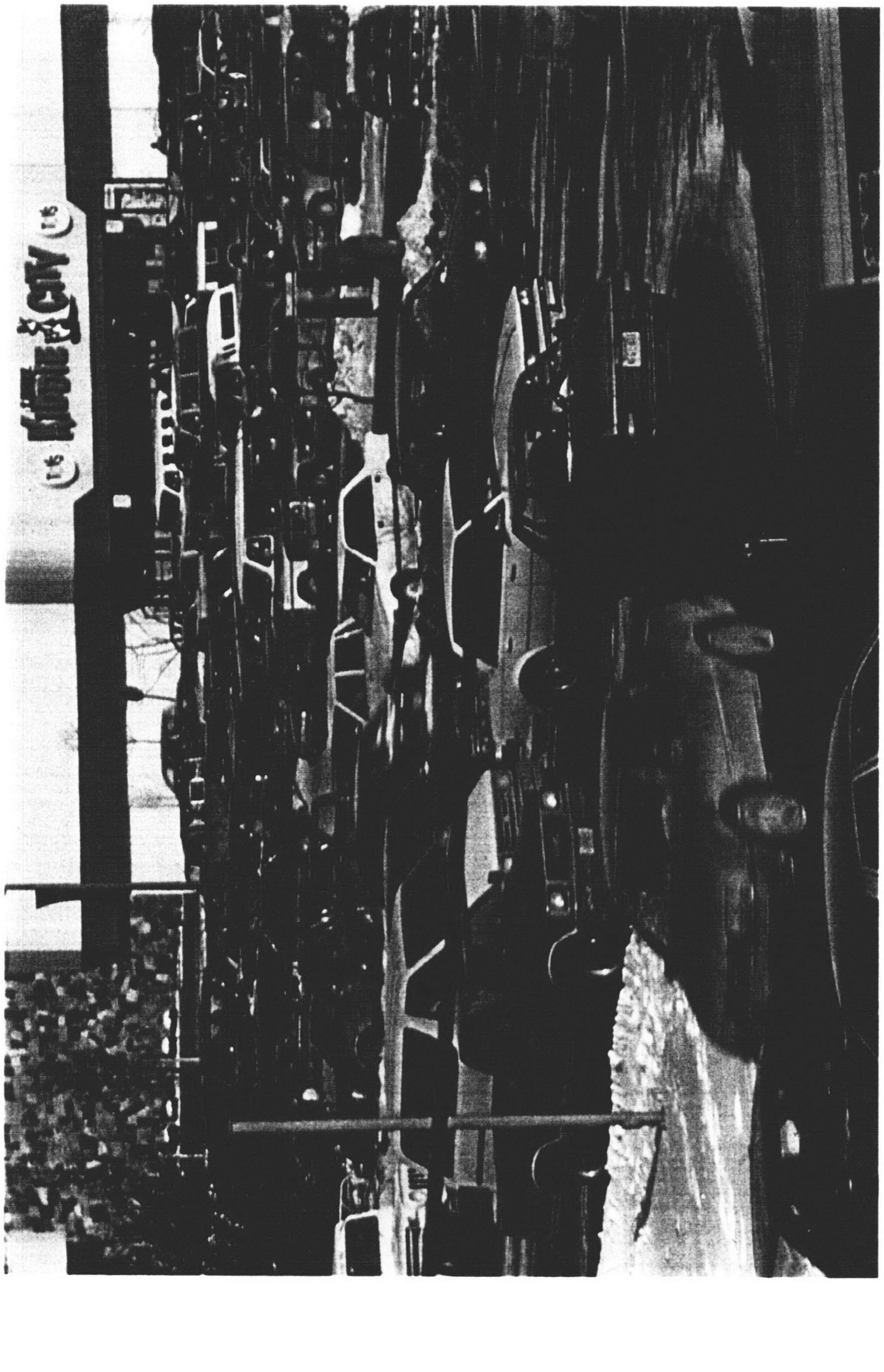


Figure 5-2: First frame of the 'Traffic' sequence

image derived from upsampling it must be of a sufficient quality that enhancement layer coder is capable of correcting it or enhancing it. It was the second requirement which dictated the choice of base layer bitrate in this project, mainly because of the blocking artifacts mentioned on page 92. These artifacts, shown for the two images in figures 5-3 and 5-4 are not very noticeable at the low resolution, but when an image containing even minor such artifacts is upsampled, they become very visible and severely degrade the perceptual quality of the image. The openloop coder cannot compensate for this artifact. Thus in the openloop case a high base bitrate must be chosen. However, there must be an upper limit on the bitrate because of scalability and channel constraints. A base bitrate of 12Mbits/second was chosen for the openloop coder. This was approximately the rate at which, for the images considered, the blocking artifacts, though visible, were not very objectionable.

There is a lot more leeway in choosing the bitrate in the openloop case, since the enhancement layer can correct for the errors made by MPEG. After some experimentation, a base rate of 8Mbits/second was found to provide a good balance between (almost) eliminating blocking artifacts at the full resolution being able to enhance and correct for other parts of the image. The total bitrate for both the openloop and closedloop cases was kept at 20Mbits/second.

For the reasons outlined on page 92, coding progressive sequences at the low resolution is more efficient. Thus, for the experiments with a progressive base layer, the same base layer bitrates were sufficient for good image quality, even though the the amount of data being coded at the base layer in the progressive case is twice the amount being coded in the interlaced case.

Using the above numbers for bitrates, the four coders (openloop-interlaced, openloop progressive, closedloop interlaced, closedloop progressive) were tried on both sequences. The final version of theses coders were settled upon after some experimentation, and represented approximately the best setup given the images. Fourteen frames were coded in each case.

Temporal subbands were not used, since they degraded the performance of the system, given our very crude filters. Equation 4.6 was used to perform the bit alloca-

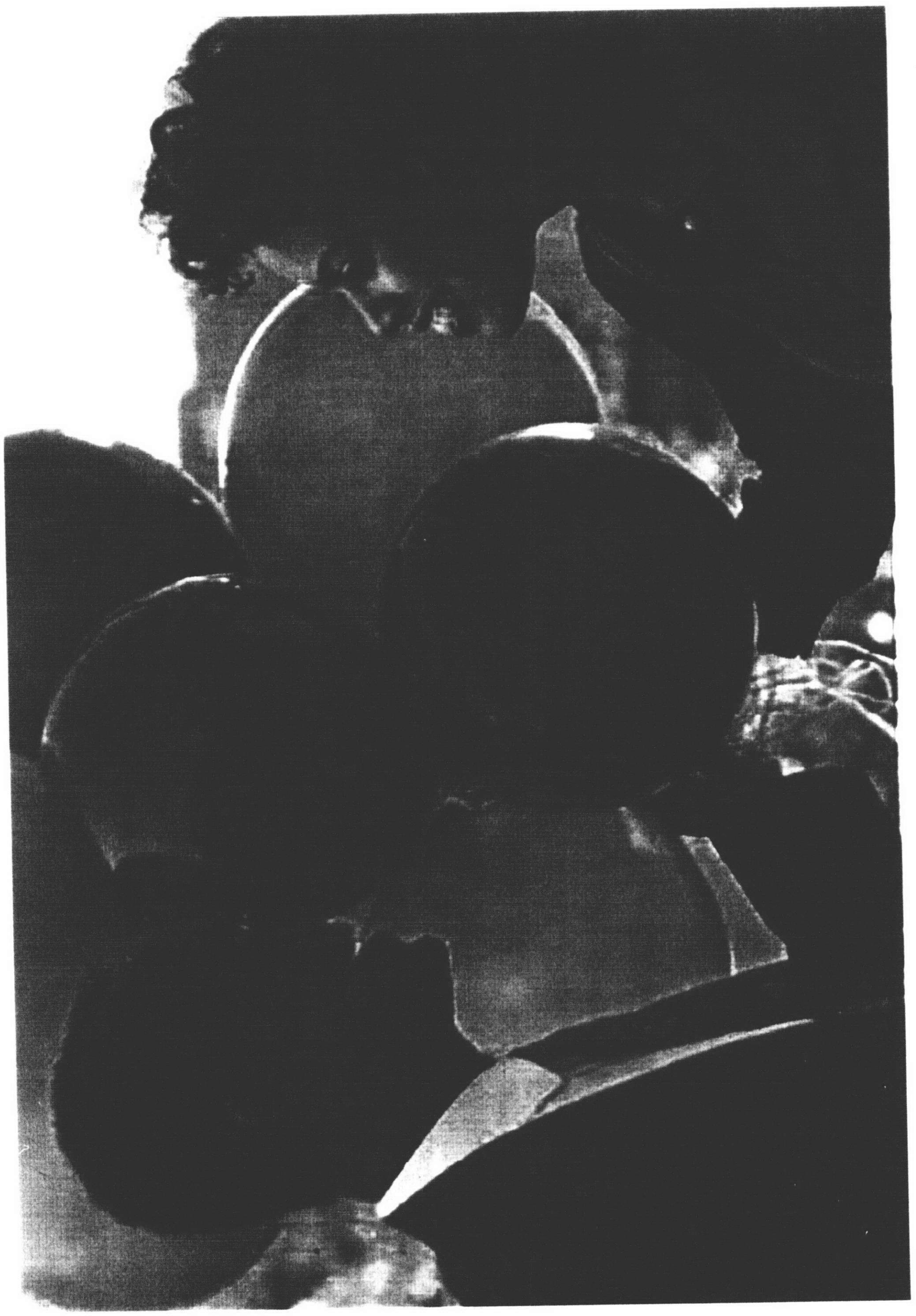


Figure 5-3: Blocking artifacts in 'Balloons'

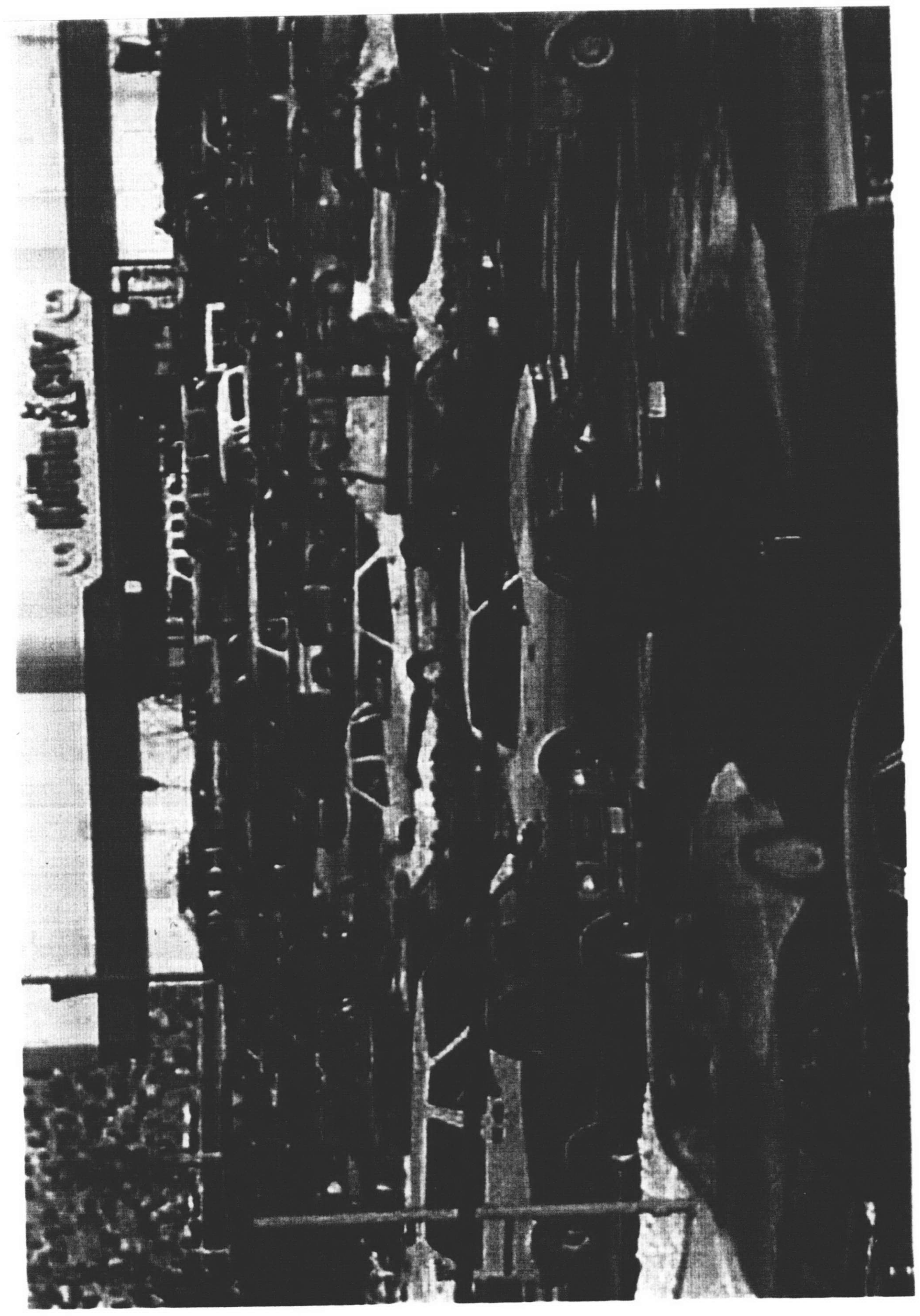


Figure 5-4: Blocking artifacts in 'Traffic'

tion, with the visual weighting factors given in figure 4-15 for the luminance subbands. The chrominance subbands were assigned the weights given in figure 4-15 reduced by one. The best downsampling filters, as described in chapter 4, were chosen.

A dead zone factor of 2 was used in the higher subbands. However, in the experiments that were performed, the higher subbands usually ended up not being assigned any bits at all and were not coded. In the closedloop case, the most energetic subbands of the difference image were the ones which had already been coded by MPEG-2 at the lower resolution and the subbands immediately adjacent to them. Using our bit allocation scheme, they were to only ones coded. This is not surprising given the fact that, (1) in a typical image, most of the energy is contained in the lower spectral regions, and (2) one source of significant error is the aliasing energy leaking across subbands due to the gradual transition regions and relatively low stopband attenuations of the QMF filters shown in figure 4-13 (b).

The main results from the four experiments are shown in tables 5.1 and 5.2.

Coder Type	Openloop Interlaced	Openloop Progressive
	'Balloons' Sequence	
Base bitrate (Mbits/s)	12.00	11.96
Base SNR (dB)	40.64	39.16
Base MSE	3.33	4.52
Total bitrate (Mbits/s)	20.03	20.01
Total SNR (dB)	27.58	29.86
Total MSE	79.02	55.08
	'Traffic' Sequence	
Base bitrate (Mbits/s)	11.96	11.95
Base SNR (dB)	35.44	33.95
Base MSE	10.25	13.87
Total bitrate (Mbits/s)	20.00	19.97
Total SNR (dB)	25.02	27.52
Total MSE	158.48	96.06

Table 5.1: Statistics for the openloop coders

It is evident from these tables that the closedloop coder performs significantly better than the openloop one. The difference between the two is most dramatic in the

Coder Type	Closedloop Interlaced	Closedloop Progressive
	'Balloons' Sequence	
Base bitrate (Mbits/s)	8.01	7.99
Base SNR (dB)	39.36	38.55
Base MSE	4.39	5.18
Total bitrate (Mbits/s)	20.08	20.09
Total SNR (dB)	30.47	30.30
Total MSE	36.06	50.04
	'Traffic' Sequence	
Base bitrate (Mbits/s)	7.96	7.97
Base SNR (dB)	33.64	32.38
Base MSE	15.03	19.80
Total bitrate (Mbits/s)	20.04	20.04
Total SNR (dB)	25.89	27.64
Total MSE	138.73	92.46

Table 5.2: Statistics for the closedloop coders

presence of blocking artifacts, which are visually unpleasant, a fact not appreciated by looking simply at the SNR and MSE. A progressive format at the low resolution seems to work better for the openloop coder. This is as expected, since the more information the MPEG-2 coder has available to it at the base layer, the better a job it can do of coding. It was observed that the upsampled progressive image exhibited significantly lesser blocking artifacts than the upsampled interlaced image. As discussed earlier, this is not surprising given the large amount of relatively uncorrelated energy that is present on account of aliasing in the interlaced image, which tends to overload the MPEG-2 rate control mechanism. In both the closedloop and openloop cases, a progressive base layer format is seen to have a slight advantage in terms of Signal to noise ratio and Mean Squared Error. Perhaps because of the blocking artifacts again, the perceptual difference seemed somewhat greater.

In terms of perceptual quality, all the coding structures, with the exception of the openloop-interlaced coder yielded good looking images at the specified bitrate of 20 Mbits/sec. Despite the relatively low SNRs shown for 'Traffic' in the tables, it did not look bad at all due to a high degree of masking. There was so much detail and motion present in the image that the noise was not very noticeable unless it was specifically looked for. Blocking artifacts, if they were not too severe, appeared as blurs in the moving sequence, and did not look quite as bad as one might be led to believe from looking at a still frame.

The figures presented in tables 5.1 and 5.2are obviously not as impressive as what MPEG-2 would yield for the same bitrates, and they are not meant to be. The idea behind this project was to design a simpler coder which can nevertheless do an effective job of creating a scalable bitstream. However, one useful standard with which to compare these coders is the image quality of MPEG-2 coding the full resolution picture with the bitrate used by the enhancement level coders. Leaving aside the issue of the expense of implementing the MPEG-2 coder for the full resolution image, the MPEG-2 coder alone can be used to provide a scalable bitstream by means of simulcasting, and this provides a good reference point for coder such as those developed here. In order for simulcasting to be equivalent to the openloop case, with the numbers used here, the MPEG-2 coder would be required to code the full resolution image at 8 Mbits/s. For simulcasting to be equivalent to the closedloop coder with the numbers used in this project, it must code the full resolution picture at 12 Mbits/sec. The image qualities resulting from MPEG-2 coding the two sequences used in this project are shown in table 5.3.

Bitrate (Mbits/s)	SNR (dB)	MSE
'Balloons' Sequence		
8	36.82	7.66
12	37.65	6.31
'Traffic' Sequence		
8	30.98	27.90
12	32.51	19.40

Table 5.3: Results of MPEG-2 coding the full sized image at enhancement level rates.

Table 5.3 shows that simulcasting performs significantly better than either the openloop or closedloop coders. However a look at the computational requirements of MPEG-2 at the end of chapter 3 and a glance at the computational complexity of the enhancement layer coder reveals that the number of operations required for motion compensation alone is an order of magnitude greater than the total requirements of the openloop or closedloop enhancement layers. The fact that MPEG-2 performs so much better is thus not very surprising. The closedloop coders developed in this project do a reasonable job of coding quite demanding images at high compression rates, and are a good compromise when the cost of computational complexity is very high.

5.0.5 Directions for Future Work

A number of improvements can be suggested to the setup that was presented in this thesis.

Firstly, there is nothing special about the the image sizes and the ratios between the full resolution and base layer images that were used in this experiment (apart from the fact that the CCIR601 format is commercially very popular). A more complicated ratio than 2/3 will demand a greater number of spatial subbands, and approximately perfect reconstruction filters for them. For a scheme such as the one presented here to become generic it is necessary that higher dimensional QMFs with suitable properties (such as the ones described in section 2.3.1 be developed.

Another line of inquiry might be to explore the coding gain, if any, from using more complicated coding techniques, such as vector quantization, on the subbands. Vector quantization (VQ) provides a gain over the scalar case by exploiting the redundancy of the data being coded. It quantizes vectors of pixel values to points in N dimensional space instead of treating each pixel value as independent. However, VQ gives the most coding gain when the data is strongly correlated. The subbands which were coded in this project all had a relatively highpass character and thus there was relatively weak correlation amongst the pixels. It is open to question whether VQ could have achieved greater redundancy reduction than the entropy coder that were included in the setup.

One interesting variant on this project might be to try a good temporal filter to see how much of a coding gain that yields and whether or not that is worth the extra coding delay. Another interesting thing to try might be to use a vertical-temporal filter when downsampling, so that the region of support of the interlaced sequence is indeed the diamond shaped region indicated in figure 4-1 (a). This would result in increased picture quality at the base resolution, though its effect at the full resolution might not be significant, provided this frequency shape is taken account of when assigning bits to the enhancement layer subbands.

One area which could certainly yield significant benefits is more careful design of the MPEG-2 coder's bit allocation algorithm, so that it may be able to deal with interlaced inputs more effectively. Given the advantage of progressive sequences with respect to MPEG-2 coding, one possible three layer structure might be to transmit the base layer as a progressive sequence, from which decoders can extract an interlaced sequence if desired. This process does the reverse of what was postulated in figure 1-1.

5.1 Conclusion

This work demonstrated the viability of using simple coding techniques built on top of a sophisticated coding 'black box' at low resolutions to derive a scalable two layered bitstream that can be extended to more layers. With the current explosion of video standards around the world, and particularly given the current battle to define a worldwide High Definition TV standard, such an approach can be used to connect across many different formats, applications and image resolutions.

Bibliography

[1] William J. Butera. Multiscale coding of images. Master's thesis, Massachusetts Institute of Technology, 1988.

[2] Test Model Editing Committee. Test model 5. Technical report, ISO-IEC/JTC1/SC29/WG11, April 1993.

[3] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* John Wiley and Sons, 1991.

[4] R. L. Dhei, W. E. Glenn, Karen Glenn, and I.C. Abrahams. Reduced bandwidth requirements for compatible transmission of high definition television. *38th Annual Broadcast Engineering Conference*, pages 297–305, 1984.

[5] D. E. Troxel et. al. Bandwidth compression of high quality images. *International Conference on Communications*, pages 31.9.1–31.9.5, June 1980.

[6] R. E. Crochiere et. al. Digital coding of speech in subbands. *The Bell System Technical Journal*, 55(8), October 1976.

[7] Didier Le Gall. Mpeg: a video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58, April 1991.

[8] Motion Picture Experts Group. Generic coding of moving pictures and associated audio, recommendation h.26x, fourth working draft (brussels). Technical report, ISO/IEC, 1993.

[9] Ellen C. Hildreth and John M. Hollerbach. A computational approach to vision and motor control. Technical Report A.I. Memo 864, MIT Artificial Intelligence Laboratory, August 1985.

[10] Y. Huang and P. Schultheiss. Block quantization of correlated gaussian random variables. *IEEE Transactions on Communication Systems*, September 1963.

[11] David H. Hubel and Torsten N. Wiesel. *The Mind's Eye.* W. H. Freeman and Company, 1986. Readings from Scientific American, see chapter on Brain Mechanisms of Vision.

[12] D. A. Huffman. A method for the construction of minimum redundancy codes. In *Proceedings of the IRE*, pages 1098–1101, 1952.

[13] Howard C. Reeve III and S. Lim Jae. Reduction of blocking effects in image coding. *Optical Engineering*, 23(1), January / February 1984.

[14] ISO/IEC 13818-2. *Generic Coding of Moving Pictures and Associated Audio, Recommendation H.262*, November 1993. Committee Draft.

[15] ITU, CCIR. *Recommendation 601: Encoding Parameters of Digital Television for Studios*, 1986.

[16] N. S. Jayant and Peter Noll. *Digital Coding of Waveforms.* Prentice Hall, Englewood Cliffs, New Jersey 07632, 1984.

[17] Nikhil Jayant, James Johnston, and Robert Safranek. Signal compression based on models of human perception. *Proceedings of the IEEE*, 81(10), October 1993.

[18] Pasquale Romano Jr. Vector quantization for spatiotemporal sub-band coding. Master's thesis, Massachusetts institute of Technology, 1990.

[19] V. Michael Bove Jr. Three dimensional subband coding and interlace. Technical report, MIT Media Laboratory, 1991.

[20] Roger G. Kermode. Requirements for real time digital video compression. Technical report, M.I.T. Meida Laboratory, July 1993. Report prepared for Digital Sight and Sound Group, Silicon Graphics Inc.

[21] Jae. S. Lim. *Two-Dimensional Signal and Image Processing.* Prentice Hall, Englewood Cliffs, New Jersey 07632, 1990.

[22] Andrew Lippman and V. Michael Bove Jr. Atv profile. Technical report, ISO: Coding of Motion Pictures and Associated Audio, 1992.

[23] John N. Little and Loren Shure. *Signal Processing Toolbox for Use with MATLAB.* The Math Works Inc., 1993.

[24] J. Makhoul, S. Roucos, and H. Gish. Vector quantization in speech coding. *Proceedings of the IEEE*, 73, November 1985.

[25] David Marr. *Vision.* W. H. Freeman and Company, 1982.

[26] H. Meyr, Hans G. Rosdolsky, and Thomas S. Huang. Optimum run length codes. *IEEE transactions on Communications*, COM-22(6):826–835, June 1974.

[27] Mark Nelson. *The Data Compression Book.* M & T Books, 1991.

[28] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing.* Prentice Hall, Englewood Cliffs, New Jersey 07632, 1989.

[29] Donald Edwin Pearson. *Transmission and Display of Pictorial Information.* Wiley, Halstead Press, 1975.

[30] William B. Pennebaker and Joan L. Mitchell. *JPEG Still Image Data Compression Standard.* Van Rostrand Reinhold, 1993.

[31] Ashok C. Popat. Scalar quantization with arithmetic coding. Master's thesis, Massachusetts institute of Technology, 1990.

[32] W. F. Schreiber. Pychophysics and the improvement of television image quality. *SMPTE Journal*, pages 717–725, August 1984.

[33] C. E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. Journal*, 27, July 1948.

[34] Eero Peter Simoncelli. Orthogonal sub-band image transforms. Master's thesis, Massachusetts Institute of Technology, 1988.

[35] J. O. Smith. *Techniques for Digital Filter Design and System Identification with Application to the Violin.* PhD thesis, Stanford University, 1983.

[36] James A. Storer. *Data Compression: Methods and Theory.* Computer Science Press, 1988.

[37] Kenji Tsunashima, Joseph B. Stampleman, and V. Michael Bove Jr. A scalable motion-compensated subband image coder. *IEEE Transactions on Communications*, 1992.

[38] P. P. Vaidyanathan. Quadrature mirror filter banks, m band extensions and perfect-reconstruction techniques. *IEEE ASSP Magazine*, pages 4–20, July 1987.

[39] Nuno Vasconcelos. Library based image coding using vector quantization of the prediction space. Master's thesis, Massachusetts Institute of Technology, 1993.

[40] Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6), June 1987.