NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

An Algorithm and Architecture for Video Compression

Fayez M. Idris

A thesis

submitted to the school of Graduate Studies and Research

in partial fulfillment of

the requirement for the degree of

Master of Applied Science

in

Electrical Engineering

Ottawa-Carleton Institute of Electrical Engineering

Department of Electrical Engineering

Faculty of Engineering

University of Ottawa

Ottawa, Ontario, K1N 6N5

ISBN 0-315-89604-3

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

To the memory of my uncle, Mahmoud Idris

I hereby declare that I am the sole author of this theses.

I authorize the University of Ottawa to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Fayez M. Idris

I further authorize the University of Ottawa to reproduce this thesis by photocopying or other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Fayez M. Idris

Abstract

In this thesis, we present a new frame adaptive vector quantization technique and an architecture for real-time video compression. Video compression is becoming increasingly important with several applications. There are two kinds of redundancies in a video sequence namely, spatial and temporal. Vector quantization (VQ) is an efficient technique for exploiting the spatial correlation. The temporal redundancies are usually removed by using motion estimation/compensation techniques. The coding performance of VQ may be improved by employing adaptive techniques at the expense of increases in computational complexity. We propose a new technique for video compression using adaptive VQ (VC-FAVQ). This technique exploits the interframe as well as intraframe correlations in order to reduce the bit rate. In addition, a dynamic self organized codebook is used to track the local statistics from frame to frame. Computer simulations using standard CCITT video sequences were performed. Simulation results demonstrate the superior coding performance of VC-FAVQ.

We note that both VQ and motion estimation algorithms are essentially template matching operations. However, they are compute intensive necessitating the use of special purpose architectures for real-time implementation. Associative memories are efficient for template matching in parallel. We propose an unified associative memory architecture for real implementation of VC-FAVQ. This architecture is based on a novel storage concept where image data is stored by association rather than by contents. The architecture has the advantages of simplicity, partitionability and modularity and is hence suitable for VLSI implementation.

Acknowledgments

First of all, I wish to express my gratitude and appreciation to my supervisor, Dr. S. Panchanathan. His guidance, encouragement and support were the driving force for the successful completion of my thesis work. Special thanks are due to Dr. Morris Goldberg for his valuable comments and suggestions.

The co-operation of the members of the Multimedia Communications Research Laboratory, especially, R. Gandhi, G. Iyengar and Dr. M. Brahmanandran is appreciated.

I am thankful for the encouragement and help of the Jordanian students at Ottawa-Carleton Institute of Electrical Engineering. A special note of thanks is due to M. Banat, K. El-baher and N. Ghozlan.

Thanks to the staff of the Department of Electrical Engineering, University of Ottawa, for their cheerful help.

The financial support from the Canadian International Development Agency and the Jordan University of Science and Technology for my Master's program is gratefully acknowledged.

My sincere thanks to Mr. A. Atiyat and his family for their support.

Finally, I am grateful to my beloved wife Sawsan and my family. It would not have been possible to finish this work without their constant support and understanding.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Image/video compression is becoming increasingly important with the advent of broadband networks (ISDN, ATM, etc.) and compression standards (JPEG, MPEG, H.261, etc.). Typical image/video compression applications are in the areas of high-definition television (HDTV), image/video data bases, multimedia communications, teleconferencing, telepresence, etc.[22].

In image processing, it is often necessary to represent the image in digital form. This is done by first sampling a continuous image in time to obtain the frames. The individual frames are then sampled into $N_1 \times N_2$ pixels (picture elements) as shown in Figure 1.1. We note that $(\Delta x, \Delta y)$ and Δt are the sampling intervals in the spatial and temporal dimensions, respectively. Each pixel is then quantized into one of 2^b gray levels $(1, 2, \cdots, 2^b)$. However, this process results in a large data rate. For example, the data rate for a 512×512 television color image sequence at 8 bits/pixel per color with a frame rate of 30 frames/second is 1.8×10^8 bits/second. The large channel bandwidth and memory requirements for the transmission and storage images necessitates the use of efficient image/video compression techniques.

The goal of image/video compression is to reduce the number of bits required to represent an image while maintaining an acceptable image fidelity. Efficient compression is achieved by exploiting the spatial and temporal redundancies in the video sequence. The spatial redundancies are removed by using intraframe techniques, while the temporal

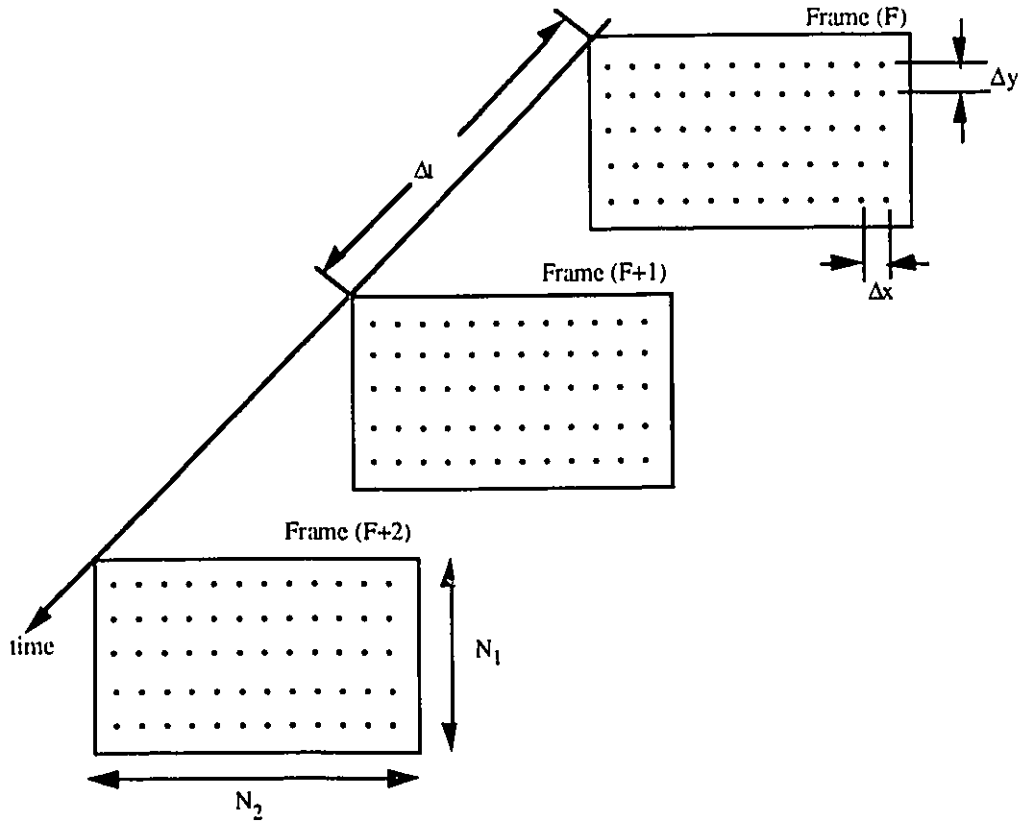


Figure 1.1: Sampling of a video sequence.

correlations are usually removed using interframe techniques.

Several intraframe and interframe image/video coding techniques based on scalar quantization have been reported in the literature [2]. Examples include, differential pulse code modulation (DPCM), transform coding, and hybrid coding. According to Shannon's rate distortion theory [16, 19], a better performance can be achieved by coding vectors instead of scalars, even though the source is memoryless. Several vector quantization techniques for speech[18] and image coding[21] applications have been developed in recent years. Nasrabadi *et al* [22] have presented a review of the basic vector quantization techniques and many of their variations for image/video compression.

In VQ, a set of representative images (training set) is decomposed into $L-$dimensional vectors. An iterative clustering algorithm such as the LBG algorithm [23] is used to gen-

erate a codebook $CB = \{W_i; i = 1, \cdots, N\}$, where N is the size of the codebook. This codebook is then made available at both the transmitter and the receiver. In the encoding process, the image to be coded is decomposed into $L-$dimensional vectors. For each input vector $V_i = \{v_{i1}, v_{i2}, \cdots, v_{iL}\}$, the codebook CB is searched using a nearest neighbour rule to find the closest codeword W_j. Compression is achieved by transmitting the label (index) j corresponding to W_j. Reconstruction of images is implemented by using j as an address to a table containing the codewords.

The computational complexity of VQ for K input vectors of dimension L and a codebook size N is $O(KLN)$. For example, a 512×512 image with vector dimension of $L = 16$ encoded using a codebook of size $N = 256$ requires approximately 192 million arithmetic operations. This high computational complexity has been an impediment for real time implementation in many applications. Recently, special purpose architectures that implement VQ in real time have been reported in the literature [82]-[93]. The first class of architectures is based on faster execution of the distortion measure. For example, Davidson *et al* [84] have presented several systolic architectures for VQ which provide a tradeoff between speed and VLSI chip area. A second class of architectures is based on the restriction of the search space for each input vector to a smaller subcodebook. For example, Ramamoorthy *et al* [90] have presented a bit-serial systolic architecture for multi-stage VQ. In multi-stage VQ, a smaller codebook size is used at each stage instead of a large codebook, hence reducing the computational complexity. The third class of architectures is based on the use of fast search methods. For example, at the expense of some degradation in quality, architectures that implement tree structured VQ [85]-[88] have been proposed. We note that these architectures implement algorithms which exploit the spatial correlation. We refer to them as intra-image architectures. A straight forward extension of the intra-image architectures for video compression may not exploit the temporal correlations present in an image/video sequence.

The temporal redundancies in an image/video sequence are usually removed using motion estimation/compensation techniques. Several motion estimation algorithms have been reported in the literature. Examples include, pel-recursive [3], block matching [12],

etc. The block matching algorithm is widely used because of its simplicity. Here, the current frame is divided into non-overlapping reference blocks. Each reference block in the current frame is compared with candidate blocks from a search area in the previous frame in order to obtain the best match. The offset between the reference block and the best match candidate block specifies the displacement (motion) vector.

Recently, architectures which implement the full search block matching algorithm have been reported in the literature [99]-[103]. These architectures differ in terms of the mapping strategy and the number of processing elements employed. We note that the motion estimation architectures are essentially inter-image architectures.

Typically, a combination of the inter- and intra-image architectures are used in video compression applications. This combined architecture may result in inefficiencies in implementation as the common elements between the two algorithms and architectures are not fully exploited. In addition, these designs are complex and/or non-cascadable which results in an increased hardware requirement and reduced flexibility in VLSI implementation.

In this thesis, a reduced complexity adaptive technique and an architecture for video compression are presented: *Video Compression using Frame Adaptive Vector Quantization (VC-FAVQ)* [110]; and *Associative Memory Architecture for Video Compression* [109].

The VC-FAVQ coder, exploits the inter-/intraframe correlations and provides frame adaptability at a reduced complexity which makes possible real time implementation. Computer simulations demonstrate the superior coding performance of VC-FAVQ compared to other techniques reported in the literature [64]. In addition, VC-FAVQ is highly adaptive and is elegant for constant quality and fixed bit rate applications.

The associative memory architecture is based on a novel storage concpt which implements both motion estimation and frame adaptive vector quantization algorithms in a single structure. The proposed architecture has the advantages of simplicity, partitionability and modularity; and is hence suitable for VLSI implementation.

The rest of this thesis is organized as follows: Chapter 2 presents a review of image/video compression and vector quantization techniques. This follows in chapter 3 with

a review of architectures for vector quantization and block matching. The VC-FAVQ technique is presented in chapter 4. The associative memory architecture for real time video compression is detailed in chapter 5. Finally, the conclusions and suggestions for future research work is presented in chapter 6.

Chapter 2

Review of Image/Video Compression and Vector Quantization

In this chapter the concepts of lossless and lossy image data compression methods are presented. Intraframe and interframe image coding techniques are then discussed in sections 2.2 and 2.3. Emphasis is placed on the review of vector quantization methods as the proposed technique is essentially based on VQ. In section 2.4, we present a review of video compression standards. A comparison of the different coding techniques is presented in section 2.5 followed by a summary in section 2.6.

2.1 Image/Video Data Compression

The large memory capacity and channel bandwidth requirements for the storage and transmission of visual data necessitates the use of efficient image/video compression techniques. The goal of image/video data compression is to reduce the number of bits required to represent an image/video signal while maintaining an acceptable fidelity [2].

Image/video compression is essentially a redundancy removal process. Efficient compression is achieved by exploiting the spatial, temporal and psychovisual redundancies.

The neighbouring pixels in an image/video frame are usually highly correlated. For example, there is a strong dependency between the pixel values in a local region and hence a pixel value can be inferred from the neighbouring pixel values. This correlation (dependency) is called the spatial redundancy, and is typically removed by employing intraframe coding techniques such as predictive coding, transform coding, etc.

Temporal redundancy refers to the correlation between the successive frames in an image/video sequence. The differences between successive frames are due to object motion, camera motion, panning and zooming [2]. Temporal redundancy is usually removed by employing interframe compression techniques such as motion estimation/compensation, frame replenishment, etc.

Most image/video frames contain psychovisual redundancies; that is, some information may be removed without sacrificing the subjective image quality. Therefore, if the image/video signal is encoded based on the properties of the human visual system, a higher compression ratio may be achieved [70, 71]. Some of the properties of the human visual system that may be exploited to a great advantage are:

- Greater sensitivity to small signal changes in dark areas.
- Lower sensitivity to random noise compared to systematic noise.
- Lower sensitivity to faster moving objects.
- Lesser perception of distortion at higher spatial frequencies.

Image/video data compression techniques can be broadly classified into: "distortionless" or "lossless" and "minimum distortion" or "lossy" schemes.

Lossless Coding

Lossless coding is concerned with minimizing the average number of bits per pixel without any loss in image quality; i.e. the decoder should reconstruct the exact input image from the encoded image. Information theory states that the source can be exactly encoded with

H bits/pixel, where H is the source entropy. For a source with 2^b possible independent symbols with probabilities p_i, $i = 1, 2, \cdots, 2^b$, the entropy is given by:

$$H = -\sum_{i=1}^{2^b} p_i \log_2 p_i \tag{2.1}$$

This results in a variable length code (VLC) where shorter codewords are assigned to more probable pixel values and longer codewords are assigned to less probable pixel values. Huffman [16] and Arithmetic coding [57] are the most popular approaches for lossless coding. Arithmetic coding achieves higher compression ratios than Huffman coding, but it is more difficult to implement. Another technique for lossless coding is run length coding [17] in which a reduction in the bit rate is achieved by sequentially transmitting or storing the pixel values (run) followed by the number of its repetitions (length). Significant compression is possible if the input image is characterized by long runs.

Lossy Coding

The purpose of minimum distortion or lossy coding is to minimize the bit rate for a given average distortion or equivalently, to minimize the average distortion for a given bit rate. For an image source X, the average distortion D is defined as

$$D = E\{d(X, \hat{X})\} \tag{2.2}$$

where $d(X, \hat{X})$ is a distortion measure between the source X and its reproduction $\hat{X}$. Clearly, the design of such an encoding scheme depends on the statistics of the source X and the characteristics of the distortion function d [4]. Rate distortion theory [16] provides the theoretical lower bound on the bit rate of any quantizer. For the source X with a probability function $p_X(X)$, the rate distortion function is defined as

$$R(D^*) = min\{I(X, \hat{X})\} \tag{2.3}$$

where the minimum is taken over all the encoding schemes which result in average distortion D less than some value D^*, and $I(X, \hat{X})$ is the average mutual information between X and $\hat{X}$ and is defined as

$$I(X, \hat{X}) = \sum_{x} \sum_{\hat{x}} p_X(x) p_{\hat{X}/X}(\hat{x}/x) \log \frac{p_{\hat{X}/X}(\hat{x}/x)}{p_{\hat{X}}(\hat{x})} \tag{2.4}$$

where $p_{\hat{X}/X}(\hat{x}/x)$ is the conditional probability for $\hat{X}$ given X. This definition of the rate distortion function indicates that for a given average distortion D, the minimum transmission rate is $R(D)$. Shannon's coding theorem states that it is possible to design an encoding system which achieves the average distortion D at a transmission rate arbitrarily close to $R(D)$. Although the theory does not detail the design of such an encoding system, it is valuable in comparing the performance of different encoding schemes.

2.2 Image Compression Techniques

We recall from section 2.1 that in image compression, the spatial redundancies are typically removed by employing intraframe coding techniques. Examples include, predictive coding, transform coding, vector quantization, and hybrid techniques. In this section, the various intraframe image coding methods are reviewed, with a special emphasis on vector quantization.

2.2.1 Predictive Coding

Predictive coding exploits the mutual redundancy between neighbouring pixels. Rather than encoding the pixel intensity directly, its value is first predicted from the previously encoded pixels. The predicted pixel value is then subtracted from the actual pixel value and the difference (prediction error) is quantized and coded for transmission. The quantized prediction error is used at the receiver to reconstruct the image.

Differential Pulse Code Modulation

Differential Pulse Code Modulation (DPCM) is the most commonly used predictive coding technique. A block diagram of the DPCM system is shown in Figure 2.1. The transmitter consists of two major components, namely: the quantizer, and the predictor. The predictor estimates the value of the pixel to be encoded using the values of the previously transmitted pixels. The difference e_k, known as the prediction error, between the actual pixel value x_k and the predicted value $\hat{y}_k$ is quantized, coded and transmitted.

The predictor can be linear or nonlinear. A nonlinear predictor which involves a nonlinear combination of the previously encoded pixels is difficult to implement; and hence in most DPCM systems a linear predictor is employed. A linear prediction of x_k can be determined using the following equation:

$$\hat{y}_k = \sum_i a_i x_{k-i} \tag{2.5}$$

where a_i are the weighting coefficients. The optimal predictor in terms of mean square error is obtained by selecting the coefficient a_i so that the variance of the prediction error is minimized. The optimal coefficients can be determined by solving the following set of equations:

$$R(j) = \sum_i a_{i,opt} R(j-1) \tag{2.6}$$

where $R(j)$ is the correlation function.

Adaptive DPCM

The coding performance of a DPCM system can be improved by employing adaptive techniques which tracks the local statistics of the input image. Adaptability can be achieved by either fixing the characteristics of the quantizer and varying the predictor parameters, or by varying the characteristics of the quantizer and fixing the parameters of the predictor. For example, the weighting coefficients in Equation 2.5 can be modified periodically, or the quantizer levels can be changed according to the input image statistics[15].

2.2.2 Transform Coding

Transform coding achieves compression in the transform domain, rather than the spatial domain. For still images, the input image is first divided into non-overlapping blocks X_{m_1,m_2} of $M_1 \times M_2$ pixels. In order to decorrelate the image data, a two dimensional transform is then applied to X_{m_1,m_2} as shown in Figure 2.2. The transformation maps X_{m_1,m_2} into a two dimensional array $Q_{u,v}$ of transform coefficients with the same dimen-

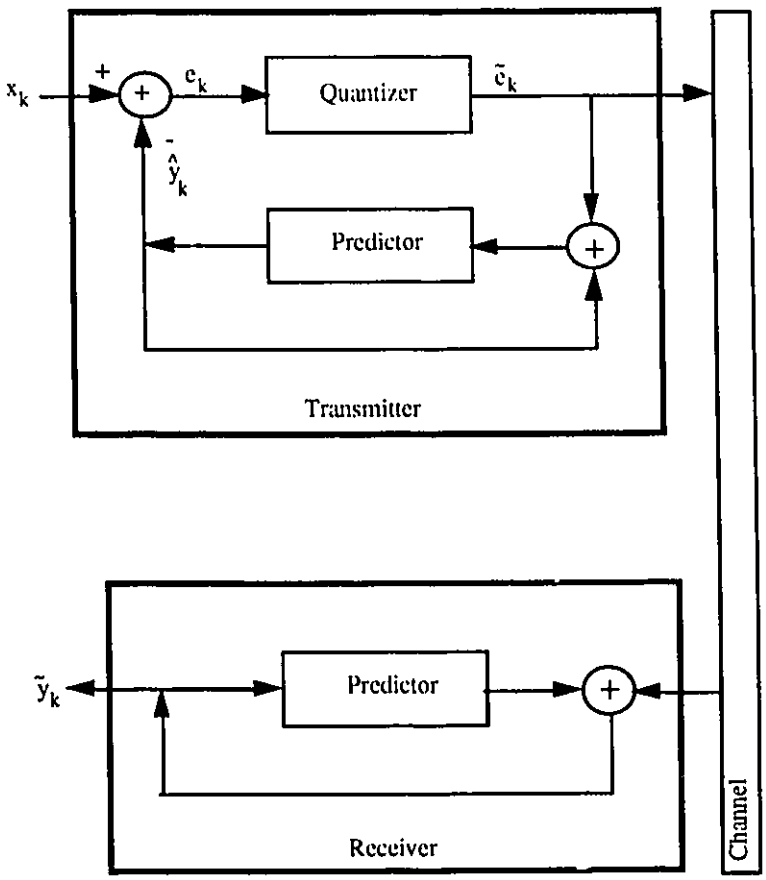


Figure 2.1: Block diagram of a DPCM coding system.

sion. Mathematically this operation is given by:

$$Q_{u,v} = \sum_{m_1=0}^{M_1} \sum_{m_2=0}^{M_2} X_{m_1,m_2} A_{u,v;m_1,m_2} \tag{2.7}$$

where $A_{u,v;m_1,m_2}$ is the transform kernel. The resulting coefficients $Q_{u,v}$, $u = 1, 2, \cdots M_1$, $v = 1, 2, \cdots, M_2$ are then quantized, coded and transmitted. At the receiver, an inverse transform operation is applied to the quantized coefficients $\tilde{Q}_{u,v}$ to reconstruct the image:

$$\hat{X}_{m_1,m_2} = \sum_{u=0}^{M_1} \sum_{v=0}^{M_2} \tilde{Q}_{u,v} A^{-1}_{u,v;m_1,m_2} \tag{2.8}$$

where $A^{-1}_{u,v;m_1,m_2}$ is the inverse transform kernel.

An optimum transform should result in statistically independent coefficients [6]. Karhunen Loeve transform is an optimum transform in terms of both the mean square

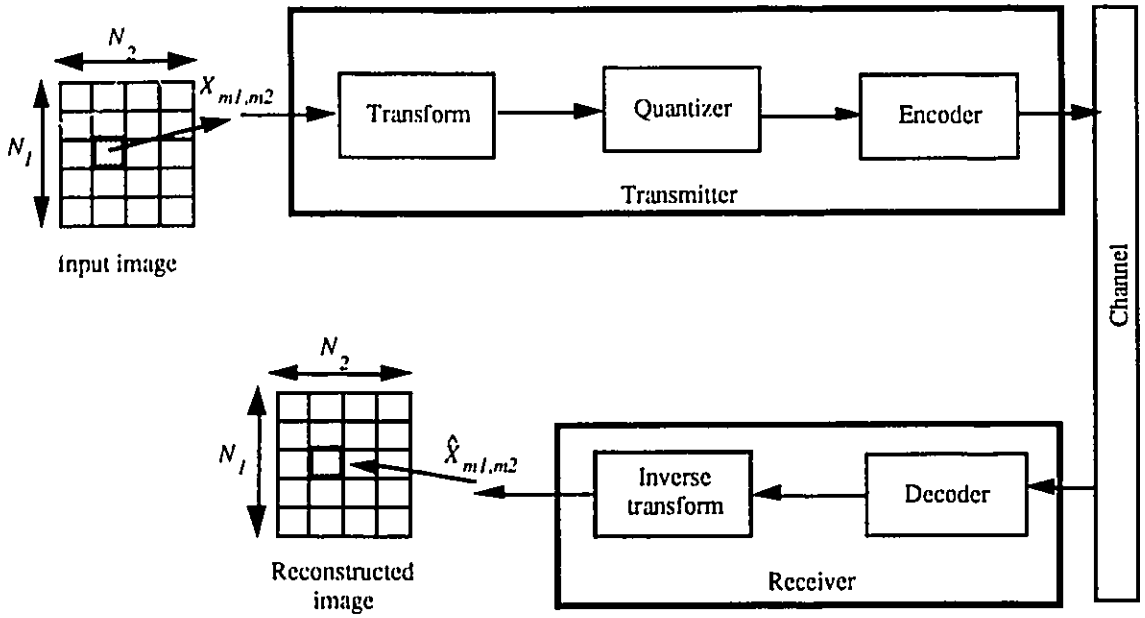


Figure 2.2: Block diagram of a transform coding system.

error and subjective quality. However, it requires a large number of operations to compute; and is hence usually replaced by suboptimal transforms[15], such as Fourier transform, cosine transform, or Hadamard transform. It has been shown that the performance of the cosine transform[5] is close to the optimal Karhunen Loeve transform.

Adaptive Transform Coding

In adaptive transform coding [10, 15], one or more of the coder parameters are changed in a way such that the coder better matches the local image statistics. For example, each block is classified into one of several categories according to some activity index. Adaptivity is achieved by allocating a larger number of bits to blocks with high activity index and fewer bits to blocks with lower activity index.

2.2.3 Hybrid Transform/Predictive Image Coding

Hybrid transform/predictive coding [2] is a combination of transform and predictive coding. As shown in Figure 2.3, the basic configuration is a transform operation followed

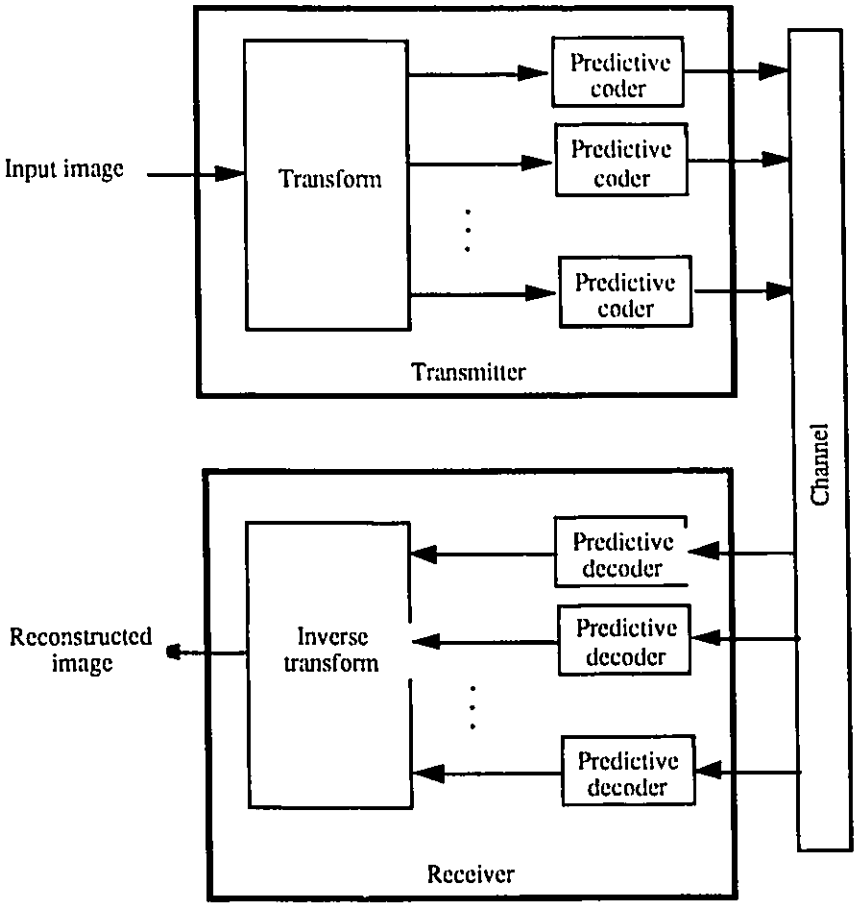


Figure 2.3: Hybrid transform/predictive coding system.

by a predictive coder for each transform coefficient. Typically, one or two dimensional transform is applied on subimages, the coefficients are then coded using the previous coefficients as a prediction estimate.

2.2.4 Image Compression using Vector Quantization

Vector quantization (VQ) is an efficient technique for low bit rate image coding [22]. In vector quantization [21], an L-dimensional vector $V_i = \{v_{i1}, v_{i2}, \cdots, v_{iL}\}$ is mapped into another L-dimensional vector W_j:

$$q : V_i \rightarrow W_j \tag{2.9}$$

where W_j is one of a finite set (codebook) of reproduction vectors (codewords) $CB = \{W_1, W_2, \cdots, W_N\}$, where $W_i = \{w_{i1}, w_{i2}, \cdots, w_{iL}\}$. In other words, vector quantization involves the partitioning of the L-dimensional Euclidean space into N decision regions $\{\Re_i, i = 1, 2, \cdots, N\}$, each containing one of the N reproduction vectors or codewords W_i. The vector V_i is quantized as W_j if it is in the region $\Re_j$,

$$q : V_i \rightarrow W_j \qquad if\ V_i \epsilon \Re_j \tag{2.10}$$

which implies that the mapping is completely characterized by the partition $\Re_i$.

If $d(V_i, W_j)$ is a distortion measure which represents the error when V_i is reproduced by W_j, then the optimal quantizer is the one which minimizes $d(V_i, W_j)$ over all other quantizers. The quantizer performance can be measured by the average distortion:

$$D = E\{d(V, W)\} \tag{2.11}$$

$$= \sum_{i=1}^{N} p(V \epsilon \Re_i) E\{d(V, W_i) | V \epsilon \Re_i\} \tag{2.12}$$

It follows that the optimal quantizer must satisfy the following conditions:

1. The selection rule should be a minimum distortion or nearest neighbour rule; i.e. :

$$q(V_i) = W_j \quad iff \quad d(V_i, W_j) \leq d(V_i, W_k) \quad for\ all\ k \tag{2.13}$$

where $q(.)$is the quantization operation.

2. The codewords $\{W_i, i = 1, 2, \cdots, N\}$ are determined such that the average distortion in each region is minimized. (i.e. the codewords are the centroids of the decision region).

The steps involved in VQ as applied to image/video compression are vector formation, codebook design, and quantization.

Vector Formation

The first step in VQ is to decompose the input image into vectors. The image is partitioned into two dimensional blocks of equal or variable sizes. The features or values are extracted

from the blocks and then rearranged into vectors. Various vector formation schemes have been proposed. For example, the vectors can be formed from the original pixel values of the blocks [29]; the transform coefficients of a block of pixels [34, 37]; the prediction error of a block [30, 31]; the pixel values of a block normalized by the average [38]; the intensity normalized by the mean and standard deviation [58]; and the color components of a pixel [69].

Codebook Design

Linde *et al* [23] have presented an algorithm for codebook design based on the two conditions for optimality, referred to as the generalized Lloyd or the LBG algorithm. In this algorithm, given an initial codebook, each training vector is assigned to its nearest neighbour codeword. Each codeword is then modified to minimize its distortion relative to the vectors assigned to it. This process continues iteratively until the change in distortion between two successive iterations is within a threshold of acceptance. The algorithm is described as follows:

1. Given an initial codebook, $CB_0 = \{W_i, i = 1, 2, \cdots, N\}$, a threshold $\epsilon \geq 0$, and a training set $\{V_i, i = 1, 2, \cdots, K\}$, set m to 0 and D_{-1} to ∞.

2. Assign each input vector to its nearest neighbour codeword:

$$V_i \epsilon W_{j,m} \quad iff \quad d(V_i, W_{j,m}) \leq d(V_i, W_{k,m}) \quad for\ all\ j \neq k \tag{2.14}$$

3. Find CB_{m+1}, by computing the centroids of the training vectors assigned to each codeword:

$$W_{i,m+1} = \frac{1}{M_{i,m}} \sum_{V_j \in W_{j,m}} V_j \qquad i = 1, 2, \cdots, N \tag{2.15}$$

   where $M_{i,m}$ is the number of vectors assigned to $W_{j,m}$.

4. Compute the average distortion:

$$D_m = \frac{1}{N} \sum_{I=1, W \in CB_m}^{N} D(V_j, W) \tag{2.16}$$

   if D_m relative to D_{m-1} is less than $\in$, then stop; otherwise, go to step 2.

To obtain the initial codebook CB_0, one possible approach is to select the first, or the evenly spaced N vectors as an initial codebook. Alternatively, one might use the splitting algorithm [23], where the centroids of the training set is calculated and split into two codewords. The LBG algorithm is applied to yield a codebook of two codewords. Each codeword is then split into two codevectors to yield a codebook of four codewords. This procedure is repeated until an N-level codebook is constructed.

Note that the LBG algorithm may converge to a local minimum [23]. Furthermore, the solution is not unique and depends upon the initial codebook. Simulated annealing [26] [27] is a procedure which introduces randomness in each iteration of the LBG algorithm can be used to avoid the local minima. However, this procedure is computationally intensive [26]. Recently, techniques based on neural networks [24, 25] and the pairwise nearest neighbour (PNN) algorithm [28] have been reported as alternatives for codebook design. In the neural network approach, the weights (which represent the codewords) between the neurons are adaptively adjusted after the presentation of each vector. The main advantage of neural network is that it converges to an asymptotic value faster than the LBG algorithm [25]. In the PNN approach, each of the K training vectors is considered as a separate cluster. We start with the K clusters and merge together the two clusters which result in the minimum increase in average distortion. This yields $K-1$ clusters. The merging process continues until only N clusters remain. PNN is faster than the LBG algorithm, however the LBG algorithm results in a codebook that satisfies the two conditions for optimality.

Computational and Storage Complexity

The encoding process in VQ is executed as follows: For each input vector V_i, the distortion between V_i and the codewords, $\{W_j, 1 \leq j \leq N\}$, is computed. The codeword with the minimum distortion is selected as the quantized version of the input vector, V_i, as shown in Figure 2.4. Hence, the basic computation is a distortion calculation between an input vector V_i and a codeword W_j which requires the computation of L elemental distortion operations. The elemental distortion operation can be defined based on the distortion

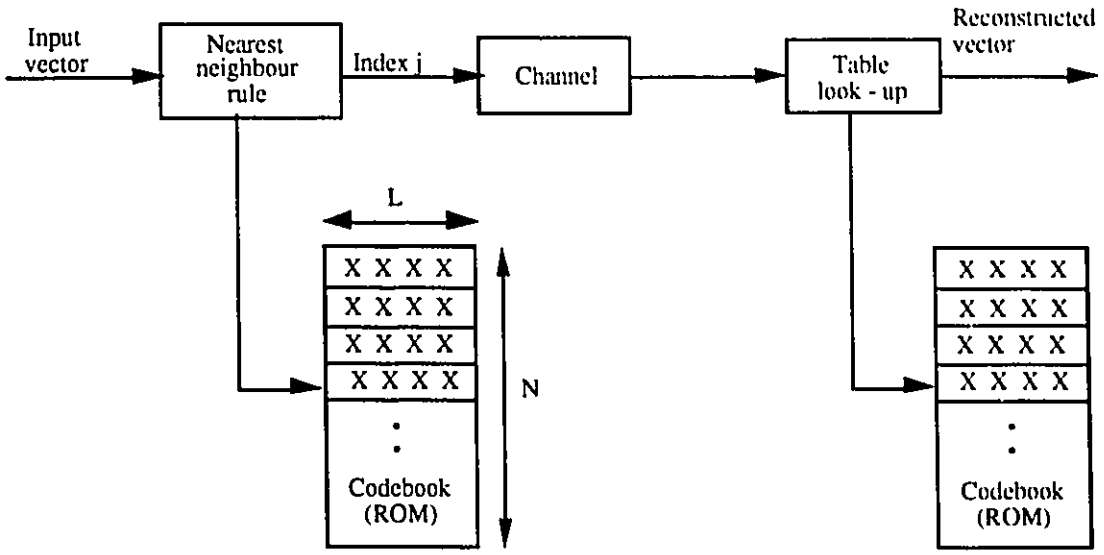


Figure 2.4: Vector quantization.

measure. For example, if the squared error measure is used,

$$d(V_i, W_j) = \frac{1}{L}\sum_{k=1}^{L}(v_{ik} - w_{jk})^2 \tag{2.17}$$

the elemental distortion operation is the computation of the difference, and squaring between an element of V_i and the corresponding element of W_j.

Since the input vector is compared to all the codewords, this type of VQ is referred to as a full search VQ. The encoding complexity of a full search VQ is $O(KLN)$ and the storage cost is $O(LN)$ assuming one storage location per vector dimension. Thus, both the computational complexity and storage cost grow linearly with the number of codewords and the vector dimension.

We recall that codebook generation involves two processes: a classification process in which each training vector is assigned to its nearest neighbour and a centroid computation process. Most of the computation is utilized for the classification process. The codebook generation complexity and storage cost for an N-level codebook are $O(K_vLNI)$ and $O(L(N + K_v))$, respectively, where K_v is the number of training vectors and I is the number of iterations required.

Vector Quantization using an Universal Codebook

Vector quantization techniques can be broadly classified, with respect to training and codebook generation, as universal and adaptive. In this section, image coding using universal VQ is reviewed, while adaptive VQ is discussed in the next section.

Universal VQ [29, 55] employs a fixed codebook generated using a large set of training vectors selected from different types of images. To ensure a good image fidelity the codebook must be large, which in turn increases both the bit rate and the coding complexity. In addition, if the input images/video frames have different statistics, good coding performance may not be achieved for a variety of applications. The codebook size can be reduced using techniques which exploit the local image statistics. Examples include, classified VQ, predictive VQ, finite state VQ, multi-stage VQ, tree structured VQ, and address VQ.

Classified VQ (CVQ): In CVQ [32, 33], the training vectors are classified into a finite set of classes based upon some perceptually important features, such as, the edge content and a separate codebook is generated for each class. In the quantization process, a classifier determines the class of the input vector, which is then encoded using the appropriate codebook. The CVQ technique has been extended to include different codebooks within a class and different coding methods for each class. For example, a CVQ in the discrete cosine transform domain (DCT-CVQ) has been reported [34]-[36] where the input blocks are first transformed using DCT and assigned to a class. The DCT blocks belonging to a class are then partitioned into several vectors. The subvectors are quantized separately using an appropriate codebook.

Predictive VQ (PVQ): Memoryless VQ can efficiently exploit the intra-vector correlation, but can't utilize the inter-vector redundancy. PVQ schemes belong to a class of VQ with memory and can exploit the inter-vector correlation. The PVQ coding system employs a predictor followed by vector quantizer [30, 31]. The input vector is predicted from the previously transmitted vectors. The error vector is formed by subtracting the predicted vector from the current input vector which is then vector quantized.

Finite State VQ (FSVQ): Another VQ approach with memory which reduces the

bit rate is the finite state VQ (FSVQ) [38, 39]. A FSVQ encoder consists of a state transition function f and a set of finite states. A small subcodebook is associated with each state. The function f uses the previously encoded vectors to determine the current state of the encoder. The input vector is quantized using the subcodebook corresponding to the current state. For example, Kim [39] has proposed a FSVQ technique where the state transition function f is based upon the gray level transition across the boundaries of the neighbouring blocks.

Address VQ (AVQ): The address codebook consists of a set of address-codevectors [53] as shown in Figure 2.5. Each address-codevector constitutes the addresses (labels) corresponding to the four codewords in the LBG-codebook. In the encoding process, the four neighbouring blocks are coded using the LBG-codebook to form the address-codevector. The address codebooks at both the transmitter and the receiver are re-organized, using the block transition probability matrices, in order to maintain the most probable address-codevectors in the active address-codebook. The active address-codebook is then searched for a matched address-codevector. If a match is obtained, the index corresponding to the matched address-codevector is transmitted. Otherwise, four labels are transmitted independently to the receiver.

An extension of AVQ is the multilayer address VQ (multilayer AVQ) [54], where the input image is encoded at the first layer using the LBG-codebook. The resulting labels from the first layer are merged together in the second layer to form a new address-codevector. This process of merging is repeated recursively in a hierarchical bottom up fashion. At each layer the address-codebook consists of the most probable address combinations. Each address-codevector is encoded starting at the top layer, using the corresponding address-codebook similar to AVQ.

Multi-Stage VQ (MSVQ): Figure 2.6 shows the structure of an n-stage vector quantizer [47]. The input vector V_i is first vector quantized using the N_1-level quantizer and the error vector is formed. The error vector is then quantized using the N_2-level quantizer. The process is repeated by finding the error vector from the second stage and feeding it to the third stage and so on. The final quantized version of V_i is the sum of all repro-

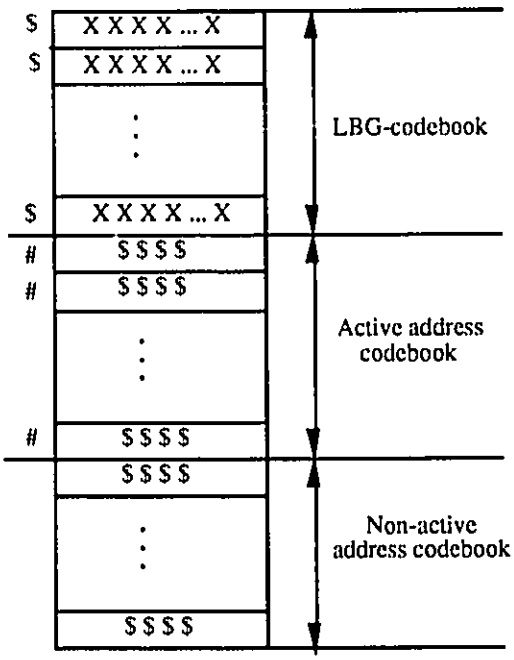


Figure 2.5: Address codebook.

duction vectors. The computational and storage costs are $O(KL\sum N_i)$ and $O(L\sum N_i)$, respectively.

Several variations of MSVQ have been reported in the literature. For example, Wang *et al* [48] have proposed a MSVQ technique applied to the transform coefficients with optimal bit allocation strategy. Koessentini *et al* [49] have presented a variable rate MSVQ technique. Chan *et al* [50] have proposed a joint design algorithm of the stage codebook to optimize the performance of MSVQ.

Tree structured VQ (TSVQ): The computations necessary for finding the matched codeword can be reduced by structuring the codebook as a binary tree [44]. The tree is constructed with a codeword at each node, such that the first layer of the tree divides the $L-$dimensional space into two regions, the second layer divides the $L-$dimensional space into four regions, and so on. Only the codewords at the leaves are used for encoding. In quantizing each vector, a decision is made between one of the two branches at each layer, and the matched codeword is found at the last level. In comparison with the full search

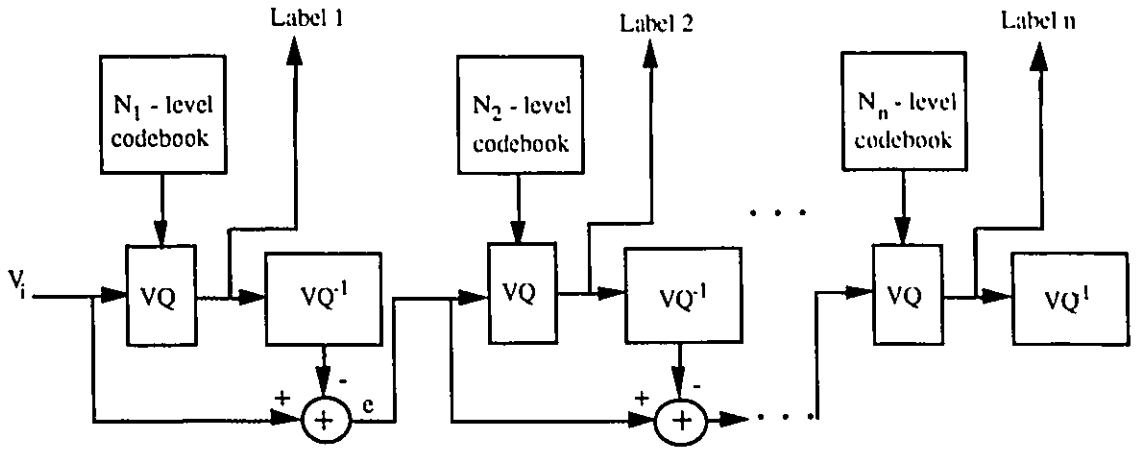


Figure 2.6: Multi-stage vector quantizer.

VQ, the computational complexity is reduced to $O(KL\log_2 N)$ at the expense of doubling the storage cost.

The K-dimensional tree (K-d) is a generalization of the binary tree and it provides a data structure which allows for multi-dimensional search. Recently, Equitz [28] and Ramasubramanian *et al* [46] have proposed fast search algorithms for VQ encoding using K-d tree structure.

The performance of a TSVQ will in general have some degradations compared to a full search VQ. To improve the coding performance of TSVQ, Riskin *et al* [45] have presented an unbalanced TSVQ structure where the tree is designed one node at a time rather than a layer at a time.

Vector Quantization using an Adaptive Codebook

In adaptive VQ, the codebook is adapted in order to match the local image statistics. For example, a new codebook can be generated using the vectors of the input image as a training set. The new codebook is transmitted, followed by the labels corresponding to the vectors of the image. Goldberg *et al.* [41] have presented an adaptive VQ scheme for coding monochrome and color images. In their scheme, the image is partitioned into non-overlapping subimages, and for each subimage a separate 16 - 64 level codebook is created.

These codebooks better match the local image statistics, however, the improvement in coding performance is achieved at the expense of the overhead incurred for transmitting the new codebooks. Zeger *et al* [43] have proposed a technique where a new codebook is designed for each input image. The new codebook is vector quantized using a large universal codebook; hence, reducing the overhead for transmitting the new codebook. An alternative scheme, which reduces the overhead, is to update or replenish part of the codebook. For example, Gersho *et al* [40] have proposed a technique where the distortion of each input vector is monitored, and if it is larger than a predetermined threshold, the codeword with the "largest time since use" is replaced by the input vector. Yeh [51] introduced a technique where the codewords that causes excess quantization error is replaced. We note that in the above techniques, the major drawback is that the improvement in image quality is achieved at the expense of increasing the computational complexity.

Panchanathan *et al* [42] have presented a reduced codebook design algorithm for adaptive VQ. Here, a large universal codebook of size $\bar{N}$ is employed to encode the input vectors. A reduced codebook of size N consisting of only the used codewords in the universal codebook is then formed. Since $N \ll \bar{N}$, this technique results in lower bit rate for label encoding. In another variation of this scheme, the input vectors are used as a training set and one iteration of the LBG algorithm, using the reduced codebook as the initial codebook, is performed. The changes to the codewords are then transmitted. Panchanathan *et al* [52] have also presented a dynamic VQ scheme where the codebook is generated on the fly from the input vectors to be coded. In their scheme, a small primary codebook is used to store the most frequently used codewords, and a larger secondary codebook is used to store the less frequently used codewords. Both the transmitter and the receiver keep track of identical codebooks without any overhead. This scheme is adopted as a part of the VC-FAVQ algorithm presented in chapter 4.

Bit Rate

In the universal VQ, only the labels are transmitted. Hence, for a codebook of size N, the bit rate in bits/pixel is:

$$R = (\log_2 N)/L \tag{2.18}$$

where L is the vector dimension. In the case of adaptive VQ, the codebook and the labels are transmitted. Hence, the total bit rate for an $M_1 \times M_2$ image in bits/pixel is given by:

$$R = \lfloor (\log_2 N)/L \rfloor + \lfloor (N_c \times B)/K \rfloor \tag{2.19}$$

where N_c is the number of transmitted codewords, B is the number of bits per vector, and K is the number of vectors in the image.

2.3 Video Compression Techniques

We recall from section 2.1 that compression of the video sequence is achieved by removing the spatial and temporal correlations using intraframe and interframe coding techniques, respectively. In this section, a review of video compression techniques is presented.

2.3.1 Motion Estimation/Compensation

Motion estimation/compensation is a widely used technique to exploit the temporal correlation in a video signal. Motion estimation techniques attempt to obtain the motion information for the various objects in the scene. In principle, if the motion information of the object is known, then high compression could be achieved by coding the first frame together with the motion information. There are two classes of motion estimation/compensation algorithms: pel-recursive [12] and block matching (BMAs) [3]. Pel-recursive algorithms evaluate the displacement of each pixel individually. These algorithms do not require the transmission of motion information but recursively use the luminance change to find the motion information. On the other hand block matching algorithms assume that all pixels within a block have the same motion. The motion vectors for a group of pixels are calculated and transmitted to the receiver.

A popular method for block matching is the full search block matching algorithm (FBMA). The FBMA searches all possible displaced locations within a search area to find the best match. Several matching criteria have been proposed to find the best match block [102]. These include: the mean squared error, the mean of absolute error, etc.The mean absolute difference error (MAD) criterion is preferred because of its lower computational complexity. The MAD is given by:

$$MAD_{(k,l)}(X,Y) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} |X_{i,j} - Y_{i+k,j+l}| \qquad -p \leq k,l \leq p \tag{2.20}$$

where $X_{i,j}$ is the reference block data at coordinates (i,j) and $Y_{i+k,j+l}$ is the candidate block data at coordinates $(i+k, j+l)$.

Figure 2.7 illustrates the block matching process between adjacent frames. The current frame (F) is divided into non-overlapping reference blocks of size $n \times n$. Each reference block in (F), is compared with the candidate blocks within a search area (SA) in the previous frame $(F-1)$ in order to obtain the best match. The size of SA is $(n+2p)^2$ where p is the maximum displacement, and is centered around the candidate block with identical coordinates as the reference block. The offset between the reference block and the best match candidate block specifies the displacement (motion) vector.

For each of the $(2p+1)^2$ locations to be tested, the calculation of the MAD requires $3n^2$ operations (subtraction, absolute value and addition). Hence for K input blocks the computational complexity of FBMA is $O(3Kn^2(2p+1)^2)$. For example, a 512 × 512 image with a block size of 4 × 4 and a maximum displacement of 4 pixels requires approximately 63 million operations. The corresponding values for a displacement of 6 and 8 pixels are 132 and 225 million operations, respectively. In order to reduce the computational complexity, a number of fast search algorithms such as the 2-dimensional logarithmic search, the three step hierarchical search [3, 100] have been reported. These algorithms, reduce the number of computations by limiting the number of candidate blocks. However, for hardware realization, the FBMA offers the advantage of low control overhead since it has regular data flow and fixed number of operation steps. We note that motion estimation techniques are typically combined with image compression techniques

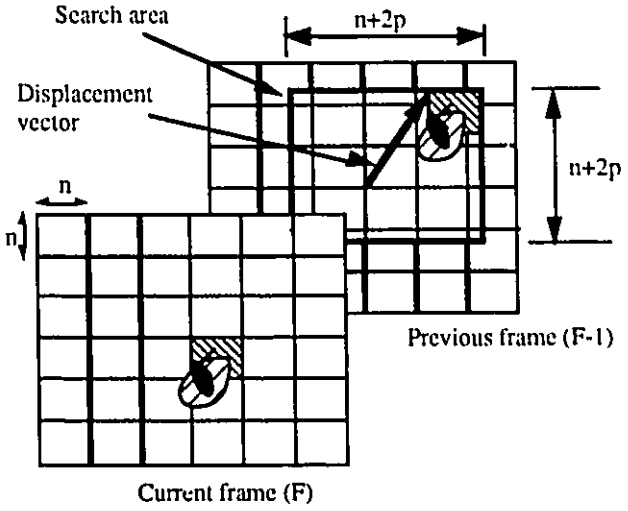


Figure 2.7: Block matching process.

to remove the spatial correlations.

2.3.2 Predictive Coding with Motion Compensation

Motion compensated predictive coders are a class of adaptive predictive coding systems that take into account the motion of the objects in the scene in order to reduce the prediction error. For example, if an object in a moving scene has a spatial translation of $(\Delta i, \Delta j)$, then one expects that a pixel $x_{i,j,F}$ (belonging to the moving object) is maximally correlated with the pixel $\hat{x}_{i+\Delta i,j+\Delta j,F-1}$ rather than the pixel $\hat{x}_{i,j,F-1}$. A motion compensated adaptive coding scheme is shown in Figure 2.8. The motion vector $(\Delta i, \Delta j)$ of each pixel $x_{i,j,F}$ is first estimated using a matching procedure, as described in section 2.3.1, which determines the displacement of each pixel in a way such that the difference between the current pixel and the predicted pixel is minimized. The prediction error $x_{i,j,F} - \hat{x}_{i+\Delta,j+\Delta,F-1}$ and the displacement vector are then coded separately and transmitted to the receiver.

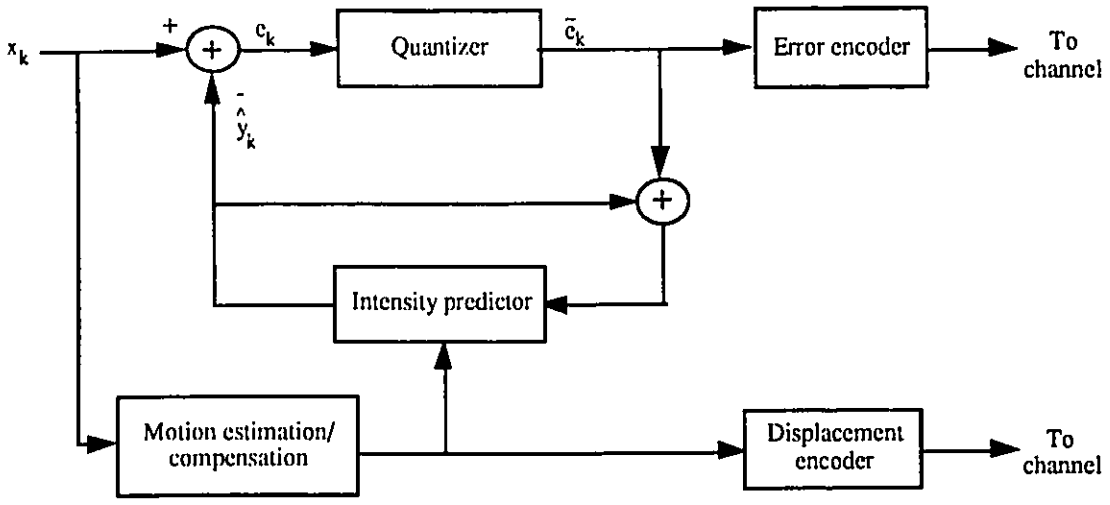


Figure 2.8: Motion compensated DPCM coder.

2.3.3 Conditional Replenishment

Conditional replenishment technique [4] is based on dividing each frame into stationary and non-stationary parts. Only the changed parts are coded. If $x_{i,j,F}$ is the pixel at location (i,j) in the current frame (F), then the predicted value $x_{i,j,F}$ is the reconstructed value $\hat{x}_{i,j,F-1}$ at the same spatial location in the previous frame $(F-1)$. The prediction error in this case is calculated using:

$$e_{i,j,F} = x_{i,j,F} - \hat{x}_{i,j,F-1} \tag{2.21}$$

If the magnitude of $e_{i,j,F}$ is greater than a prespecified threshold, then it is quantized, coded, and transmitted along with the location (i,j) of $x_{i,j,F}$.

A number of algorithms based on the basic conditional replenishment technique have been reported in the literature. For example, instead of coding the address of each changed pixel separately, the addresses and the quantized errors are coded in clusters along a line [13]. Another variation is to change the temporal resolution in stationary parts and the spatial resolution in the moving parts in order to obtain further reductions in bit rate without affecting the subjective quality.

2.3.4 Interframe Transform Coding

Interframe transform coding exploits the temporal correlations between successive frames as well as the spatial correlations within the individual frames. A sequence of frames is partitioned into three dimensional blocks $X_{m_1,m_2,t}$ of $M_1 \times M_2 \times T$ pixels. Each block then undergoes a three dimensional transform according to the following general formula:

$$Q_{u,v,w} = \sum_{m_1=0}^{M_1} \sum_{m_2=0}^{M_2} \sum_{t=0}^{T} X_{m_1,m_2,t} A_{u,v,w;m_1,m_2,t} \tag{2.22}$$

to generate the transform coefficients $Q_{u,v,w}$, where $A_{u,v,w;m_1,m_2,t}$ is the transform kernel and M_1, M_2, and T denote the row, column, and temporal indices, respectively. The transform coefficients are then quantized, coded and transmitted to the receiver. To reconstruct the encoded image sequence, the receiver applies an inverse three dimensional transform on the quantized coefficients $\hat{Q}_{u,v,w}$:

$$\hat{X}_{m_1,m_2,t} = \sum_{u=0}^{M1} \sum_{v=0}^{M_2} \sum_{t=0}^{T} \tilde{Q}_{u,v,w} A^{-1}_{u,v,w;m_1,m_2,t} \tag{2.23}$$

where $A^{-1}_{u,v,w;m_1,m_2,t}$ is the inverse three dimensional transform kernel.

In image/video sequences the statistics along the temporal dimension may vary significantly, therefore adaptive techniques can substantially improve the coding performance [15]. For example, the effective number of bits assigned to each coefficient can be made proportional to the coefficient variance [7].

2.3.5 Hybrid Interframe Transform/Predictive Image Coding

Interframe hybrid coding [2] combines transform and interframe predictive coding.Typically, each frame is partitioned into two dimensional blocks, a transform is executed on the resulting blocks, the coefficients are then predicted from the quantized coefficients along the temporal direction.

2.3.6 Interframe VQ

Several image/video sequence coding algorithms based on VQ have been reported in the literature. The first class of algorithms uses a fixed VQ codebook. Murakami *et al* [58]

have presented a vector quantizer for video signals where a fixed codebook is generated using a long training sequence of normalized (by average and standard deviation) vectors. The mean and standard deviation are transmitted as side information along with the label. Corte-real *et al* [59] have proposed an image sequence vector quantization scheme where the differential image is divided into blocks of different shapes and sizes. Several vector quantizers are then used to encode the resulting blocks. Huguet *et al* [60] have extended the concept of 2-dimensional VQ to image sequence coding (3-dimensional VQ). In their technique, the image sequence is first divided into 3-dimensional blocks. The blocks are then rearranged into vectors and quantized using a previously trained codebook. Guo *et al* [62] have proposed a technique in which the difference between the input frame and the predicted frame is vector quantized. The difference frame is divided into 16–dimensional vectors. For each difference vector, directional conditional vector probability matrices are used to select a sub-codebook from a larger codebook. The input vector is then encoded using the sub-codebook or the larger codebook based on which results in lower distortion. Nasrabadi *et al* [55, 63, 64] have proposed an interframe hierarchical vector quantizer using quadtree decomposition (IHA-VQ), where each frame is partitioned into 7×7 blocks. Block matching [3] is used to estimate the motion of each block. A bit map to indicate the status (motion/no motion) of the blocks followed by the motion vectors of the moving blocks is transmitted. The difference between the current frame and the motion estimated frame is then segmented hierarchically into $32 \times 32, 16 \times 16, 8 \times 8, 4 \times 4$, and 2×2 blocks. The 32×32, 16×16, and 8×8 blocks are replenished from the previous frame whereas a set of codebooks is used to encode the smaller blocks. There are two problems with this class of algorithms. First, a limited size codebook is not sufficient to represent the different types of image sequences. Therefore, a large codebook is required which increases the bit rate and entails practical difficulties in codebook generation and encoding processes. Secondly, the mismatch between the codebook and image sequences outside the training set may degrade the coding performance.

The second class of algorithms is based on updating the codebook during the encoding process so that it reflects the the local statistics of the frame to be encoded. Goldberg

et al [65, 66] have presented several interframe coding techniques where the changes in successive frames are tracked by incorporating label replenishment and codebook replenishment. Here, the vectors of the first frame are used to generate the codebook. The frame is vector quantized, and the corresponding labels are stored in the label memory. The vectors in the subsequent frames are quantized, and the indices are compared with the corresponding labels in the label memory. If they differ, the label memory is replenished. In order to adapt the codebook to the local frame statistics, codebook replenishment is used where the mean of the new clusters is determined using the vectors of the current frame. The new means are then compared with the corresponding codewords. If the difference is larger than a prespecified threshold, the codeword is modified. Another approach to replenish the codebook is to design a new codebook for each frame using the vectors of the current frames as a training set and the previous codebook as the initial codebook [66]. The resulting centroids are then compared to the codewords in the initial codebook. If the difference is greater than a threshold, the codeword is replenished. Monet *et al* [67] have proposed the application of codebook replenishment technique for classified pruned tree structured VQ. This technique uses the average distortion resulting from the assignment of a set of input vectors to a particular codeword as a measure to determine whether to delete, split or modify the codeword. Yeh [69] has presented an adaptive VQ technique for image sequence coding. In this technique, a mean/residual binary tree VQ is employed where the quantization error is used to determine whether to replenish the codeword or not. However, in all these algorithms frame adaptibility results in further increases in computational complexity making real time implementation difficult.

2.4 Video Compression Standards

Recently, the International Standards Organization (ISO) have proposed standards for still image and video compression known as the JPEG (Joint Photographic Experts Group) [72, 73] and the MPEG (Moving Picture Experts Group) [75, 76]. In addition,

the Consultative Committee on International Telephony and Telegraphy (CCITT) has recommended a standard for videotelephony called the H.261 [74] at $p \times 64$ Kbits/s. In this section we review the MPEG video compression algorithm as it is used as a baseline for comparison.

In the MPEG video compression standard a block based motion estimation/ compensation is employed to remove the interframe correlation and discrete cosine transform (DCT) for the removal of the intraframe correlation. Block diagrams of the MPEG encoder and decoder are shown in Figure 2.9 and 2.10. Here, a group of pictures approach is used instead of the frame by frame coding [75]. A group of pictures is typically a combination of one or two intra-pictures (I), predicted picture (P), and the rest of bidirectional pictures (B) as shown in Figure 2.11. The I frames provide random access points and are also used as a reference for P frames. The I frames are coded using DCT on 8×8 blocks. The DC coefficient is differentially encoded using variable length codes (VLC). The AC coefficients are zig-zag scanned as shown in Figure 2.12 and ordered into {RUNLENGTH, AMPLITUDE} pairs. A variable length coder is used to encode each pair.

The P and B frames are decomposed into 16×16 blocks and the motion vector for each block is calculated. The motion compensated difference frame is partitioned into 8×8 blocks which then undergo a 2-dimensional DCT. The DC and AC coefficients are quantized, ordered along the zig-zag scan line into {RUNLENGTH, AMPLITUDE} pairs and coded using a VLC. The motion vectors are also variable length coded and transmitted.

2.5 A Comparison of Image Coding schemes

In the previous sections, predictive, transform, and hybrid transform/predictive coding have been reviewed. Each coding scheme possess relative advantages and disadvantages[2, 4]. In terms of coding performance, transform coding relatively outperforms predictive coding. At low bit rate (less than 1 bits/pixel), transform coding achieves a higher image quality, while predictive coding results in high fidelity images at high bit rates (around 2

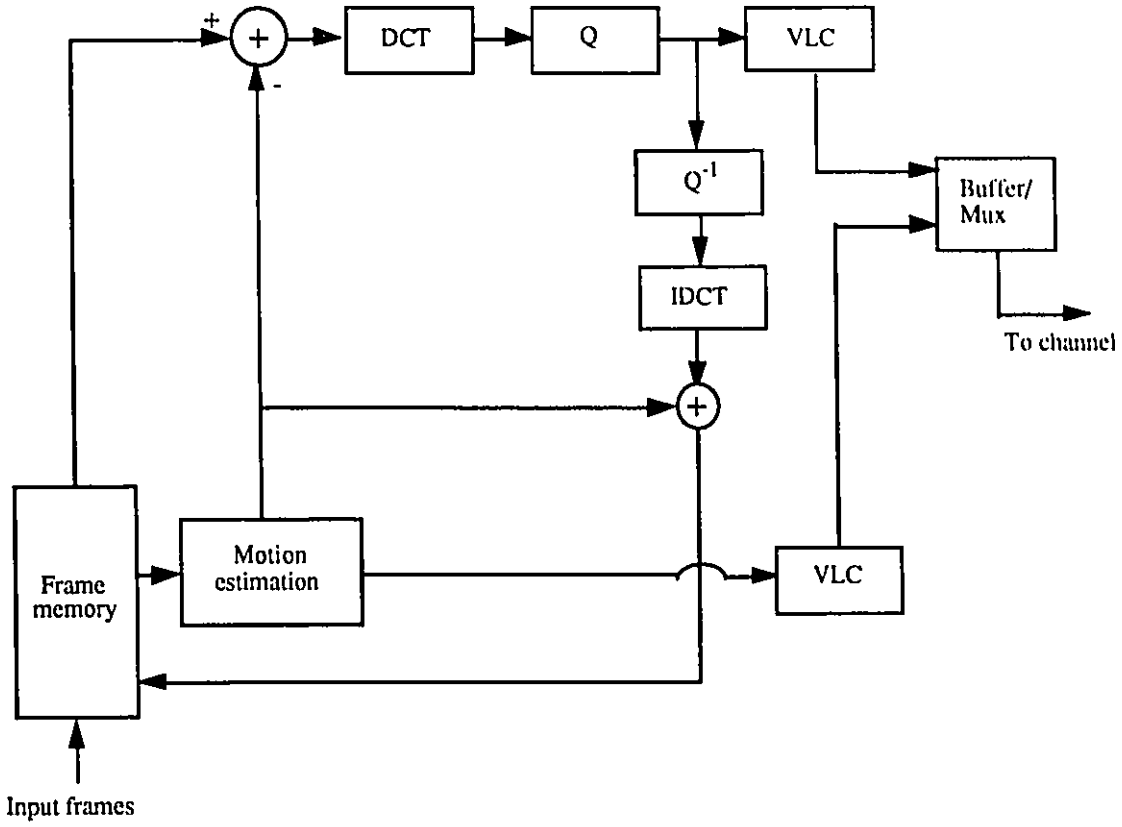


Figure 2.9: MPEG encoder block diagram.

bits/pixel). Distortions due to quantization and channel errors in the transform coding are distributed over the entire image whereas predictive coding is much more sensitive to such errors(distortions are locally distributed). However, transform coding has a much higher complexity both in terms of computation and memory requirements. Hybrid coding combines the advantages of the high performance of transform coding and the simplicity of predictive coding. Its coding performance is in between transform and the predictive coding. Good quality images are reported at moderate bit rates (1 to 2 bits/pixel). The sensitivity of hybrid coding to distortions is less than that of DPCM, but it is not as robust as transform coding. VQ is efficient at very low bit rates (less than 0.5 bits/pixel). However, VQ may not be as robust as the previous coding methods in the presence of channel errors. The complexity of the encoding process of VQ is comparable to that of the

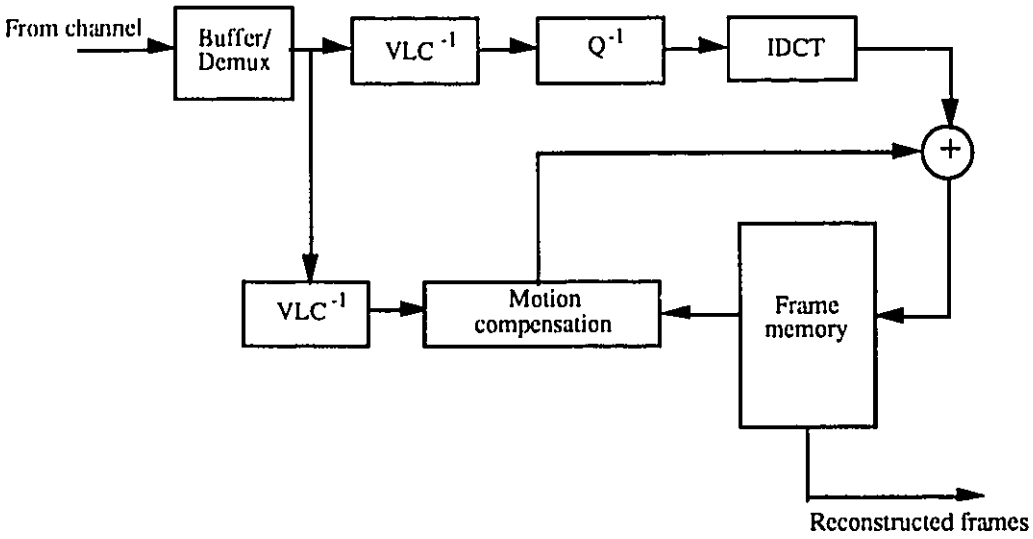


Figure 2.10: MPEG decoder block diagram.

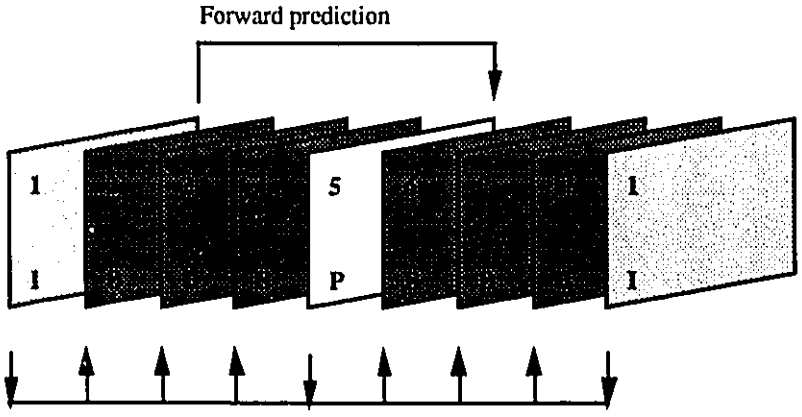


Figure 2.11: Example of a group of pictures used in the MPEG.

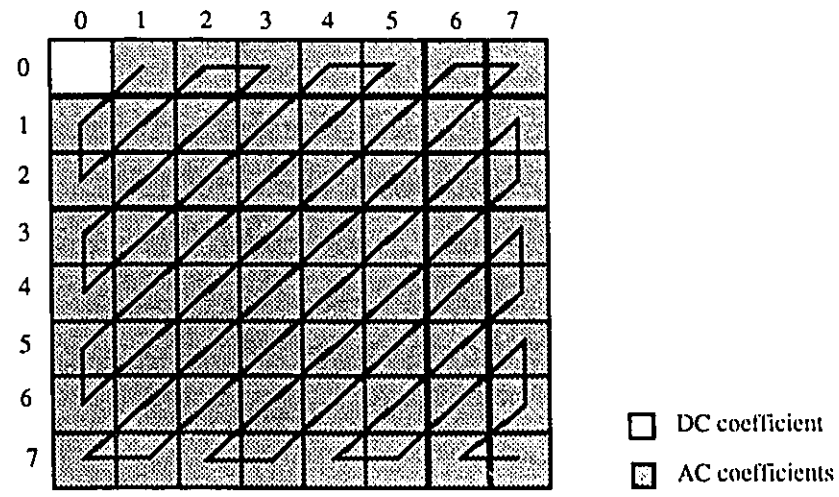


Figure 2.12: The zig-zag scan path of the AC coefficients used in encoding the I frames.

transform coding. In comparison to all other techniques, the VQ decoder is the simplest, as it can be implemented using table look up techniques.

2.6 Summary

In this chapter, we first discussed the concepts of lossless and lossy image coding techniques. This was followed by a review of predictive coding, transform coding, and hybrid transform/predictive coding schemes. A review of vector quantization as applied to image coding was then presented. Video compression techniques including motion estimation/compensation, 3-dimensional transform coding, and interframe VQ schemes were then reviewed. Finally, a brief review of the standards for video compression followed by a comparison between the different coding techniques was presented.

Chapter 3

Review of Architectures for Vector Quantization and Motion Estimation

Image processing is characterized by high processing and data throughput requirements which cannot be easily met by the existing general purpose computers. These requirements coupled with the advent of VLSI technology have led to several application specific architectural designs.

In this chapter the need for architectures is first established in section 3.1. This is followed by the classification of architectures in section 3.2. The architectures for VQ and motion estimation/compensation are reviewed in sections 3.3 and 3.4, respectively. Finally, a summary is presented in section 3.5.

3.1 The Need for Architectures

Image processing algorithms can be classified according to the complexity into low, medium, and high level algorithms. Low level algorithms are simple and are executed on the individual pixels of an image/subimage. The operations are carried out in a predefined sequence, and only neighbouring pixels are involved. Filtering and scaling are typical

examples of this class of algorithms. Medium level algorithms, such as edge detection algorithms, generate features and other characteristics of the image scene. The operations are data dependent and involve more complex data structures. High level algorithms, operate on the output of the low and medium levels for analysis and interpretation. For example, pattern recognition algorithms belong to this class. These algorithms not only exhibit the properties of the medium level algorithms, but also involve search and list processing operations.

The problem in implementing the above algorithms arises from the need to process large amounts of data in real time. Consider, for example, the processing of a television sequence at 30 frames per second. The amount of data that must be processed per second is 180 million bits, assuming a spatial resolution of 512×512 pixels per frame with 3 bytes per pixel (3 colors, 8 bits/color). The number of operations per pixel ranges from hundreds (for low level processing) to many thousands (for high level processing), which implies a computation rate in the range of 2 - 22 billion operations per second.

In order to handle the high computational overload, different types of computer architectures have been applied in image processing applications. However, no single architecture can solve all possible problems. For example, array processors are well suited for low level algorithms, however, they are not efficient for high level algorithms because of their limited control. Multiprocessor systems are best fit for high level algorithms, but they require expensive support for input/output, programming, and interprocessor communications. In general, it has been realized that application specific architectures, onto which the algorithms are mapped, are best suited for image processing applications.

3.2 Classifications of Architectures

The most efficient methods for speeding up image processing operations (or any computational intensive task) are: the use of faster technology and parallelism. The use of more advanced, faster technology can provide certain gains; however, there are strict limits to what is achievable [79]. By employing P processors (rather than one) and assuming that

these processors operate concurrently, a speed up factor of P is achievable which seems to be the most effective approach to meet the massive computation requirements. Parallel architectures can be classified based on the type of parallelism found in the system [105, 106]. Flynn [105] categorizes computers, on the basis of the instruction and data flow, into four organizations:

- **Single Instruction Stream Single Data Stream (SISD)** is the conventional Von Neumann computer, where instructions are executed sequentially by a single processor, under the control of a single control unit.
- **Single Instruction Stream Multiple Data Stream (SIMD)** consists of several processing elements (PEs) working with the same control unit (CU). The individual PEs execute the same instruction broadcasted by the control unit.
- **Multiple Instruction Stream Single Data Stream (MISD)** consists of a set of PEs where each PE executes a different instruction on the input data stream and the output of one PE becomes the input of the next PE.
- **Multiple Instruction Stream Multiple Data Stream (MIMD)** is characterized by multiple PEs executing different instructions on their data streams.

Flynn's classification is widely accepted although there are some architectures which cannot always fit within Flynn's taxonomy. In this thesis, a more general classification outlined by Hwang [106] is adopted. By definition, a parallel architecture is a system in which there is a concurrent execution of instructions on the different PEs. Concurrency implies parallelism, simultaneity, and pipelining [106]. Based on this principle, the parallel architectures found in the literature can be classified into one of the following three architectural configurations: Array Processors, Pipeline Processors, and Multiprocessor Systems. The details are presented in the following subsections.

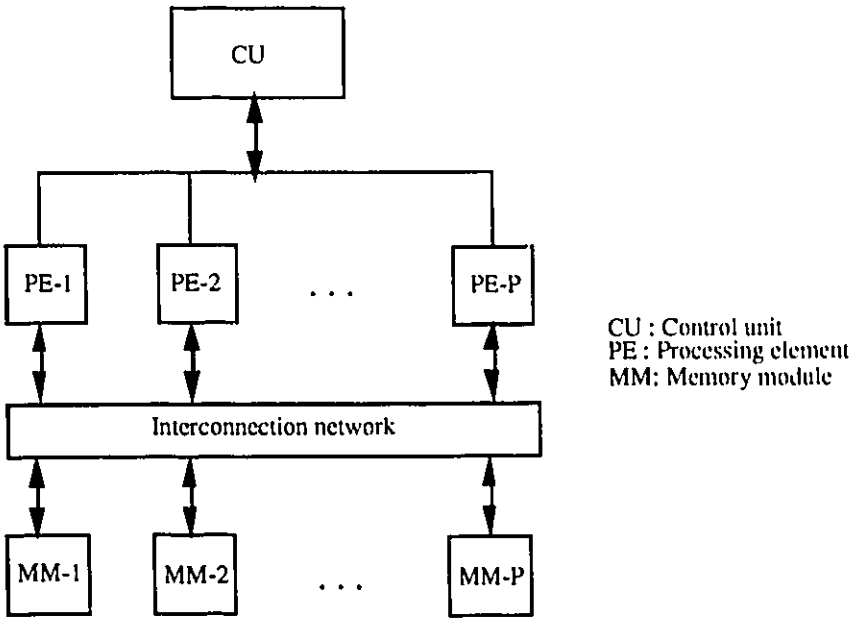


Figure 3.1: Array processor architecture.

3.2.1 Array Processors

Array processors consist of a set of identical synchronized PEs such that they execute the same instruction but on different data. A single control unit broadcasts the instructions to all PEs while the data is placed in the memory units associated with each PE as shown in Fig. 3.1. An interconnection network is used to provide the necessary interprocessor communications. Image data is mapped directly onto the array processor; and hence maximum parallelism is obtainable. However, this architecture has several limitations. Although it is effective for low level algorithms involving individual pixels or neighbourhood operations (such as convolution and correlation), it is inefficient for operations that involve spatially distant pixels requiring interprocessor communications over large distances. In addition, the input/output bandwidth is limited and the processing is delayed until the whole image is loaded into the memory, and the results are not available until the whole image is processed.

Associative Array Processor

An associative array processor employs an associative or content addressable memory in which the stored data items are accessed by their contents or partial contents. The content addressable capabilities of this processor has made it attractive for a wide range of applications as it allows for parallel search and comparison operations.

Fig. 3.2 shows a block diagram of an associative array processor. The main component is a memory unit consisting of $n_1 \times n_2$ identical memory cells $\{B_{ij} : 1 \leq i \leq n_1, 1 \leq j \leq n_2\}$ for storing $n_1 n_2$ bit words. Each memory cell B_{ij} is a flip-flop with associated logic gates for matching and read/write control. The input register I and the mask register M serve to specify a search argument. $M_j = 1$ means I_j is unmasked while $M_j = 0$ means I_j is masked as " don't care" condition, where M_j and I_j are the jth bit of registers M and I, respectively. Register S is used to select the set of words to be involved in searches. $S_i = 1$ denotes that the ith word is in the set. Whenever there is a match in all the bit positions B_{ij}, then R_i, the ith bit of the response register R, is set to 1 indicating that word i satisfies the search condition. In the event of multiple matches, the multiple response resolver will select the appropriate word according to some priority.

Pyramid Processors

A pyramid is configured as a stack of array processors. Each array processor is typically one fourth as large as the array below it. Except for the base array, each PE is connected to the four PEs in the layer directly below it, and to the neighbouring PEs in the same layer. Pyramids have good global characteristics as they provide fast communication in $O(log\ m)$, where m is the edge size of the pyramid base. Pyramids also have good local characteristics for several important classes of problems, since the near-neighbour linkage of their array processor structure is appropriate for handling the near-neighbour interactions in image processing. These characteristics can significantly improve the execution of low level algorithms, especially those requiring communications between spatially distant PEs.

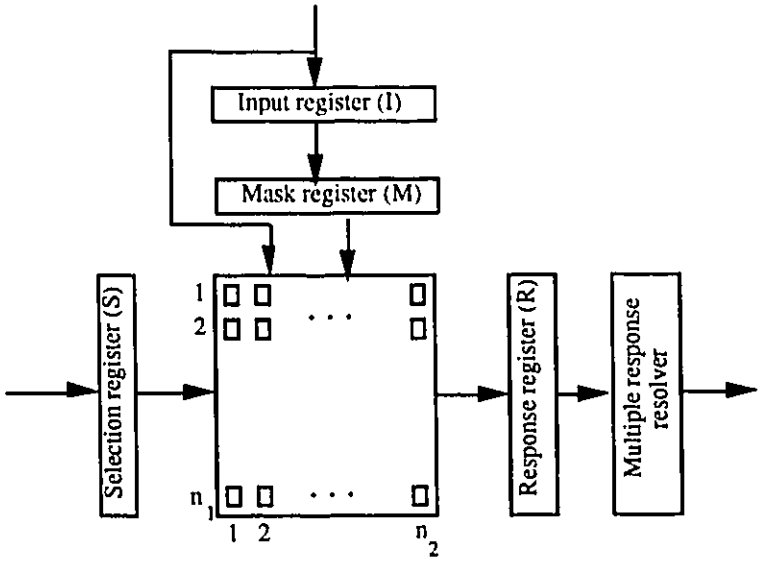


Figure 3.2: Associative array processor architecture.

The effectiveness of pyramid processors as architectures for image processing however, is limited due to the following reasons. First, they are most suitable for low level operations. Second, the number of PEs are very large which limits the capabilities of the PEs. Finally, the input/output bandwidth is limited which in turn reduces the utilization of the processors.

3.2.2 Pipeline Processors

A pipeline processor exploits the temporal parallelism by executing overlapped computations which usually implies, in the context of image processing, concatenation of operators performing arithmetic and logical operations on a small neighbourhood. Although a pipeline processor has the advantage of parallelism, it has a latency of P clock cycles in computing the first result, where P is the number of processors in the pipeline, while subsequent outputs are available at intervals of one clock cycle.

In general, pipeline processors are well suited for applications involving low level processing. However, for complex operations (medium and high level) it is more difficult to use pipelining as it is not possible to map the algorithm directly onto the architecture.

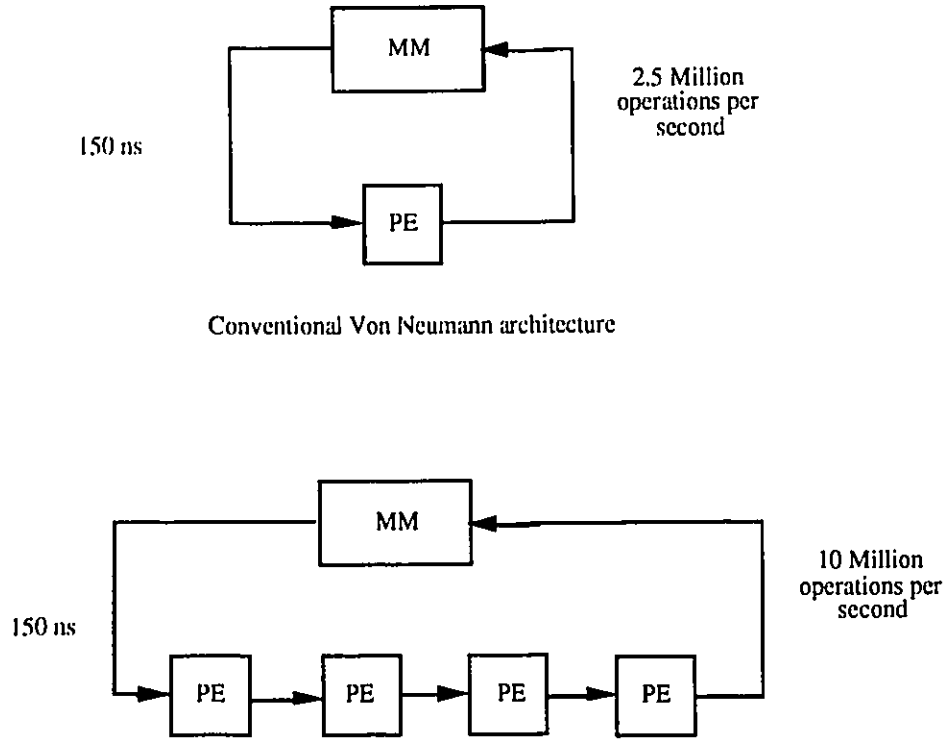


Figure 3.3: Basic principle of a systolic array.

Systolic Arrays

The systolic architectural concept [80, 81] is a general methodology for mapping high level computations onto hardware structures. Systolic arrays consist of locally connected, pipelined processing units where data flows from the system memory in a rhythmic fashion, passing through many PEs before it returns to memory. The basic principle of systolic array approach is illustrated in Fig. 3.3. A higher throughput without increasing the input/output bandwidth is possible if the single PE is replaced by an array of PEs. The main advantage of the systolic arrays is that they provide a balance between the computations and the input/output which is a basic requirement in image processing. Other advantages include high performance, simplicity, regularity, and modularity.

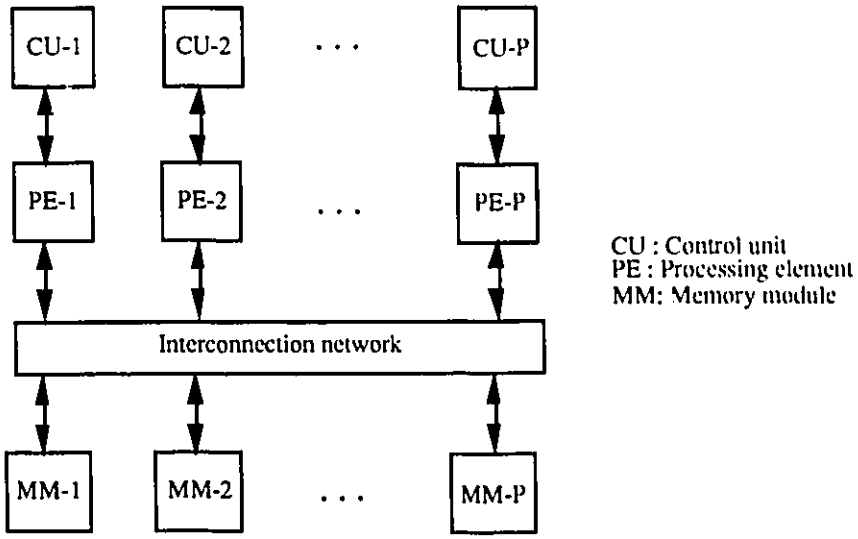


Figure 3.4: Multiprocessor system.

3.2.3 Multiprocessor Architectures

An architecture which partially meets the high level image processing requirements is a multiprocessor [78]. A multiprocessor system consists of a number of PEs, where each PE is supervised by a separate control unit having its own memory as shown in Fig. 3.4. Parallelism is achieved by distributing the processing task over all the PEs. Synchronization between the processors is achieved by passing messages to each other using an interconnection network. In image processing applications, three types of interconnection networks are used: common bus, loop, and shared memory. The capabilities of a multiprocessor PE are much more than those of an array processor. In general, they are a powerful general purpose processors with additional support for input/output and interprocessor communications.

3.3 Review of Architectures for VQ

The intensive computational requirements of VQ has been an impediment for real time implementation in many applications. Recently, several special purpose architectures

which implement VQ in real time for speech and image coding applications have been reported. These architectures can be classified into one of three categories: high speed architectures, modified VQ based architectures, and fast search based architectures.

3.3.1 High Speed Architectures

The first class of architecture is based on faster execution of the distortion measure by using fast processors, pipeline processors, systolic arrays, array processors, and multiple array processors.

Fast Processors

The first approach is to use a fast processor with a high clock rate to execute the elemental operation. This yields a speed up of:

$$S_P = T_{SISD}/T_e \tag{3.1}$$

where T_{SISD} and T_e are the time to compute the elemental operation using an SISD machine and a fast processor, respectively.

Tao *et al* [87] have implemented a full search VQ for speech coding using high speed memories (CMOS), low power Schottky logic (LSTTL) and a table look up for the squaring operation. The elemental operation is executed in 500 ns. However, the speed up achieved is not large enough for image coding applications.

Pipeline Processors

A higher speed up is possible using pipeline processors. Here, each segment of the pipeline executes a portion of the elemental operation. This results in a speed up factor of:

$$S_P = (T_{SISD}/T_e) \times P \tag{3.2}$$

where P is the number of segments in the pipeline.

Davidson *et al* [83] have reported on the design of a high speed codebook search processor (CSP) organized around a semi-custom VLSI pattern matching chip (PMC).

The elemental operation is executed in the PMC using a 4-segment pipeline. The PMC computes the elemental operation in 250 ns. Although this architecture results in a sufficient speed up for real time speech coding, it is not suitable for image coding applications as the throughput rate for an image is three times greater than that of speech.

Systolic Arrays

Another approach to speed up the execution of the elemental operation is to use systolic arrays. Davidson *et al* [84] have presented several systolic architectures for speech and image coding which provide a tradeoff between speed and VLSI chip area. These include: sub-linear array, linear array, superlinear array, and two dimensional array architectures. The number of cycles required for a full search ranges from $O(NL)$ cycles to $O(1)$ cycles.

A higher speed up is possible by exploiting parallelism in the directions of the codebook size N and the vector dimension L. This yields a speed up of:

$$S_P = (T_{SISD}/T_e) \times N \times L \tag{3.3}$$

Panchanathan *et al* [93] have proposed a systolic array for adaptive VQ where the encoding process and the codebook generation are overlapped in the same array. The basic PE operates in two modes, forward and reverse. In the forward mode, the PE executes the elemental operation, while in the reverse mode the PE computes the new centroid. The speed up achieved is not only sufficient for image encoding, but also for sub-optimal codebook generation in real time.

Array Processors

An array processor can be used to compute the elemental operation in parallel in the direction of L. This results in a speed up factor of:

$$S_P = (T_{SISD}/T_e) \times P \times L \tag{3.4}$$

Abut *et al* [86] have reported on a high speed architecture for speech coding where L elemental operations are executed in parallel. Here, each elemental operation is executed in 100 ns using a 3-segment pipeline processor.

Multiple Array Processors

A higher speed up is possible by exploiting parallelism in the direction of N using a multiple array processor. The speed up is given by

$$S_P \quad (T_{SISD}/T_c) \times P \times L \times N \tag{3.5}$$

Lafage *et al* [91] have presented several variations of VLSI architectures which allow tradeoffs between the chip area and the input/output bandwidth. Here, the codebook is partitioned into several on chip codebooks. An array of pipeline processors, associated to each subcodebook, is used to compute the distances between the input vector and the individual codewords in the subcodebooks. A tree comparator then determines the closest codeword. Dezhgosha *et al* [82] have also designed a VLSI architecture for color TV signals where the codebook is partitioned into subcodebooks and a set of pipeline processors are used to compute the elemental distortions.

3.3.2 Modified VQ-based Architectures

The computational complexity of VQ can be reduced by restricting the search space of each input vector to a small number of codewords. For example, in multi-stage VQ, it is possible to use a smaller size codebook at each stage instead of a large sized codebook. Hence, the number of processors can be reduced. Ramamoorthy *et al* [90] have designed a bit serial systolic array using multi-stage VQ for real time image coding. An inner product processor computes the distortion and a comparator and address generator determine the label of the closest codeword. This architecture enables integration of a large number of PEs. However, it has a large chip count and a long start up delay.

3.3.3 Fast Search Based Architectures

This class of architectures is based on the use of fast search methods such as tree structured VQ and by processing multiple input vectors in parallel using associative memories.

Architectures for Tree Structured VQ

Yan *et al* [88] have reported on a bit level systolic array for tree structured VQ. The architecture consists of an inner product processor to execute the elemental operation, and addressable memories to determine the label of the matched codeword. The bit level systolic array can be expanded to more general tree search operations. Abut *et al* [86] have also proposed a systolic array for speech coding using tree structured VQ. Here, the elemental operation is executed in 100 ns.

Dianysian *et al* [85] have presented the design of a programmable array processor for tree structured VQ, where the codebook is partitioned in the direction of L into several codebooks. A pipeline processor associated to each subcodebook computes the partial distortions which are then accumulated to compute the total distortion. The parallelism in the direction of L coupled with the pipelining permits searches in $O(log\ N)$ clock cycles.

Associative Array Processors for Fast Search VQ

Panchanathan *et al* [92] have proposed a content addressable memory (CAM) for adaptive image coding. The input image is stored in the CAM array and the search process is carried out in parallel from the perspective of the codewords, hence exploiting parallelism in the directions of K and L. The speed up factor is given by:

$$S_P = (T_{SISD}/T_e) \times K \times L \tag{3.6}$$

This speed up is sufficient for image coding and codebook generation in real time. Abut *et al* [86] have reported on a fuzzy associative memory which provides an increase in speed up by exploiting parallelism in the direction of L and processing multiple input vectors in parallel.

3.4 Review of Architectures for Motion Estimation

We recall from section 2.3.1 that block matching (BMA) is widely used for motion estimation because of its simplicity. Existing BMAs are full search (FBMA), two-dimensional

logarithmic search, three step hierarchical search, etc. [3, 100]. In contrast to other BMA schemes, the FBMA has a regular data flow, fixed number of operation steps, and a high computational complexity. However, for hardware realization, FBMA is preferred because of its lower control overhead requirements which compensate for the large number of computations per pixel.

Recently, architectures which implement the FBMA have been reported in the literature [94]-[103]. The architectures can be classified into two categories: general purpose multiprocessor systems and systolic arrays.

3.4.1 Multiprocessor Systems for FBMA

Several general purpose processor systems have been proposed to overcome the high computational complexity of FBMA. These systems consist of programmable processors. Data is preloaded into the local memories via the system bus and the matching task is distributed among several processors. For example, Wehberg *et al* [94] have designed a programmable parallel processor structure based on custom designed PEs. Each PE consists of a linear kernel processor which operates at 100 MOPS, a standard signal processor, and two local memories. By setting the necessary parameters, block matching can be executed in real time. May [95] has reported on a system consisting of 8 digital signal processors (DSP) and a control processor for executing general tasks including block matching for videotelephony. Tamitani *et al* [96] have designed a real time video signal processor system which employs three multiprocessor clusters. The individual clusters are composed of: a set of processing modules, two bus switch units, and a variable delay unit. The system operates at 500 MOPS and are used for real time video signal processing including block matching. These implementations have the disadvantages of high cost and sub-optimal mapping of the algorithm onto the general purpose processors [103]. In addition, communication bottlenecks as a result of multiple processors accessing shared data resources [99] limits the over all computational rate.

3.4.2 Systolic Arrays for FBMA

The second class of architectures for real time implementation of FBMA is based on systolic arrays. The various systolic systems, which have been presented, differ in terms of the mapping strategy and the number of processing elements employed. Yang *et al* [98] have presented a one-dimensional semi-systolic array which executes FBMA on 16×16 blocks. Dianysian *et al* [97] have proposed an 8×8 bit-serial systolic array for FBMA. However, the large number of off-chip memory accesses and the complexity of the address generation circuitry increases the input bandwidth and control requirements.

To reduce the complexity of the control circuitry, architectures based on data flow scheme which allow sequential inputs but perform parallel processing have been proposed. For example, Yang *et al* [101] have reported on the design of a programmable architecture to accommodate different block sizes. The design also includes the circuitry for testability. Hsieh *et al* [103] have reported on a systolic array architecture with programmable shift registers. This architecture has serial data inputs and can be adapted to the size of the search area.

The off-chip memory access can be reduced by employing on-chip buffers. For example, Komarek *et al* [99] have proposed several one and two-dimensional systolic arrays with local memories and shift registers to reduce the off-chip memory accesses. The architectures provide a tradeoff between the speed and the number of PEs employed. Vos *et al* [100] have presented a systolic architecture with on-chip registers and line buffers to solve the input bandwidth problem. Chou *et al* [102] have reported on a systolic architecture capable of operating on a search area of size up to 256×256. This flexibility is offered by using on-chip RAM and control registers. Although the computations are efficient in the architectures reported in the literature, they are complex and/or non-cascadable.

3.5 Summary

Image processing applications are characterized by large data which must be processed in real time. Conventional architectures are not suitable for executing image processing

applications and hence there is a need for application specific architectures which are adapted to the algorithm.

Systolic arrays consist of locally connected, pipelined processing units. They provide a balance between the computation rate and the input/output rate. Associative memories have the capability of performing parallel search operations based on the content/partial content of the stored words. This has made it attractive for database and template matching applications.

Vector quantization architectures are essentially intra-image architectures, and can be classified into three categories. The first class of architectures is based on faster execution of the distortion measure. A second class of architectures is based on the restriction of the search space for each input vector to a smaller subcodebook. The third class of architecture is based on the use of fast search methods.

Motion estimation architectures are essentially inter-image architectures. The architectures which implement the full search block matching algorithm are mainly based on systolic arrays. These architectures differ in terms of mapping the algorithm onto the systolic architecture and the number of processing elements employed.

For video compression applications, a combination of the inter- and intra-image architectures are used. This combined architecture may result in inefficiencies in implementation as the common elements between the two algorithms and architectures are not fully exploited. In addition, these designs are complex and/or non-cascadable which results in an increased hardware requirement and reduced flexibility in VLSI implementation.

Chapter 4

Video Compression Using Frame Adaptive Vector Quantization

4.1 Introduction

In this chapter, a new adaptive algorithm for the compression of video sequences using vector quantization is presented. Video sequences are highly nonstationary and generally exhibit variations from frame to frame and from scene to scene; hence using a fixed codebook to encode the different frames/sequences cannot guarantee a good coding performance. We recall from section 2.3.6 that two approaches are usually used to improve the coding performance of VQ. The first is to use a large fixed codebook, while the other is to use a smaller codebook but replenish or modify the codebook so that it matches the statistics of the frame to be coded. However, we note that both approaches result in further increases in computational complexity and/or the bit rate. We propose a new inter-/intraframe adaptive technique for video compression (VC-FAVQ) [110]. This technique exploits the interframe as well as intraframe correlations in order to reduce the bit rate. In addition, a dynamic, self organized codebook is used to track the local statistics from frame to frame. Hence, the proposed technique provides excellent coding performance at a reduced complexity. In addition, VC-FAVQ is a single pass technique which makes possible real-time implementation.

In the following section, the VC-FAVQ technique is presented. The simulation results for a set of video sequences using VC-FAVQ are reported in section 4.3. An analysis of the computational complexity is presented in section 4.4. The summary is then presented in section 4.5.

4.2 Video Compression Using Frame Adaptive VQ

We now describe a new adaptive technique for video compression (VC-FAVQ). In VC-FAVQ two coding modes are employed: interframe and intraframe. the input frame is first partitioned into 4×4 blocks. Each block is then re-organized into a 16-dimensional vector. To start with, the encoder attempts to code the input vector in the interframe mode, however, if it fails, the input vector is encoded in the intraframe mode.

Interframe Mode: A block diagram of the interframe mode is shown in Figure 4.1. Here, each input vector V^F in the current frame (F) is compared with the vector V^{F-1} at the same spatial location in the previous frame ($F-1$). If they match within a prespecified threshold Δ, a flag S_f is transmitted to the receiver. Otherwise, a search area SA in $(F-1)$ is defined and a match within SA is sought. The size of SA is determined from the displacements of the neighbouring vectors. Here, the displacement of V_i^F is set to the maximum of the displacements of the vectors V_{n1}^F, V_{n2}^F, or V_{n3}^F as shown in Figure 4.2. We note that the size of SA can be 8×8, 14×14, or 18×18 pixels corresponding to a maximum displacement p of 2, 5 and 7, respectively. If a match within SA is obtained, a flag S_m followed by the displacement (motion) vector of V^F are transmitted. However, if a match is not obtained even within SA, the VC-FAVQ coder switches to the intraframe mode where a dynamically generated, self-organized codebook is used to encode V^F. The interframe coding mode is expressed as follows:

PROCEDURE Interframe mode(**INPUT**: Previous frame $F-1$, Input vector V_i^F, Threshold Δ; **OUTPUT**: Label, Flag)

BEGIN

Flag = FALSE;

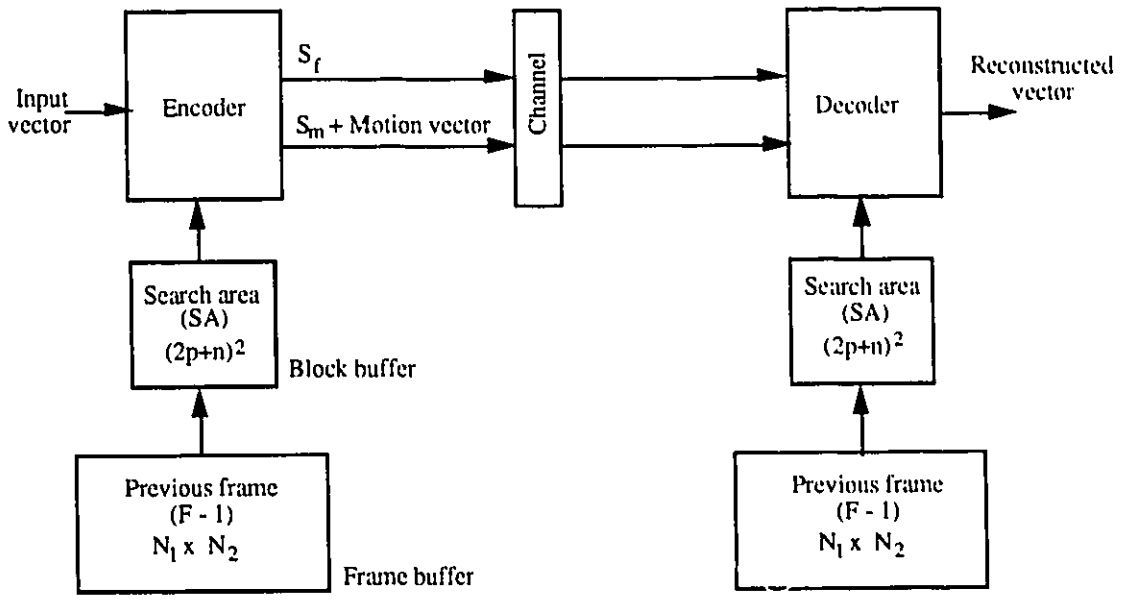


Figure 4.1: Block schematic of VC-FAVQ interframe mode.

IF $\{max(|v_{ij}^F - v_{ij}^{F-1}|)\} \leq \Delta, \quad 1 \leq j \leq L$ **THEN**

Label = S_f;

Flag = TRUE;

ELSE BEGIN

Define a search area SA in $F-1$

Decompose SA into 16-dimensional vectors (4×4 blocks) C_k^{F-1}

IF $min\{max(|v_{ij}^F - c_{kj}^{F-1}|)\} \leq \Delta$ *for* $1 \leq j \leq 16, \quad 1 \leq k \leq (2 \times p + 1)^2$ **THEN**

Determine the Motion vector;

Label = S_m + Motion vector;

Flag=TRUE;

END ELSE;

END.

The bit rate of the interframe mode (R_{inter}) consists of two components:

- R_f, the bit rate of the flags S_f.

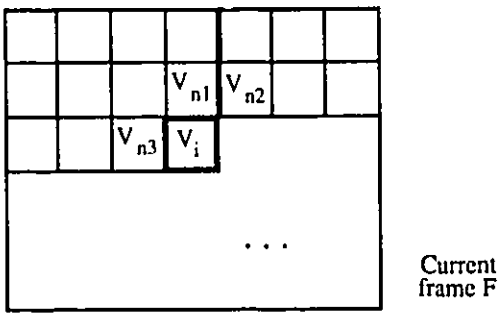


Figure 4.2: V_i^F and V_{n1}^F, V_{n2}^F, V_{n3}^F are the input vector and the neighboring vectors coded previously, respectively.

- The bit rate of the motion vectors,

$$R_{mv} = \frac{\left(\sum_{i=1}^{N_m} \lceil log_2(1 + 2 \times p)^2 \rceil\right)}{(N_1 \times N_2)} + RS_m \tag{4.1}$$

where N_m, $N_1 \times N_2$ and RS_m are the number of motion vectors, the frame size and the bit rate of the flags S_m, respectively. Hence the total bit rate of the interframe mode is given by

$$R_{inter} = R_f + R_{mv} \tag{4.2}$$

Intraframe Mode: In the intraframe mode a highly adaptive VQ technique [52] is employed. Here, two codebooks are used namely: the primary codebook (PC) and a larger secondary codebook (SC) as shown in Figure 4.3. To start with, the vectors of the image to be coded are used as the codewords. For each input vector, PC is searched for a match within a prespecified threshold. If no match is obtained the input vector is transmitted and is also appended to PC as a new codeword. If a match is obtained the index of the corresponding codeword is transmitted. When PC becomes full, the least recently used (LRU) codeword is moved from PC to SC freeing room for the new codeword. From this point on, PC is searched first for a match, however, if it fails, SC is also searched. A new codeword is appended only if no match is obtained in both codebooks, and in this case, the LRU codeword in PC is moved to SC. However, if SC is also full, the LRU codeword in SC is deleted. If a match is obtained in SC, the index of that codeword is

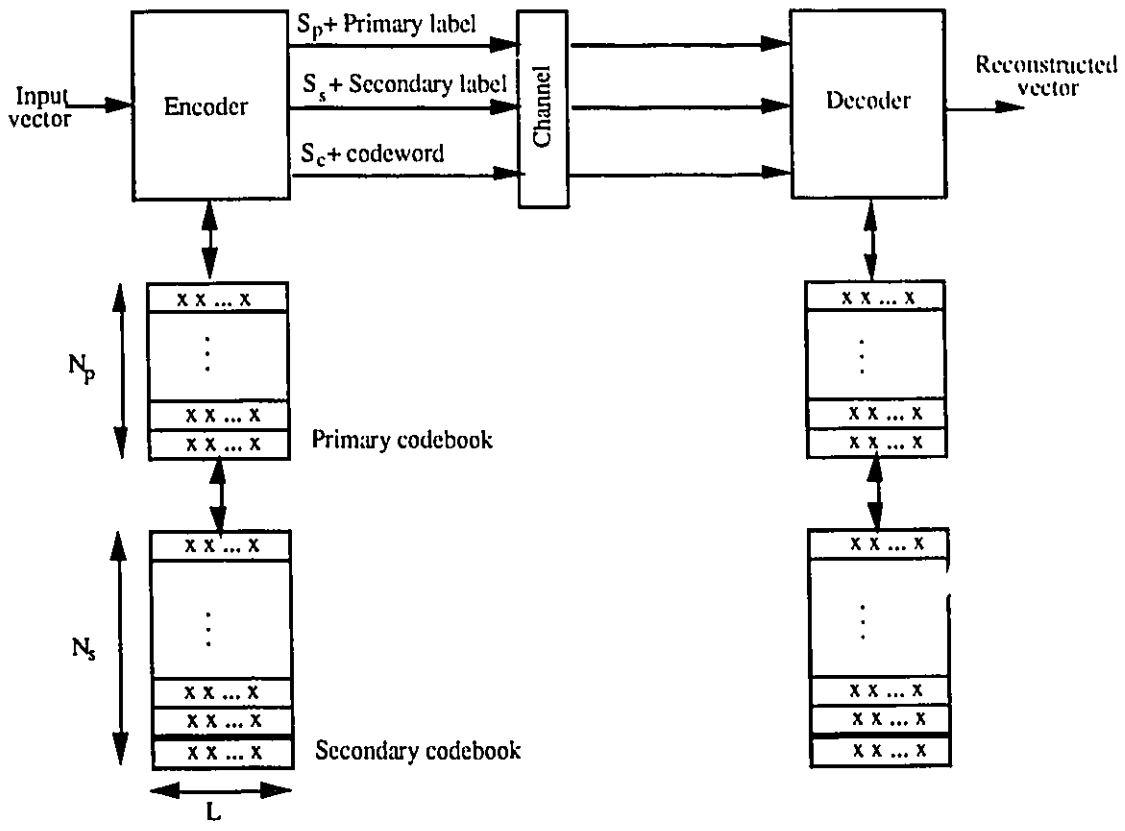


Figure 4.3: Block schematic of VC-FAVQ intraframe mode.

transmitted and the codeword is swapped with the LRU codeword in PC. A flag S_c, S_p or S_s is transmitted before transmitting the codeword, the primary label or the secondary label. The algorithm of the intraframe mode is expressed as follows

PROCEDURE Intraframe mode (**INPUT**: Primary codebook PC, Secondary codebook SC, Input vector V_i^F, Threshold Δ, LRU map; **OUTPUT**: Label)

BEGIN

IF PC is empty **BEGIN**

Place V_i^F in PC;

Label = S_c + V_i^F;

Update the LRU map;

END

ELSE

Search for the closest codeword C_k in PC ;

IF $\{max(|v_{ij}^F - c_{kj}|)\} \leq \Delta, \quad 1 \leq j \leq L$ **THEN BEGIN**

Label $S_p + k$;

Update the LRU map;

END

ELSE BEGIN

IF PC is not full **THEN BEGIN**

Transmit V_i^F to the receiver;

Append V_i^F to PC;

Update the LRU map;

END

ELSE BEGIN

IF SC is empty **THEN BEGIN**

Transmit V_i^F to the receiver;

Move the LRU codeword from PC to SC;

Append V_i^F to PC;

Update the LRU map;

END

ELSE BEGIN

Search for the closest codeword C_m in SC;

IF $\{max(|v_{ij}^F - c_{mj}|)\} \leq \Delta, \quad 1 \leq j \leq L$ **THEN BEGIN**

Label $= S_s + m$;

Swap the LRU codeword PC and the LRU codeword SC;

Update the LRU map;

END

ELSE BEGIN

IF SC is full **THEN BEGIN**

Transmit V_i^F to the receiver;

delete the LRU codeword in SC;

Move the LRU codeword in PC to SC;

Append V_i^F to PC;

Update the LRU map;

END

ELSE BEGIN

Transmit V_i^F receiver;

move the LRU codeword in PC to SC;

Append V_i^F to PC;

Update the LRU map;

END ELSE

END ELSE

END ELSE

END ELSE

END ELSE

END.

The bit rate of the intraframe mode (R_{intra}) consists of three components

- The bit rate of the primary labels,

$$R_p = \frac{N_{pl} \times log_2(N_p)}{(N_1 \times N_2)} + RS_p \tag{4.3}$$

- The bit rate of the secondary labels,

$$R_s = \frac{N_{sl} \times log_2(N_s)}{(N_1 \times N_2)} + RS_s \tag{4.4}$$

- The bit rate of the codewords,

$$R_c = \frac{N_c \times L \times b}{(N_1 \times N_2)} + RS_c \tag{4.5}$$

where N_{pl}, N_{sl}, N_c are the number of primary labels, secondary labels and codewords respectively, and RS_p, RS_s, RS_c are the bit rate for the flags to indicate that the current label corresponds to the primary, secondary, and codeword respectively. The total bit rate of the interframe mode is given by

$$R_{intra} \quad R_p + R_s + R_c \tag{4.6}$$

We now present the VC-FAVQ algorithm

PROCEDURE VC-FAVQ(**INPUT**: Video sequence, Threshold Δ; **OUTPUT**: Labels)
BEGIN
FOR all frames in the Video Sequence **DO**
BEGIN
decompose the current frame into L-dimensional vectors;
FOR all vectors V_i^F $i = 1 \cdots K$ **DO**
BEGIN
Input vector = V_i^F;
Flag = FALSE;
Interframe mode(**INPUT**: Previous frame $F-1$, Input vector V_i^F, Threshold Δ;
OUTPUT: Label, Flag)
IF Flag is FALSE **THEN**
Transmit Label;
ELSE
Intraframe mode (**INPUT**: Primary codebook PC, Secondary
codebook SC, Input vector V_i^F, Threshold Δ,
LRU map, **OUTPUT**: Label)
Transmit Label;
END FOR
END FOR
END.

To summarize, the advantages of VC-FAVQ are:

- The reconstructed vectors are always within a prespecified error threshold.
- Since two modes are employed both the inter- and intraframe correlations are exploited.
- Frame adaptability is achieved at a reduced complexity by using a variable size search area in the interframe mode and by using a dynamically generated codebook in the intraframe mode.
- VC-FAVQ is a single pass technique and hence real-time implementation is possible.

4.3 Simulation and Discussion of Results

Computer simulations were carried out on two standard CCITT image sequences: Miss America and Salesman. Both sequences are 288×360 pixels with 8 bits per pixel. Three sets of experiments were carried out using the proposed technique. In the first set of experiments the performance of VC-FAVQ is evaluated and compared to that of the IIA-VQ (section 2.3.6). In the second set of experiments simulation results were obtained for the Miss America sequence and compared to those of the MPEG coder (section 2.4). In the third set of experiments the performance of the proposed algorithm is studied for sequences with scene changes. In these experiments only every other 4th frame is coded. The rest of the frames are skipped to allow for a larger variation between the successive frames. For the sake of evaluation, the variances of the difference frames are used as a measure of the changes occurring between successive frames [66]. The simulations were performed using different threshold values corresponding to different bit rates.

4.3.1 Coding Performance

The coding performance of the proposed algorithm is evaluated using rate distortion criterion [19]. For an image of size $N_1 \times N_2$ and a maximum pixel value of 255, the Peak

Signal to Noise Ratio (PSNR) of the reconstructed image is calculated by:

$$PSNR = 10 \times log\frac{(255)^2}{MSE} \tag{4.7}$$

where MSE is the mean squared error defined as:

$$MSE = \frac{\sum_{i=1}^{N_1}\sum_{j=1}^{N_2}(X_{ij} - Y_{ij})^2}{(N_1 \times N_2)} \tag{4.8}$$

where X_{ij} and Y_{ij} are the intensity of the pixel (i, j) in the original and the reconstructed image, respectively. The total bit rate R_t is sum of the bit rates of the interframe mode (R_{inter}) and the intraframe mode (R_{intra}) and is calculated using Equations 4.2 and 4.6:

$$R_t = R_f + R_{mv} + R_p + R_s + R_c \tag{4.9}$$

In the first set of experiments, simulations results are obtained for the Miss America sequence for threshold values Δ of 10 and 16. The results are tabulated in Tables 4.1 and 4.2. The average bit rates for $\Delta = 10$ and $\Delta = 16$ are 0.60 and 0.25 bits/pixel (bpp), respectively. Figure 4.5 shows the percentage of occurrence of the codes generated in the interframe mode (S_f and the motion vectors) and the codes generated in the intraframe mode (the primary labels, the secondary labels and the codewords) as a function of the frame number. From Figures 4.4 and 4.5, it can be seen that due to the high interframe correlations, the interframe codes are dominant and as the sequence progresses and the interframe motion increases, the occurrence of the motion vectors for a maximum displacement p of 5 and 7 are also increases. The results of applying IHA-VQ on the same sequence are used for comparison. The comparative charts of the total bit rate and PSNR on a frame by frame basis are shown in Figures 4.6, 4.7, 4.8 and 4.9. It can be seen that the average coding performance of the two techniques are almost identical. However, when there are significant changes between the frames (frames 58 to 97) (see Figure 4.4), VC-FAVQ outperforms IHA-VQ by adapting to the changes and maintaining a constant PSNR throughout the sequence. Moreover, VC-FAVQ has a much smaller variation in bit rate (0.49 to 0.69 and 0.20 to 0.30 bpp) compared to IHA-VQ (0.34 to 0.74 and 0.15 to 0.42 bpp) technique. Pictorial results are presented in Figures 4.10 for frame 64. The original frame number 64 and its coded version at 0.60 bpp are shown in Figures 4.10

and 4.11, respectively. The corresponding error frame (normalized) is shown in Figure 4.12 for subjective evaluation. The reconstructed frame number 64 at 0.25 bpp and the corresponding error frame are shown in Figures 4.13 and 4.14, respectively. It is clearly seen that the over all subjective quality of the coded frames are excellent. The high details and the edges are preserved after coding although some fuzziness can be seen around the eyes at 0.25 bpp.

The results of applying the VC-FAVQ technique on the Miss America sequence for $\Delta = 12$ are tabulated in Table 4.3. The bit rate and PSNR of the frames are shown in Figures 4.15 and 4.16, respectively. It can be seen that there is a smooth transition from one frame to the next in terms of both the bit rate and PSNR. The original frame number 79 is shown in Figure 4.17 and its coded version in Figure 4.18 where the VC-FAVQ is used and results in an average bit rate of 0.53 bpp and an average PSNR of 38 dB. The corresponding error frame is shown in Figure 4.19. The corresponding results are obtained using the MPEG coder and shown in Figure 4.16. The reconstructed and error frames for frame number 79 are shown in Figure 4.20 and 4.21. It can be seen that VC-FAVQ outperforms the MPEG coder and yields much improved objective as well as subjective results by adapting to the changes in the image sequence.

The simulation results for a test sequence with scene change are tabulated in Table 4.4. The test sequence is composed of two different scenes. The first scene consists of 6 frames of the Miss America sequence. The second scene is composed of 6 frames of the Salesman sequence. Figure 4.23 shows the percentage of occurrence of the codes generated in the interframe mode (S_f and the motion vectors) and the codes generated in the intraframe mode (the primary labels, the secondary labels, and the codewords) as a function of the frame number. From Figures 4.22 and 4.23, it can be seen that both modes are utilized as expected. That is, at scene changes the intraframe codes are dominant; but as the interframe correlation increases the interframe codes become dominant. The total bit rate and the PSNR of the frames are shown in Figures 4.24 and 4.25, respectively. It can be seen from Figure 4.25 that the transition from one scene to the next is smooth and the PSNR value of the first frame of the Salesman sequence (34.7

dB) is not low compared to average PSNR of the sequence (36.7 dB). Thus, the VC-FAVQ encoder adapts to to the scene changes. The original frame number 5 of the Salesman sequence after the scene change from the Miss America sequence is shown in Figure 4.26. The corresponding reconstructed and error frames are shown in Figures 4.27 and 4.28, respectively. It is clearly seen that the overall subjective quality of the reconstructed image is good, the edges and the details are preserved after coding although their are some fuzziness and minor blocking effects. The main reason for this is that the threshold value (14) is relatively high for this highly detailed image. In general, the quality can be improved using a variable threshold.

4.4 Computational Complexity

We recall from sections 2.2.4 and 2.3.1 that for K input vectors of dimension $L = n \times n$, the computational complexity of VQ and the full search block matching (FBMA) are $O(KLN)$ and $O(KL(2p+1)^2)$ where N is the codebook size and p is the maximum displacement. For each vector ($K = 1$) the computational complexity of VC-FAVQ consists of:

- The computational complexity of FBMA which is given by:

$$C_{FBMA} = O(L(2p+1)^2) \tag{4.10}$$

- The computational complexity of searching a primary codebook of size N_p is:

$$C_{primary} = O(LN_p) \tag{4.11}$$

- The computational complexity of searching a secondary codebook of size N_s is given by:

$$C_{secondary} = O(LN_s) \tag{4.12}$$

If η_1 is the probability of obtaining a match within the search area, then for K input vectors the complexity of the interframe mode C_{inter} is:

$$C_{inter} = \left(K \times L\left(\eta_1(2p+1)^2\right)\right) \tag{4.13}$$

The complexity C_{intra} of the intraframe mode for K input vectors is calculated using Equations 4.11 and 4.12 and is given by:

$$C_{intra} = K \times L \left(\eta_2(N_p + (2p+1)^2) + (1-\eta_1-\eta_2)(N_s + N_p + (2p+1)^2)\right) \quad (4.14)$$

where η_2 is the probability of obtaining a match in the primary code book. Hence the complexity $C_{VC-FAVQ}$ of VC-FAVQ is

$$\begin{aligned} C_{VC-FAVQ} &= C_{inter} + C_{intra} && (4.15)\\ &= K \times L\left\{\eta_1(2p+1)^2 + \eta_2\left(N_p + (2p+1)^2\right) + \right. && (4.16)\\ &\quad \left.(1-\eta_1-\eta_2)\left(N_s + N_p + (2p+1)^2\right)\right\} && (4.17) \end{aligned}$$

4.5 Summary

In this chapter, we have presented an algorithm for video compression. In this technique, the input vector is encoded either in the interframe mode or in the intraframe mode depending on a prespecified error threshold. This guarantees that the distortion in the reconstructed frames is bounded. The proposed scheme is highly adaptive. High adaptability is achieved by using a variable size search area in the interframe mode and by using a dynamically generated codebook in the intraframe mode. Hence high performance video compression is achieved at a reduced complexity. Computer simulations results demonstrate the excellent coding performance of VC-FAVQ. Excellent quality images (PSNR of 39.3 dB) were obtained at compression ratio of 13:1. In addition, VC-FAVQ is simple to implement and is elegant for constant quality applications.

Frame No.	R_f	R_m $p=2$	R_m $p=5$	R_m $p=7$	R_p	R_e	R_c	R_t	PSNR
13	0.0365	0.1141	0.1319	0.0515	0.0006	0.0131	0.2161	0.5638	39.60
16	0.0366	0.1269	0.1234	0.0394	0.0008	0.0155	0.2161	0.5581	39.33
19	0.0372	0.1220	0.0953	0.0691	0.0011	0.0170	0.2906	0.6323	39.34
22	0.0414	0.1360	0.0939	0.0330	0.0008	0.0154	0.2805	0.6010	39.35
25	0.0431	0.1307	0.0751	0.0554	0.0008	0.0152	0.2653	0.5855	39.44
28	0.0446	0.1229	0.0866	0.0480	0.0008	0.0147	0.2666	0.5842	39.41
31	0.0444	0.1488	0.0660	0.0337	0.0006	0.0239	0.1706	0.4880	39.23
34	0.0451	0.1392	0.0573	0.0535	0.0007	0.0159	0.2641	0.5758	39.35
37	0.0481	0.1420	0.0511	0.0467	0.0006	0.0137	0.2337	0.5359	39.40
40	0.0441	0.1374	0.0731	0.0434	0.0008	0.0171	0.2451	0.5610	39.31
43	0.0439	0.1096	0.0901	0.0666	0.0008	0.0162	0.2540	0.5812	39.29
46	0.0470	0.1492	0.0499	0.0363	0.0009	0.0168	0.2792	0.5793	39.35
49	0.0468	0.0643	0.1471	0.0558	0.0009	0.0164	0.2792	0.6105	39.46
52	0.0464	0.0778	0.1151	0.0729	0.0006	0.0159	0.2957	0.6244	39.43
55	0.0493	0.1082	0.0831	0.0527	0.0007	0.0144	0.2729	0.5813	39.41
58	0.0474	0.1177	0.0723	0.0582	0.0007	0.0152	0.2767	0.5882	39.41
61	0.0474	0.0580	0.1719	0.0361	0.0006	0.0164	0.2628	0.5932	39.46
64	0.0456	0.0478	0.1644	0.0626	0.0009	0.0174	0.3260	0.6647	39.39
67	0.0453	0.0230	0.1938	0.0737	0.0007	0.0154	0.2628	0.6147	39.22
70	0.0454	0.0190	0.2252	0.0456	0.0007	0.0149	0.2451	0.5959	39.31
73	0.0454	0.0149	0.2269	0.0482	0.0008	0.0166	0.2489	0.6017	39.34
76	0.0426	0.0089	0.2527	0.0372	0.0008	0.0172	0.2969	0.6563	39.30
79	0.0425	0.0063	0.1147	0.1984	0.0008	0.0179	0.2653	0.6459	39.21
82	0.0416	0.0058	0.2190	0.0832	0.0008	0.0206	0.2780	0.6490	39.10
85	0.0429	0.0074	0.2295	0.0644	0.0010	0.0172	0.2919	0.6543	39.15
88	0.0419	0.0135	0.2484	0.0372	0.0006	0.0182	0.3083	0.6681	38.95
91	0.0454	0.0455	0.1759	0.0517	0.0011	0.0181	0.3487	0.6864	39.04
94	0.0443	0.0417	0.1846	0.0541	0.0009	0.0197	0.3184	0.6637	39.13
97	0.0442	0.0409	0.1892	0.0589	0.0006	0.0164	0.2300	0.5802	39.08

Table 4.1: Simulation results using VC-FAVQ on the Miss America sequence for threshold $\Delta = 10$.

Frame No.	R_f	R_m $p=2$	R_m $p=5$	R_m $p=7$	R_p	R_s	R_c	R_t	PSNR
13	0.0427	0.1053	0.0152	0.0146	0.0007	0.0144	0.0341	0.2270	36.57
16	0.0413	0.0949	0.0466	0.0074	0.0007	0.0148	0.0366	0.2423	36.48
19	0.0408	0.0666	0.0898	0.0057	0.0006	0.0143	0.0329	0.2507	36.46
22	0.0039	0.0432	0.0856	0.0264	0.0009	0.0155	0.0417	0.2172	36.56
25	0.0440	0.0958	0.0162	0.0144	0.0008	0.0140	0.0303	0.2155	36.58
28	0.0449	0.0494	0.0628	0.0235	0.0006	0.0143	0.0354	0.2309	36.50
31	0.0449	0.0932	0.0095	0.0164	0.0007	0.0145	0.0229	0.2021	36.46
34	0.0440	0.0808	0.0451	0.0035	0.0007	0.0148	0.0267	0.2156	36.50
37	0.0442	0.1074	0.0071	0.0057	0.0007	0.0143	0.0278	0.2072	36.54
40	0.0423	0.0509	0.0907	0.0122	0.0007	0.0148	0.0229	0.2345	36.54
43	0.0426	0.0618	0.0750	0.0118	0.0008	0.0148	0.0204	0.2272	36.39
46	0.0442	0.0536	0.0610	0.0214	0.0008	0.0158	0.0267	0.2235	36.45
49	0.0441	0.0924	0.0251	0.0055	0.0011	0.0175	0.0242	0.2099	36.27
52	0.0436	0.0721	0.0540	0.0078	0.0009	0.0163	0.0318	0.2265	36.19
55	0.0440	0.0583	0.0719	0.0063	0.0007	0.0150	0.0216	0.2178	36.26
58	0.0425	0.0799	0.0552	0.0062	0.0009	0.0167	0.0242	0.2256	36.44
61	0.0430	0.0593	0.0749	0.0100	0.0006	0.0157	0.0331	0.2366	36.44
64	0.0413	0.0501	0.0904	0.0216	0.0006	0.0163	0.0407	0.2610	36.28
67	0.0406	0.0454	0.1082	0.0169	0.0006	0.0152	0.0344	0.2613	36.16
70	0.0403	0.0766	0.0861	0.0002	0.0007	0.0150	0.0316	0.2505	36.22
73	0.0389	0.0115	0.1668	0.0170	0.0007	0.0152	0.0280	0.2781	36.31
76	0.0379	0.0051	0.1893	0.0103	0.0008	0.0173	0.0204	0.2811	36.16
79	0.0370	0.0032	0.1355	0.0807	0.0009	0.0173	0.0267	0.3013	36.09
82	0.0372	0.0063	0.1707	0.0349	0.0008	0.0175	0.0318	0.2992	35.93
85	0.0378	0.0066	0.1619	0.0406	0.0006	0.0163	0.0191	0.2829	35.97
88	0.0388	0.0192	0.1649	0.0049	0.0006	0.0164	0.0480	0.2928	35.74
91	0.0401	0.0773	0.0805	0.0028	0.0010	0.0205	0.0280	0.2502	35.82
94	0.0400	0.0208	0.1529	0.0081	0.0007	0.0163	0.0255	0.2643	36.00
97	0.0398	0.0100	0.1644	0.0138	0.0040	0.0161	0.0115	0.2596	36.11

Table 4.2: Simulation results using VC-FAVQ on the Miss America sequence for threshold $\Delta = 16$.

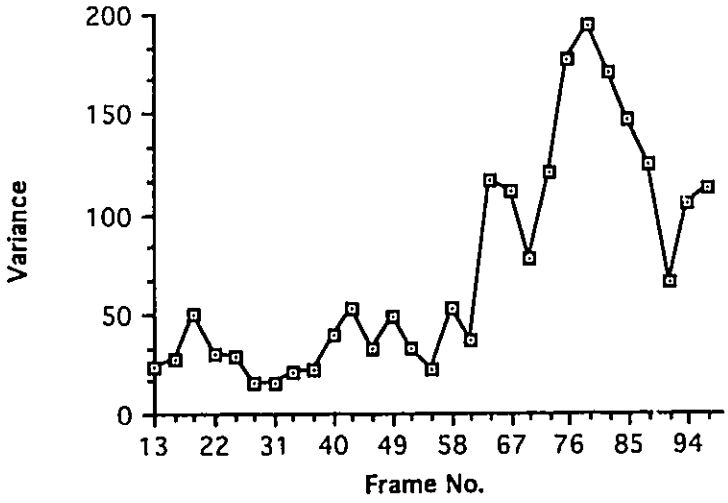


Figure 4.4: The variance of the difference frames as a function of the frame number for the Miss America sequence used in the first and the second set of experiments.

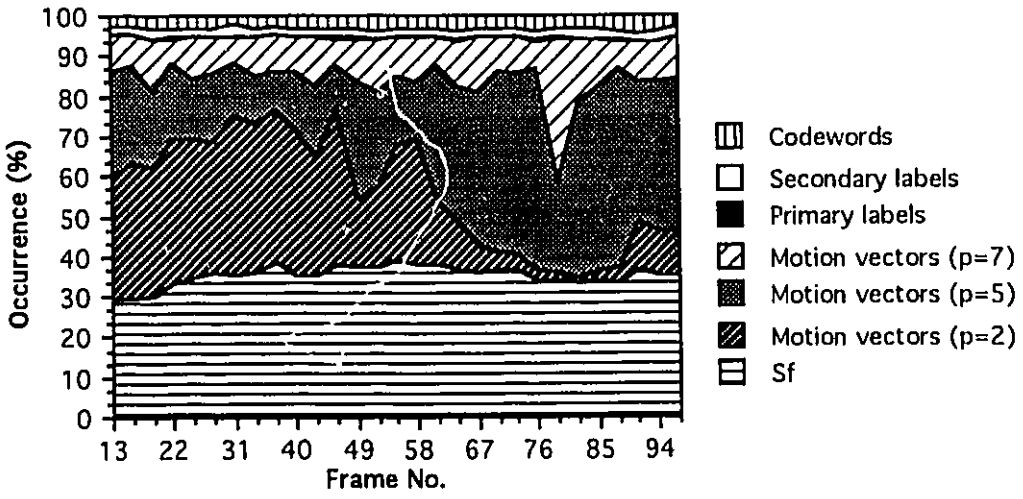


Figure 4.5: Occurrence percentage of S_f, motion vectors for a maximum displacement p of 2, 5 and 7, the primary labels, the secondary labels and the codewords for the Miss America sequence (average bit rate = 0.60 bpp for $\Delta = 10$).

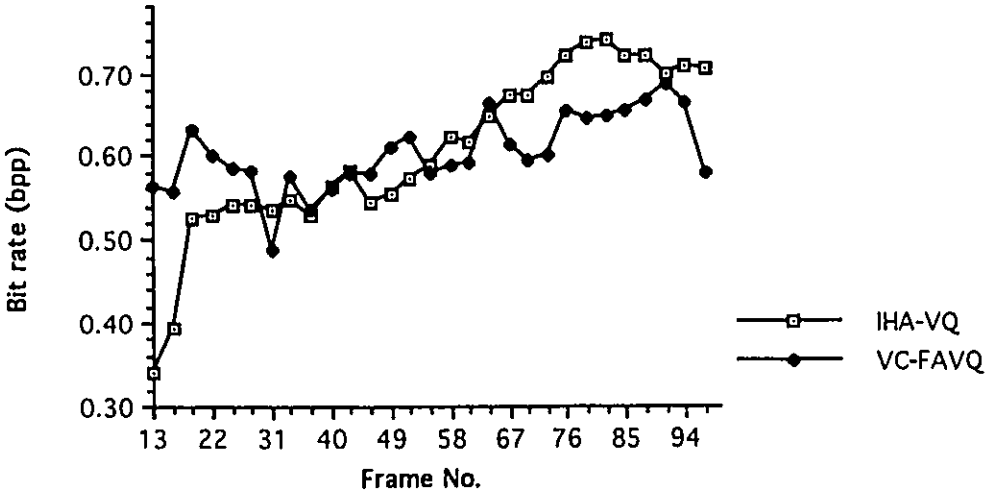


Figure 4.6: The total bit rate as a function of frame number using VC-FAVQ (average = 0.60 bpp) and IHA-VQ (average = 0.61 bpp) on the Miss America sequence.

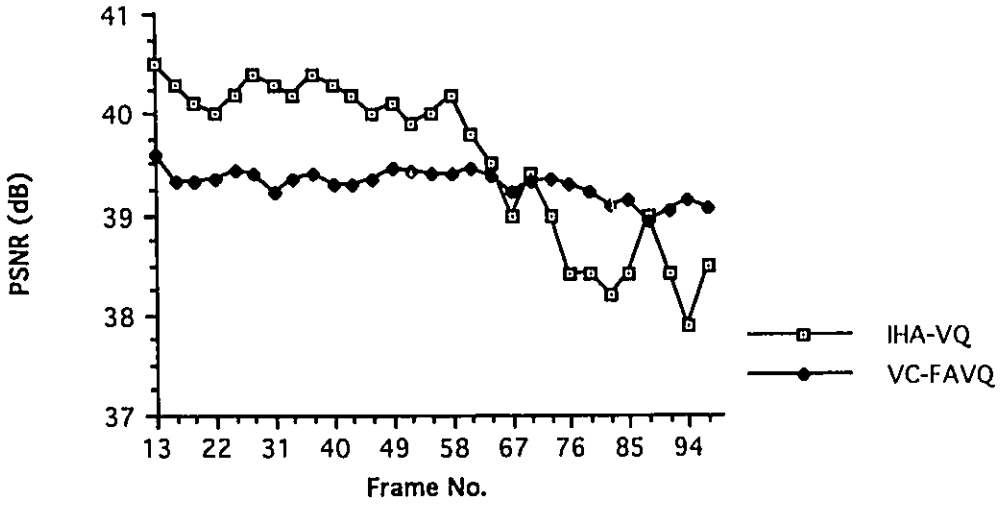


Figure 4.7: The PSNR vs. frame number using VC-FAVQ (average = 39.3 dB) and IHA-VQ (average = 39.5 dB) on the Miss America Sequence.

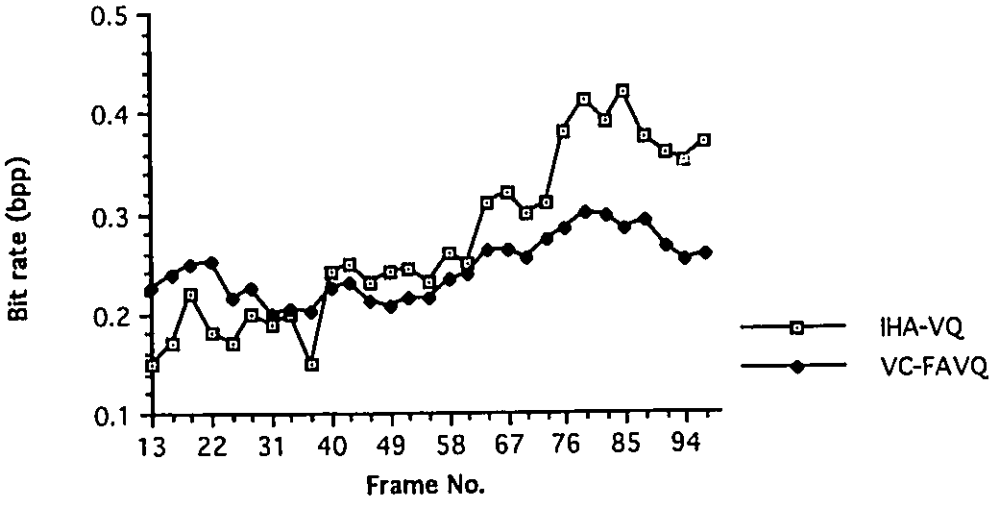


Figure 4.8: The total bit rate as a function of frame number using VC-FAVQ (average = 0.25 bpp) and IHA-VQ (average = 0.24 bpp) on the Miss America sequence.

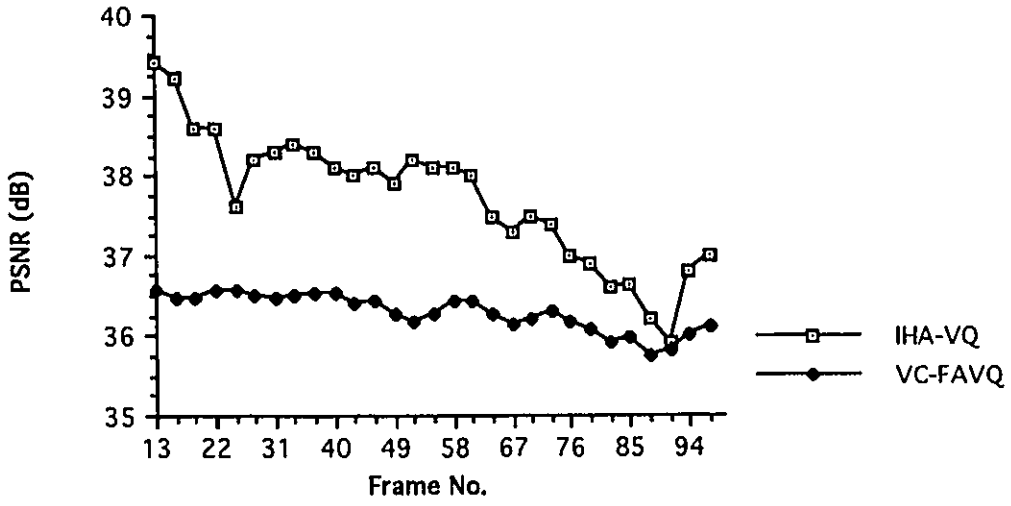


Figure 4.9: The PSNR vs. frame number using VC-FAVQ (average = 36.3 dB) and IHA-VQ (average = 37 dB) on the Miss America Sequence.

Figure 4.10: Original frame number 64 of the Miss America sequence.

Figure 4.11: Reconstructed frame number 64 of the Miss America sequence using VC-FAVQ at 0.60 bpp.

Figure 4.12: Error frame number 64 of the Miss America sequence using VC-PAVQ at 0.60 bpp.

Figure 4.13: Reconstructed frame number 64 of the Miss America sequence using VC-FAVQ at 0.25 bpp.

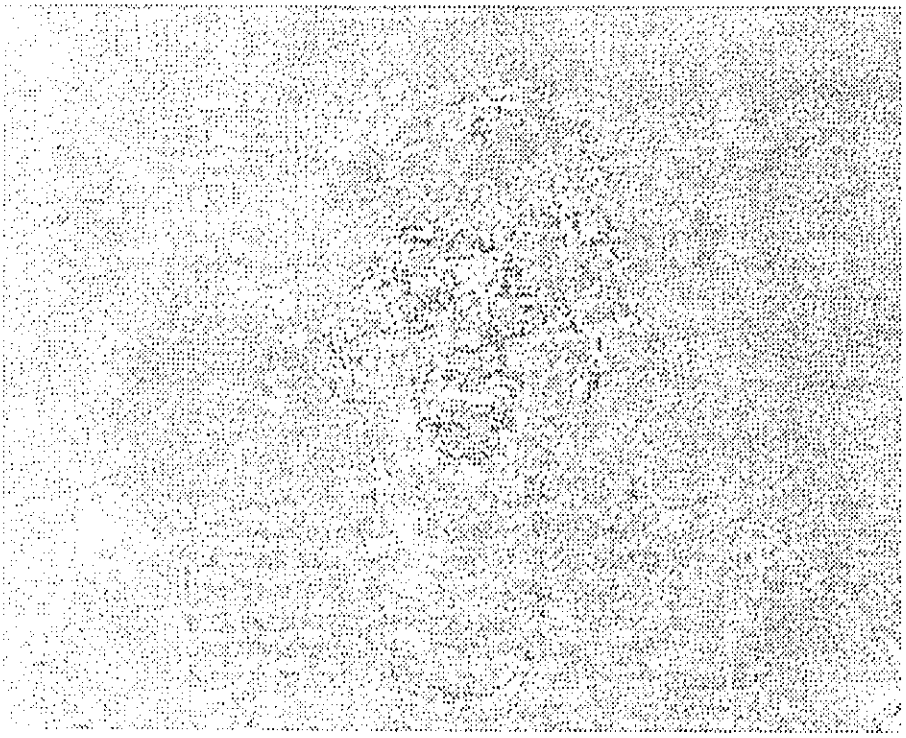


Figure 4.14: Error frame number 64 of the Miss America sequence using VC-FAVQ at 0.25 bpp.

Frame No.	R_f	R_m $p=2$	R_m $p=5$	R_m $p=7$	R_r	R_s	R_c	R_t	PSNR
13	0.0364	0.1115	0.1344	0.0563	0.0006	0.0132	0.1655	0.5179	39.31
16	0.0365	0.1345	0.1187	0.0359	0.0008	0.0160	0.1819	0.5243	39.07
19	0.0366	0.1260	0.1076	0.0582	0.0014	0.0155	0.2060	0.5513	39.01
22	0.0412	0.1447	0.0894	0.0317	0.0010	0.0141	0.2060	0.5281	38.95
25	0.0419	0.0752	0.0968	0.0852	0.0006	0.0156	0.1668	0.4821	39.07
28	0.0434	0.1231	0.0582	0.0467	0.0006	0.0159	0.1870	0.4749	38.93
31	0.0444	0.1614	0.0606	0.0264	0.0007	0.0214	0.1162	0.4311	38.85
34	0.0455	0.1274	0.0868	0.0422	0.0006	0.0151	0.1883	0.5059	38.98
37	0.0467	0.1402	0.0624	0.0471	0.0008	0.0148	0.1605	0.4725	39.01
40	0.0435	0.1274	0.0910	0.0462	0.0009	0.0171	0.1668	0.4929	38.96
43	0.0442	0.1263	0.0787	0.0568	0.0007	0.0149	0.2186	0.5402	38.94
46	0.0463	0.1332	0.0689	0.0482	0.0008	0.0167	0.1946	0.5087	38.94
49	0.0476	0.0780	0.1275	0.0602	0.0006	0.0155	0.1971	0.5265	39.01
52	0.0468	0.0957	0.0957	0.0596	0.0006	0.0149	0.1946	0.5079	38.82
55	0.0493	0.0538	0.1542	0.0598	0.0006	0.0149	0.1832	0.5158	39.03
58	0.0466	0.0871	0.1219	0.0568	0.0009	0.0152	0.2034	0.5319	39.04
61	0.0479	0.1366	0.0685	0.0383	0.0006	0.0145	0.1921	0.4985	38.94
64	0.0445	0.0241	0.2181	0.0488	0.0008	0.0194	0.2161	0.5718	38.87
67	0.0456	0.0154	0.2133	0.0679	0.0006	0.0145	0.1883	0.5456	38.84
70	0.0460	0.0383	0.1943	0.0544	0.0007	0.0158	0.1579	0.5074	38.73
73	0.0446	0.0133	0.2383	0.0457	0.0007	0.0168	0.1845	0.5439	38.94
76	0.0429	0.0065	0.2395	0.0609	0.0008	0.0178	0.1946	0.5630	38.75
79	0.0424	0.0067	0.1692	0.1409	0.0009	0.0193	0.1984	0.5778	38.55
82	0.0414	0.0116	0.2343	0.0620	0.0009	0.0220	0.2123	0.5845	38.53
85	0.0425	0.0060	0.2427	0.0613	0.0010	0.0160	0.1870	0.5565	38.56
88	0.0417	0.0153	0.2414	0.0486	0.0009	0.0190	0.2148	0.5817	38.40
91	0.0456	0.0177	0.2018	0.0726	0.0007	0.0185	0.2198	0.5767	38.56
94	0.0442	0.0306	0.2117	0.0481	0.0007	0.0190	0.2161	0.5704	38.68
97	0.0441	0.0150	0.2334	0.0536	0.0006	0.0171	0.1453	0.5091	38.64

Table 4.3: Simulation results using VC-FAVQ on the Miss America sequence for threshold $\Delta = 12$.

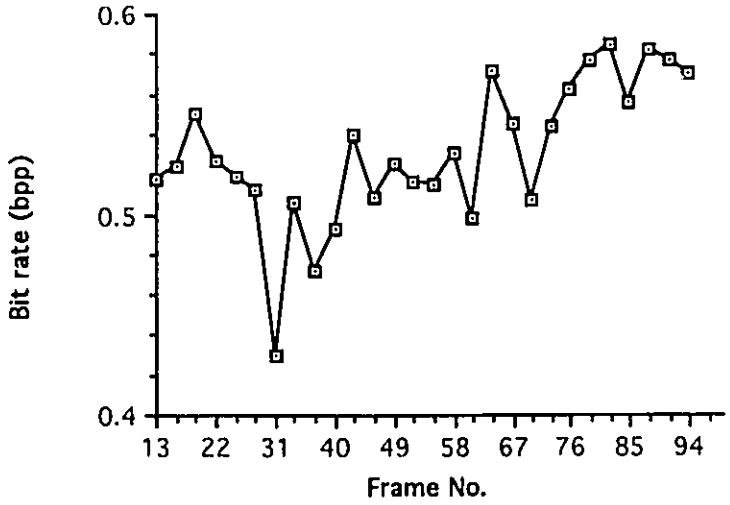


Figure 4.15: The total bit rate as a function of frame number using VC-FAVQ on the Miss America sequence (average bit rate = 0.53 bpp for $\Delta = 12$).

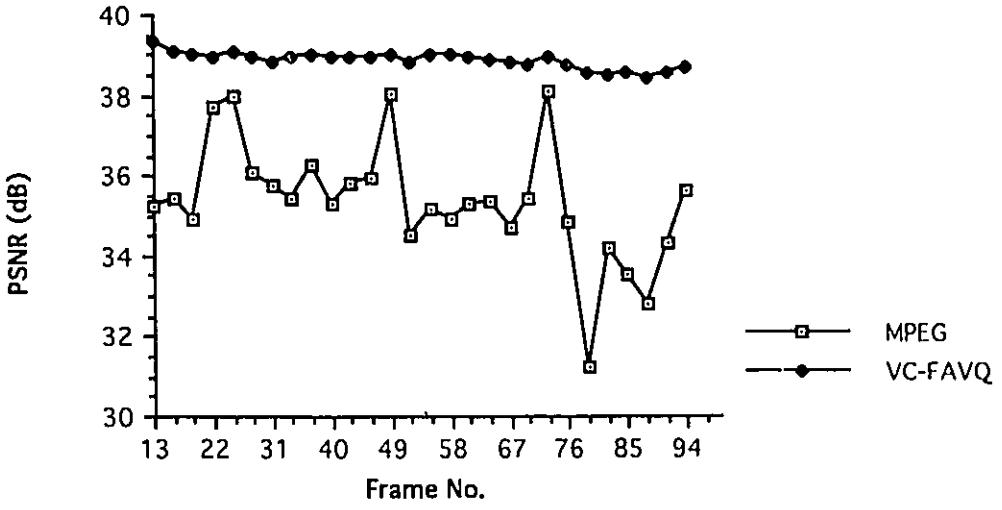


Figure 4.16: PSNR vs. frame number using VC-FAVQ and MPEG on the Miss America sequence.

Figure 4.17: Original frame number 79 of the Miss America sequence.

Figure 4.18: Reconstructed frame number 79 of the Miss America sequence using VC-FAVQ at 0.53 bpp.

Figure 4.19: Error frame number 79 of the Miss America sequence using VC-FAVQ at 0.53 bpp.

Figure 4.20: Reconstructed frame number 79 of the Miss America sequence using MPEG at 0.50 bpp.

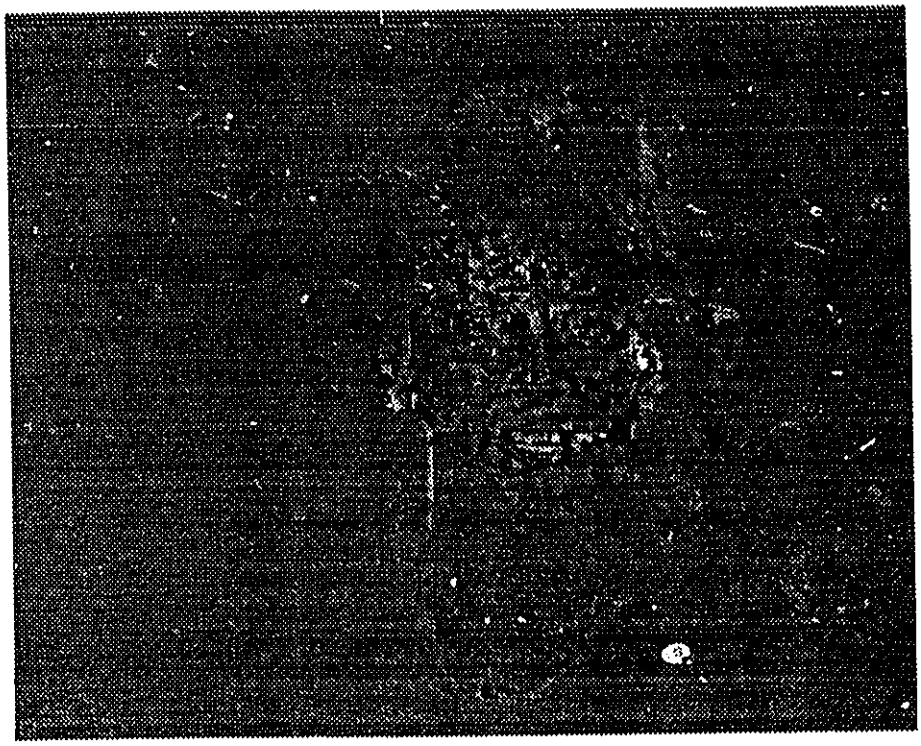


Figure 4.21: Error frame number 79 of the Miss America sequence using MPEG at 0.50 bpp.

Frame No.	R_f	R_m $p=2$	R_m $p=5$	R_m $p=7$	R_p	R_s	R_c	R_t	PSNR
1	0.0000	0.0000	0.0000	0.0000	0.1533	0.2948	1.3191	1.7672	39.72
2	0.0217	0.1822	0.1250	0.0306	0.0006	0.0120	0.1226	0.4947	39.25
3	0.0280	0.1091	0.2120	0.0104	0.0008	0.0145	0.1478	0.5226	39.02
4	0.0358	0.1241	0.1366	0.0492	0.0006	0.0145	0.1125	0.4713	38.81
5	0.0363	0.1159	0.1423	0.0441	0.0006	0.0135	0.1213	0.4739	38.99
6	0.0364	0.1071	0.1598	0.0357	0.0008	0.0145	0.1264	0.4807	38.76
7	0.0016	0.0180	0.0583	0.0709	0.1001	0.2059	1.3717	1.8266	34.70
8	0.0360	0.1259	0.1432	0.0175	0.0012	0.0206	0.2161	0.5605	34.55
9	0.0346	0.1305	0.1367	0.0111	0.0046	0.0247	0.3702	0.7124	34.26
10	0.0316	0.0808	0.1679	0.0508	0.0031	0.0362	0.4132	0.7835	34.06
11	0.0327	0.1017	0.1429	0.0383	0.0036	0.0401	0.4271	0.7863	34.40
12	0.0347	0.1205	0.1088	0.0420	0.0033	0.0380	0.4157	0.7630	34.18

Table 4.4: Simulation results using VC-FAVQ on a sequence with scene change for $\Delta = 12$. Frames 1-6 are from the Miss America sequence and frames 7-12 are from the Salesman sequence.

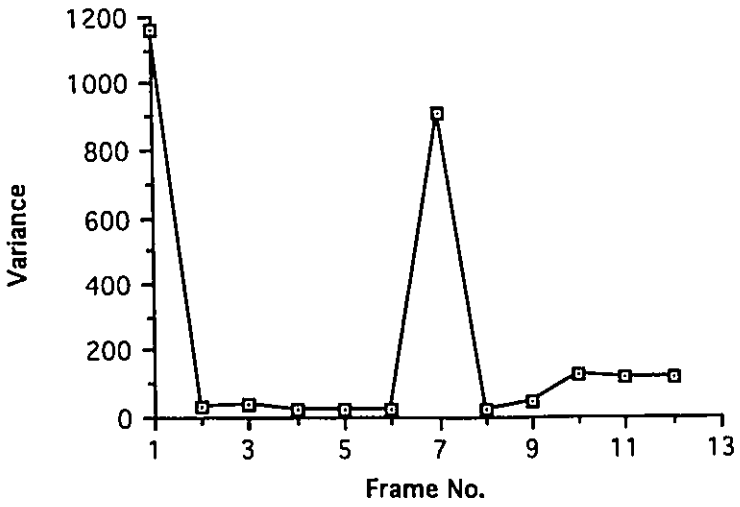


Figure 4.22: The variance of the difference frames as a function of the frame number for the sequence with scene change used in the third set of experiments.

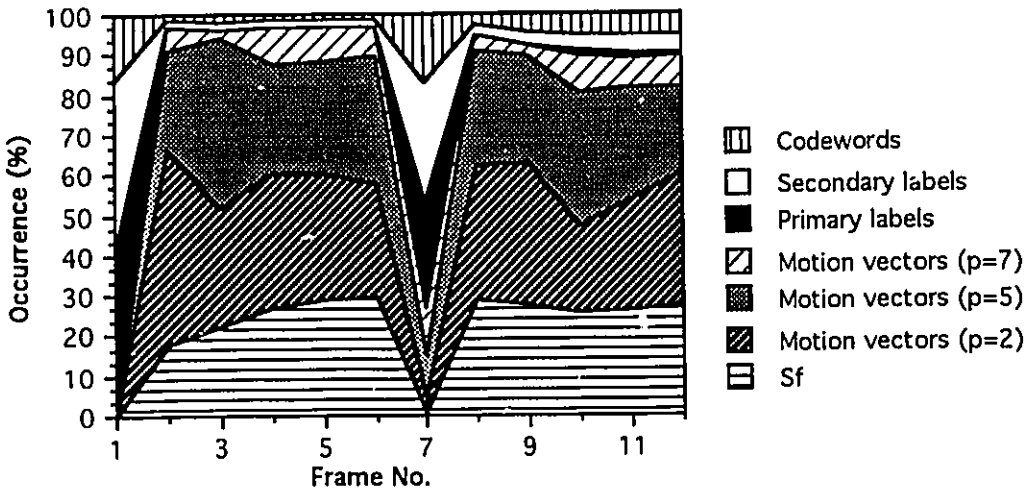


Figure 4.23: Occurrence percentage of S_f, motion vectors for a maximum displacement p of 2, 5 and 7, the primary labels, the secondary labels and the codewords for the sequence with scene change (average bit rate = 0.80 bpp for $\Delta = 14$).

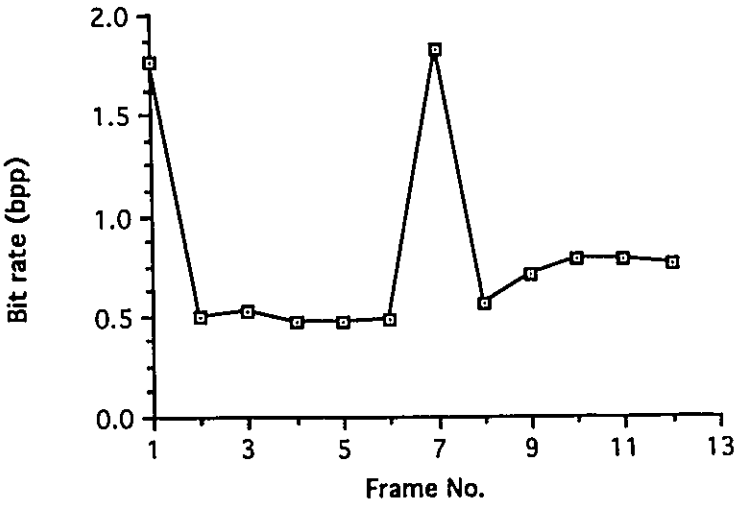


Figure 4.24: The total bit rate as a function of the frame number for the sequence with scene change (average bit rate = 0.80 bpp for $\Delta = 14$).

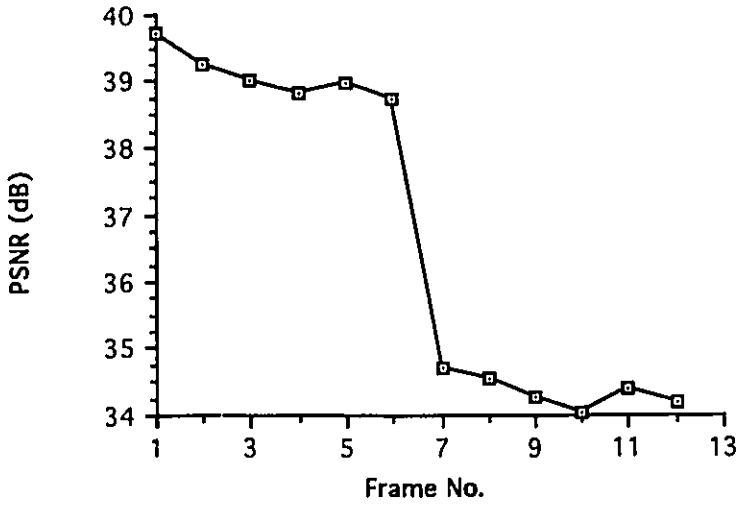


Figure 4.25: The PSNR vs. frame number for the sequence with scene change (average bit rate = 0.80 bpp for $\Delta = 14$).

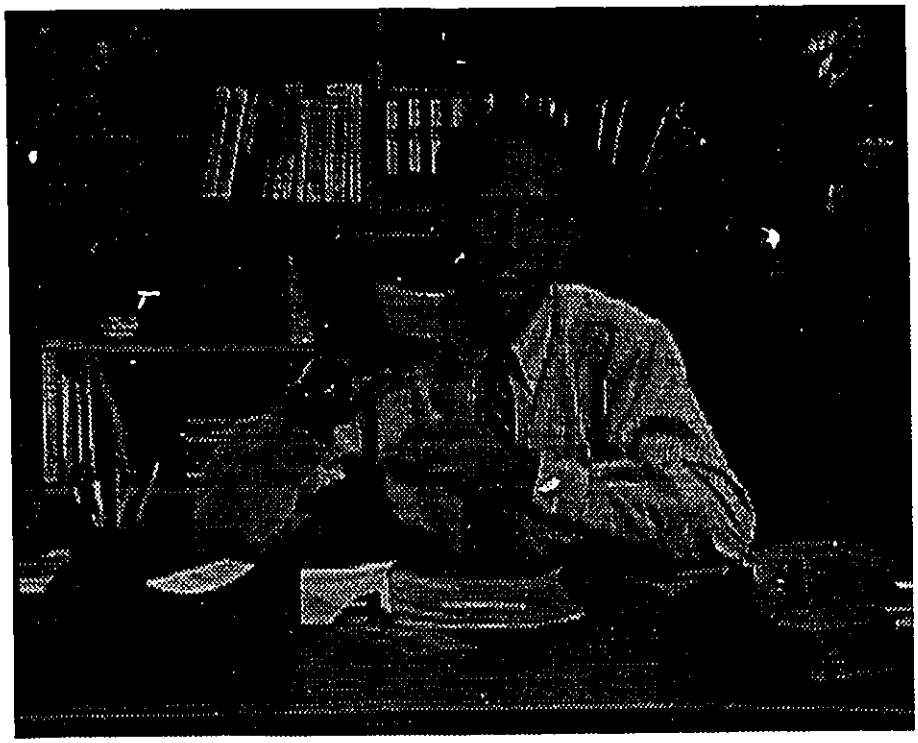


Figure 4.26: Original frame number 5 of the Salesman sequence after the scene change from the Miss America Sequence.

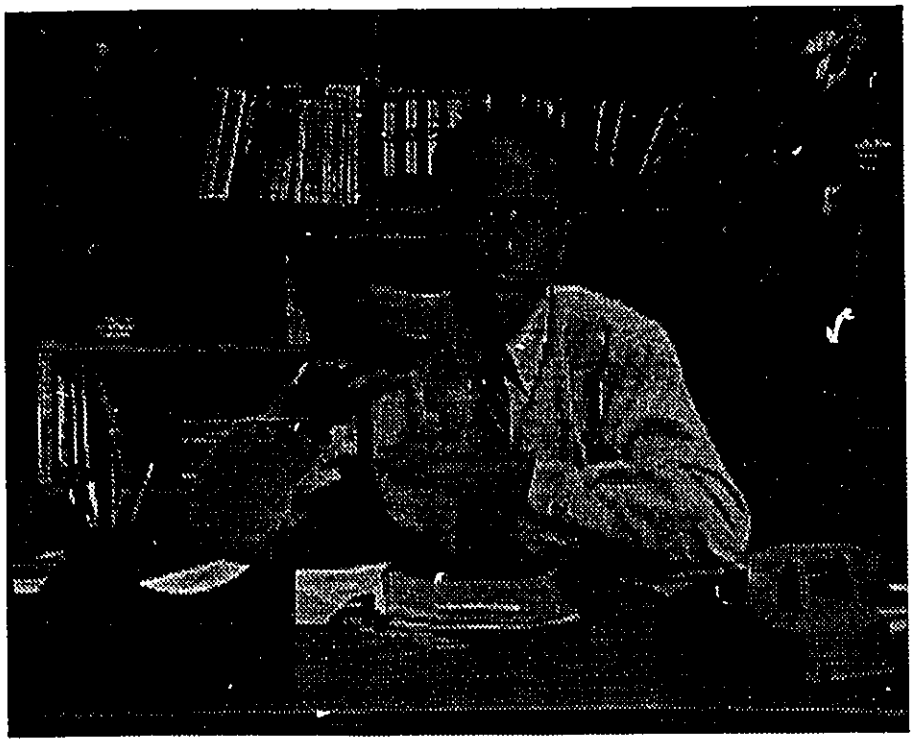


Figure 4.27: Reconstructed frame number 5 of the Salesman sequence after the scene change from the Miss America Sequence at 0.80 bpp.

Figure 4.28: Error frame number 5 of the Salesman sequence after the scene change from the Miss America Sequence at 0.80 bpp.

Chapter 5

Associative Memory Architecture for Video Compression

5.1 Introduction

In this chapter, an associative memory architecture for real time video compression using frame adaptive vector quantization (VC-FAVQ) is presented. We recall from chapter 3 that for video compression applications, a combination of the inter- and intra-image architectures are typically used. This combined architecture may result in inefficiencies in implementation as the common elements between the two algorithms and architectures may not be fully exploited. In addition, these designs are complex and/or non-cascadable which results in an increased hardware requirement and reduced flexibility in VLSI implementation. Vector quantization and motion estimation using block matching can be viewed as template matching processes where each input vector is compared with a finite set of templates (codewords and candidate blocks). Associative memories are efficient for template matching in parallel [107].

Recently a high speed architecture for VQ using content addressable memory (CAM) has been reported [92]. Here, the input image is stored in the CAM and the search operation is executed in parallel in the directions of K and L based on the content. We note that a straight forward extension of this architecture is not efficient to implement VC-

FAVQ since the algorithm cannot be mapped directly onto the architecture. In addition, the architecture is complex for large images and involve a large execution time. We propose a novel storage concept where the image data is stored by association rather than by contents [111]. Based on this concept, we propose an unified associative memory architecture for real-time implementation of motion estimation using block matching and frame adaptive vector quantization [109]. We also propose an efficient implementation of an LRU algorithm which is required to organize the primary and secondary codebooks [111]. The proposed architecture for video compression has the advantages of simplicity, partitionability, and modularity and is hence suitable for VLSI implementation.

The rest of the chapter is organized as follows. In section 5.2 the proposed storage concept is detailed. In sections 5.3 and 5.4 we present the implementation of the FBMA (full search block matching) and VQ using an associative memory. The design for implementing the LRU algorithm is detailed in section 5.4.1. The associative memory architecture for real-time video compression is presented in section 5.5. The analysis of the execution time is detailed in section 5.6. The summary is then presented in section 5.7.

5.2 Storage Concept

We now introduce the storage concept, which is the basis for implementing the basic operation in VQ and FBMA, namely finding the closest match. The block schematic of the associative memory module (AMM) is shown in Figure 5.1. The memory unit consists of $2^b \times M$ identical memory cells $\{B(i,j) : 0 \leq i \leq 2^b - 1, 0 \leq j \leq M - 1\}$, where b is the gray level resolution (e.g 8 for 256 levels) and M is the number of words to be searched in parallel. Each memory cell $B(i,j)$ is a flip-flop with read/write control logic. The words are stored in the memory cells by association: if the jth word is equal to i, where $0 \leq j \leq M$ and $0 \leq i \leq 2^b - 1$, then the memory cell $B(i,j)$ is set to 1, otherwise it is cleared to 0. The storage concept is illustrated in Figure 5.2 for word values 60,62,255, and 59 (M=4 and b=8). The address registers AR1 and AR2 serve to specify the search

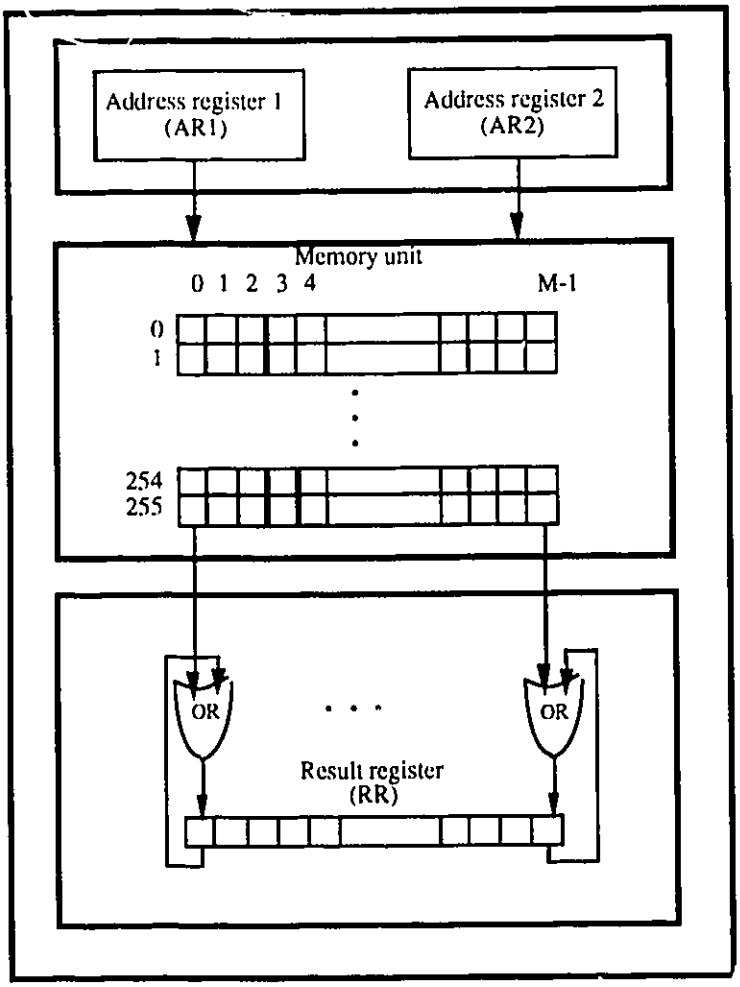


Figure 5.1: Associative memory module.

argument. The result register, RR is used along with AR1, AR2 and the M-OR gates to find the closest word. Here we seek the words C that are within a specified threshold Δ of the search argument V

$$C - \Delta \leq V \leq C + \Delta \tag{5.1}$$

To start with, RR is initialized to 0, the search argument value is loaded into the registers AR1 and AR2. The memory cells are searched to obtain an exact match. This is carried out by executing a read-OR operation. In a read-OR operation, the contents the words addressed by AR1 and AR2 are OR-ed with the contents of RR. Whenever there is a

match, RR(j), the jth bit of RR records a 1 indicating that word j satisfies the search condition. However, if an exact match is not obtained, the contents of AR1/AR2 are incremented/decremented by 1 and the search operation is repeated (within a threshold ± 1). The inexact match search is continued with increasing thresholds until a match is obtained or the input threshold Δ is exceeded. In pseudo code the algorithm can be written as:

PROCEDURE Search within Δ(**INPUT**: Search argument, Words, Threshold Δ;
OUTPUT: Result register)

BEGIN
initialize $RR = 0$;
Initialize $AR1$ = Search argument;
Initialize $AR2$ = Search argument;
Initialize $\delta = 0$;
WHILE ($RR = 0$ **AND** $\Delta \leq \delta$) **DO**
BEGIN
Perform a read-OR operation;
Increment $AR1$ by 1;
Decrement $AR2$ by 1;
Increment δ by 1;
END WHILE;
END.

For example, the steps involved in the search operation within a threshold of $\Delta = 2$ for the search argument value of 64 is illustrated in Figure 5.3. To start with, RR is initialized to 0 and 64 is loaded into the registers AR1 and AR2 and a read-OR operation is executed. The result register RR is 0, so AR1/AR2 is incremented/decremented by 1 and another read-OR operation is performed. Again, RR is 0, so AR1/AR2 is incremented/decremented by 1 and a read-OR operation is executed. Now RR / 0, which indicates that a match is obtained. In this case, the second bit of RR is 1 which implies

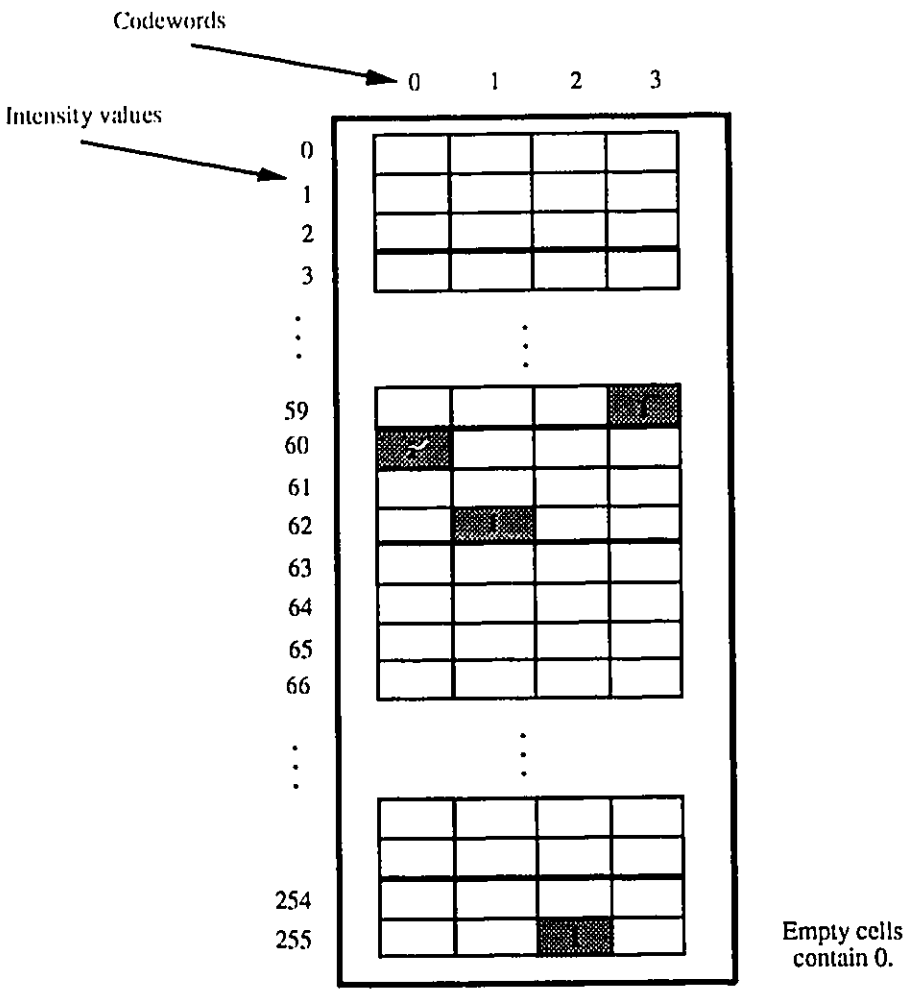


Figure 5.2: Example to illustrate how the codewords are stored in the memory cells.

that the search argument (64) matches the second word (62).

This architecture results in a speed up of $S_p(M)$ where M is the number of words. We note that in order to calculate the effective speed up, we have to factor the number of passes to perform the inexact match (a maximum of Δ passes for an input threshold of Δ). Hence the effective speed up is $S_P(M/\Delta)$. In the following sections, we refer to the AMM as elemental AMM of size $2^b \times M$.

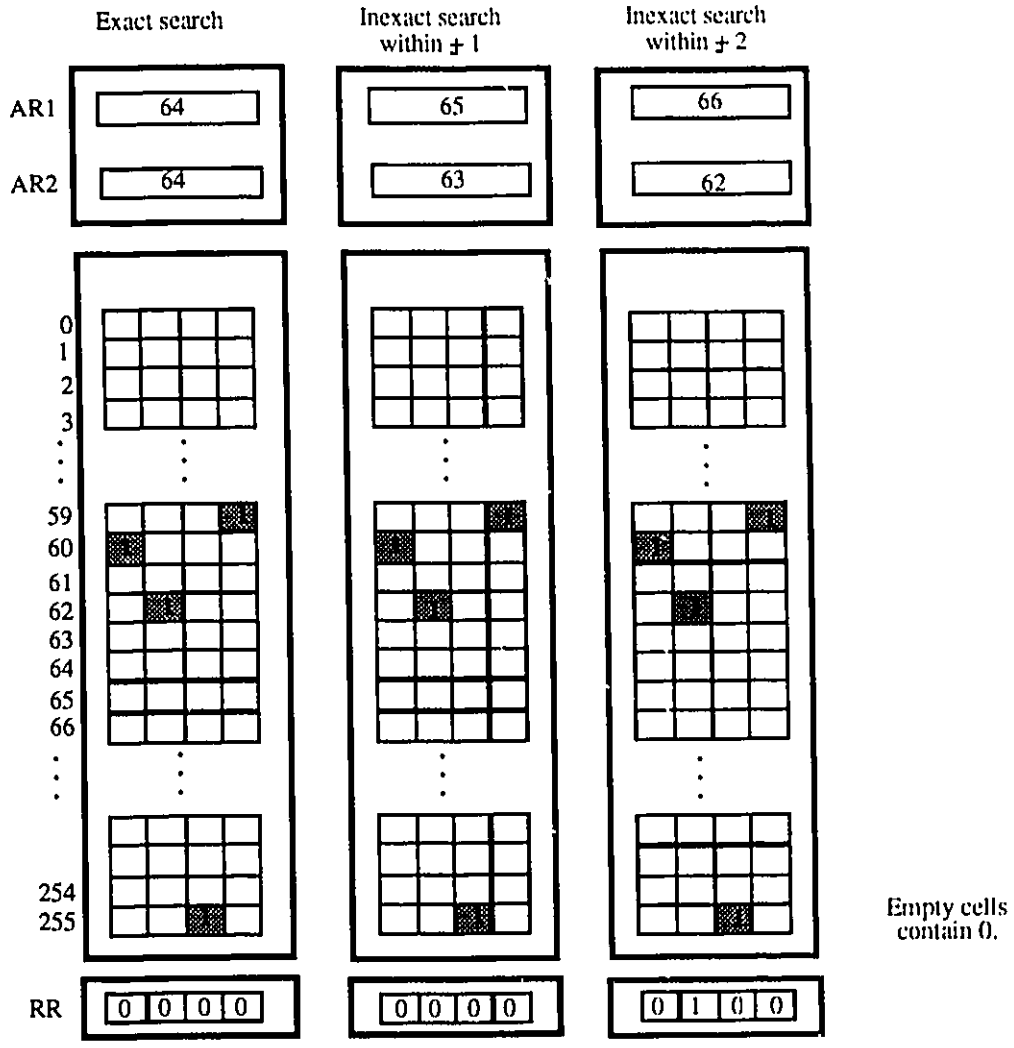


Figure 5.3: The steps involved in the search within $\Delta = 2$ for 64.

5.3 Motion Estimation Using an Associative Memory

We recall from section 2.3.1 that in motion estimation using block matching, a reference block of size $n \times n$ is compared to candidate blocks within a search area SA of size $(n+2p)^2$ in the previous frame, where p is the maximum displacement. The offset between the reference block and the best match candidate block specifies the displacement (motion) vector. In this section we present the implementation of the motion estimation algorithm (modified full search block matching) using the associative memory structure described in section 5.2.

Several block matching algorithms have been reported in the literature [3] [100]. However, these algorithms cannot be directly mapped onto the the associative memory architecture. We propose a modified full search block matching algorithm (FBMA) which can be efficiently mapped onto the associative memory architecture. In FBMA, the reference block is compared with all candidate blocks within a search area SA. For a 2×2 reference block $\{x(i,j) : 0 \leq i \leq 1, 0 \leq j \leq 1\}$ and a maximum displacement $p = 2$, the search area SA $\{y(i,j) : 0 \leq i \leq 5, 0 \leq j \leq 5\}$ is partitioned into 25 overlapped candidate blocks as shown in Figure 5.4. In order to determine the motion vector, the reference block is compared with all the candidate blocks one at a time. We note that in this process the pixels $x(0,0)$, $x(0,1)$, $x(1,0)$ and $x(1,1)$ of the reference block are compared with the pixels $\{y(i,j) : 0 \leq i \leq 4, 0 \leq j \leq 4\}$, $\{y(i,j) : 0 \leq i \leq 4, 1 \leq j \leq 5\}$, $\{y(i,j) : 1 \leq i \leq 5, 0 \leq j \leq 4\}$ and $\{y(i,j) : 1 \leq i \leq 5, 1 \leq j \leq 5\}$ of SA, respectively. Based on this, the FBMA can be executed as follows: A result store $\{s(i,j) : 0 \leq i \leq 5, 0 \leq j \leq 5\}$ is used. The pixel $x(0,0)$ is compared to the pixels $\{y(i,j) : 0 \leq i \leq 4, 0 \leq j \leq 4\}$, if

$$|x(0,0) - y(i,j)| > \Delta \quad 0 \leq i \leq 4, \quad 0 \leq j \leq 4 \tag{5.2}$$

then $s(i,j)$ is reset to 0, where Δ is a prespecified threshold. The result store is then shifted to the right. The previous procedure is repeated for pixels $x(0,1)$, $x(1,0)$ and $x(1,1)$ where

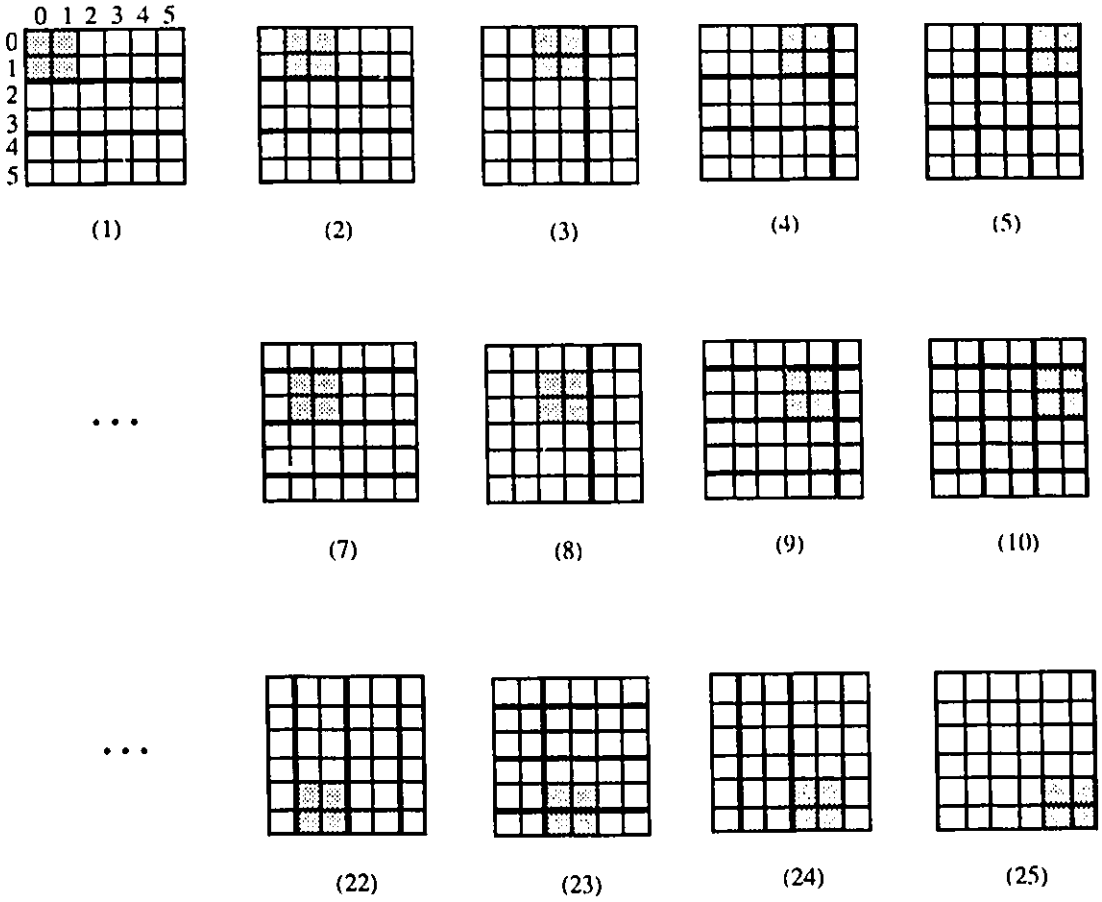


Figure 5.4: Candidate blocks corresponding to a 2×2 reference block and a maximum displacement of 2 pixels.

each pixel is compared with the corresponding pixels of SA and the appropriate shifts are applied on the result store as shown in Figure 5.5. We note that this process is executed in parallel. To determine the motion vector, $s(i,j)$ for $0 \leq i \leq 4$ and $0 \leq j \leq 4$ are examined. If $s(i,j)$ is 1, then the motion vector is (i,j). In Figure 5.5, $s(0,0)$ is 1 which implies that the pixels $x(0,0)$, $x(0,1)$, $x(1,0)$ and $x(1,1)$ of the reference block match the pixels $y(0,0)$, $y(0,1)$, $(1,0)$ and $y(1,1)$ of the search area, respectively. Hence, the reference block in this example matches the first candidate block and the motion vector is (0,0).

We note that a control signal (overhead) is required to select the appropriate pixels

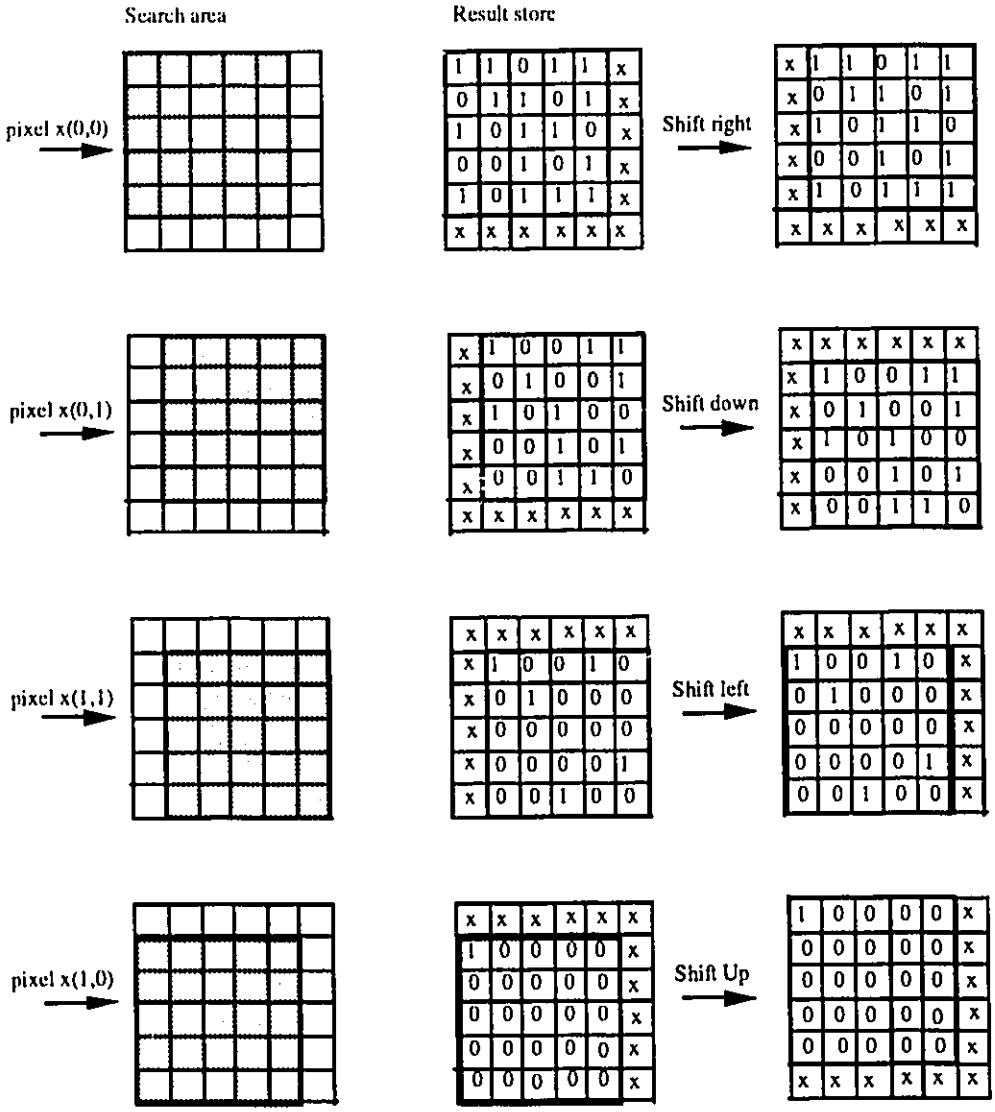


Figure 5.5: The steps required to determine the displacement vector of a 2 × 2 reference block and a maximum displacement of 2 pixels.

of SA. This control overhead can be avoided if the algorithm is modified such that each pixel $x(i,j)$ is compared with the complete search area SA. This may result in some invalid motion vectors which can be ignored. For example, in Figure 5.5 a 1 in $\{s(5,j) : 0 \leq j \leq 5\}$ or in $\{s(i,5) : 0 \leq i \leq 5\}$ corresponds to an invalid motion vector. The regions of valid/invalid motion vectors for a reference block of size $n \times n$ with a maximum displacement p are shown in Figure 5.6. The modified FBMA for motion estimation in pseudo code is given below.

PROCEDURE Modified FBMA (**INPUT**: Reference block(n,n), Search area(m,m), Threshold Δ; **OUTPUT**: Flag, Motion vector)

/* m is the Search area size $(2p + n)^2$ */

BEGIN

Initialize Result store(m,m) to 1's;

Flag = FALSE;

FOR $i = 0$ **TO** $n - 1$ **DO**

FOR $j = 0$ **TO** $n - 1$ **DO**

BEGIN

FOR $k = 0$ **TO** $2 \times p + n - 1$ **DO**

FOR $l = 0$ **TO** $2 \times p + n - 1$ **DO**

BEGIN

IF Reference block(i,j) - Search area(k,l) ¿ Δ **THEN**

/* If a match is not obtained then reset the Result store to 0. */

/* Note that the shift operation is implicitly implemented.*/

IF $k - i \geq 0$ **AND** $l - j \geq 0$ **THEN**

Result store$(k - i, l - j) = 0$;

END FOR;

END FOR;

FOR $i = 0$ **TO** $2 \times p$ **DO**

FOR $j = 0$ **TO** $2 \times p$ **DO**

IF Result store(i,j) is 1 **THEN**

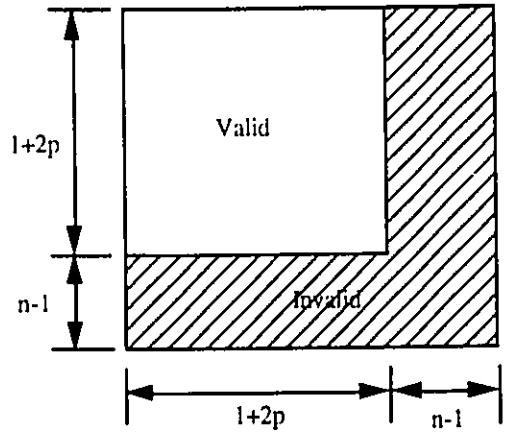


Figure 5.6: The regions of valid and invalid displacement vectors.

BEGIN

Motion vector = (i, j);

Flag = TRUE;

exit;

END IF;

END.

The comparison of one pixel of a reference block with the pixels of a search area is implemented by setting the search area in a row fashion as words and the pixel of the reference block as the search argument. The result of the comparison operation is obtained by examining the contents of the result register (RR). If every bit of RR is 0 then no match is obtained. Otherwise (RR $\neq$ 0), a match is obtained and the non-zero bit position(s) of RR correspond to the matched pixel(s). To exploit parallelism in the direction of the reference block $L = n \times n$ elemental AMM's of size $2^b \times (n + 2p)^2$ are required. For a 2×2 reference block $\{x(i,j) : 0 \leq i \leq 1, 0 \leq j \leq 1\}$ with a maximum displacement of 2 pixels and 256 gray levels, 4 elemental AMM's of size 256×36 are required as shown in Figure 5.7. The complete search area SA is stored in every AMM. The pixels $x(0,0)$, $x(0,1)$, $x(1,0)$ and $x(1,1)$ are set as the search argument of the first,

second, third and the fourth elemental AMM, respectively. The elemental AMM's are searched in parallel and the partial results are then combined.

The motion vector generation module combines the results obtained by the elemental AMM's using m AND-gates as shown in Figure 5.8. The result of the combination operation is stored in the displacement register which is then examined by the motion vector generator to determine if a match is obtained or not. If a match is obtained (DR $\neq 0$), the corresponding motion vector is generated. In pseudo code, the modified FBMA using the associative memory can be written as

PROCEDURE AM modified FBMA (**INPUT:** Reference block(n,n), Search area(m,m), Threshold Δ;**OUTPUT**: Flag, Motion vector)

/* m is the search area size $(2p+n)^2$ */

BEGIN

Initialize words=search area;

Initialize Result register $RR_i = 0$ for $i = 0 \cdots L-1$

Initialize $AR1_{i+j\times n}$ = Reference block(i,j) for $i = 0 \cdots n$;

Initialize $AR2_{i+j\times n}$ = Reference block(i,j) for $i = 0 \cdots n$;

$\delta = 0$;

Flag = FALSE;

WHILE Flag is FALSE **AND** $\delta \leq \Delta$ **DO**

BEGIN

FOR all elemental AMM in parallel $i = 0 \cdots L-1$ **DO**

BEGIN

Perform a read-OR operation;

Increment $AR1_i$ by 1;

Decrement $AR2_i$ by 1;

Increment δ by 1;

END FOR

IF Displacement register $\neq 0$ **THEN**

BEGIN

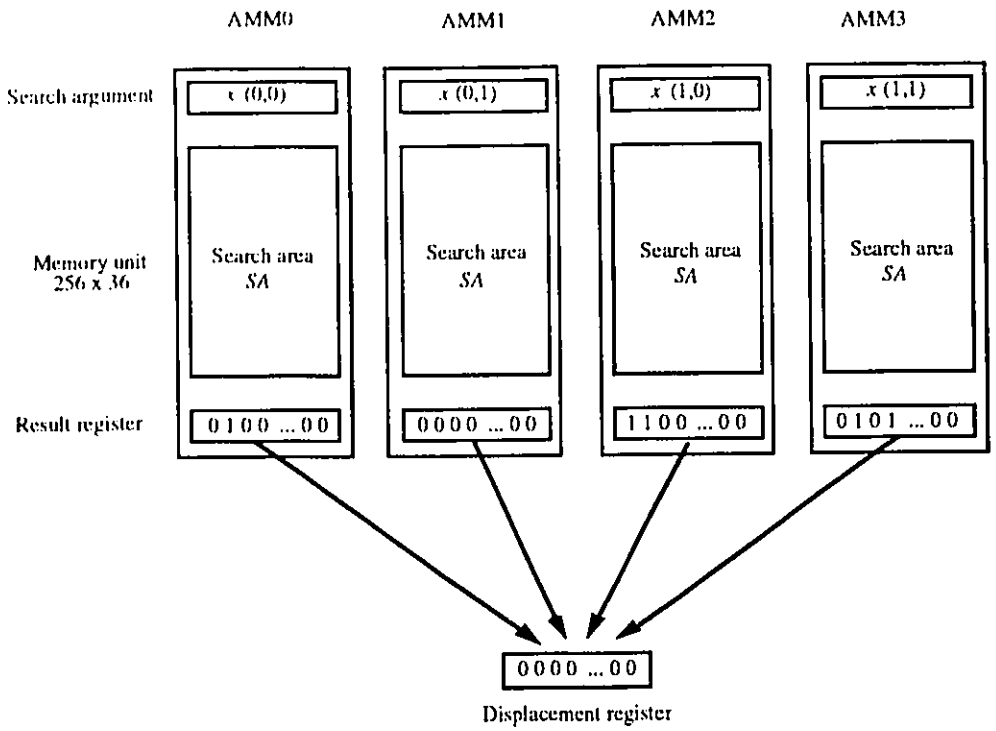


Figure 5.7: Associative memory structure for a 2×2 reference block and a maximum displacement of 2 pixels.

```
            Flag TRUE;
            Generate the corresponding motion vector;
        END IF
    END WHILE;
END.
```

Thus, the parallelism in the directions of search area and the vector dimension are exploited which results in an effective speed up of $S_p(L(2p+1)^2/\Delta)$.

5.4 VQ Using an Associative Memory

We recall from section 2.2.4 that in VQ, the encoding of an input vector essentially involves comparing the input vector with a set of codewords. The closest codeword is found and

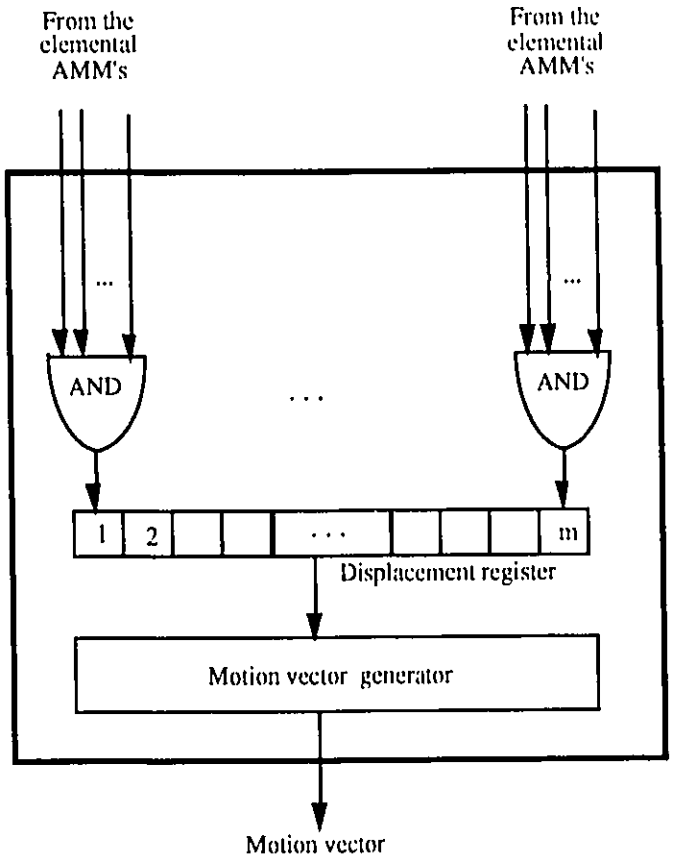


Figure 5.8: Motion vector generation module

the corresponding label is used to represent the vector. We now show how VQ can be implemented using the associative memory structure described in section 5.2.

VQ for one-dimensional vectors ($L = 1$) is implemented using a single AMM module where the codewords are set as words and the input vector is set as the search argument. For each input vector, the codewords are searched in parallel. The result of the search within Δ is obtained by examining the contents of RR. If every bit of RR is 0, then no match is obtained. Otherwise (RR $\neq$ 0), a match is obtained and the non-zero bit position(s) of RR corresponds to the label of the matched codeword(s). For the case of L-dimensional vectors, L elemental AMM's of size $2^b \times N$ are required, where N is the codebook size. To determine the closest codeword, the result registers of the elemental AMM's are combined. If all result registers record a match then the corresponding label is generated using the label generation module shown in Figure 5.9. Note that the label generator can be designed to resolve multiple matches according to certain priority. For example, the input vector may be assigned to the first matched codeword. The effective speed up is $S_p(LN/\Delta)$ as the search operation is executed in parallel in the directions of L and N. VQ using the associative memory in pseudo code is given below

PROCEDURE VQ AM (**INPUT:** Input vector, Codewords, Threshold Δ;
OUTPUT: Flag, Label)

BEGIN

Initialize Result register $RR_i = 0$ for $i = 0 \cdots L-1$

Initialize $AR1_i$ input vector(i) for $i = 0 \cdots n$;

Initialize $AR2_i$ input vector(i) for $i = 0 \cdots n$;

δ 0;

Flag FALSE;

WHILE Flag is FALSE **AND** $\delta \leq \Delta$ **DO**

BEGIN

FOR all elemental AMM in parallel $i = 0 \cdots L-1$ **DO**

BEGIN

Perform a read-OR operation;

Increment $AR1_i$ by 1;

Decrement $AR2_i$ by 1;

Increment δ by 1;

END FOR

IF Label register $\neq$ 0 **THEN**

BEGIN

Flag=TRUE;

Generate the corresponding Label;

END IF

END WHILE;

END.

5.4.1 Codeword Usage Module

We recall from chapter 4 that in VC-FAVQ the primary and secondary codebooks are organized using the LRU algorithm. We propose a simple and efficient implementation of the LRU Approximation Algorithm[104]. Here, a usage bit UB_i associated with each word i and a removal pointer (RP) are used to find the LRU codeword. Whenever the ith codeword is referenced its usage bit UB_i is set to 1. To determine the index of the LRU codeword, the usage bits are examined in sequence. If UB_i is 1, it is reset to 0. Otherwise ($UB_i = 0$), then we know that codeword i has not been referenced since the last time UB_i was reset to 0. Figure 5.10 demonstrates the LRU approximation scheme for a codebook of size $N = 8$. In Figure 5.10a codeword 1 was the last codeword appended to the codebook. In order to select a codeword for replacement, the usage bits are examined starting at UB_2 as shown in Figure 5.10b. Since UB_2 is 1 (codeword 2 has been referenced since the last time UB_2 was reset to 0), it is reset to 0 and RP is incremented by 1. In the second step, UB_3 is tested. However, codeword 3 has been referenced ($UB_3 = 1$), therefore UB_3 is reset to 0 and RP is incremented. In the last step, codeword 4 is selected for replacement since UB_4 is 0. We now describe the procedure to find the LRU codeword:

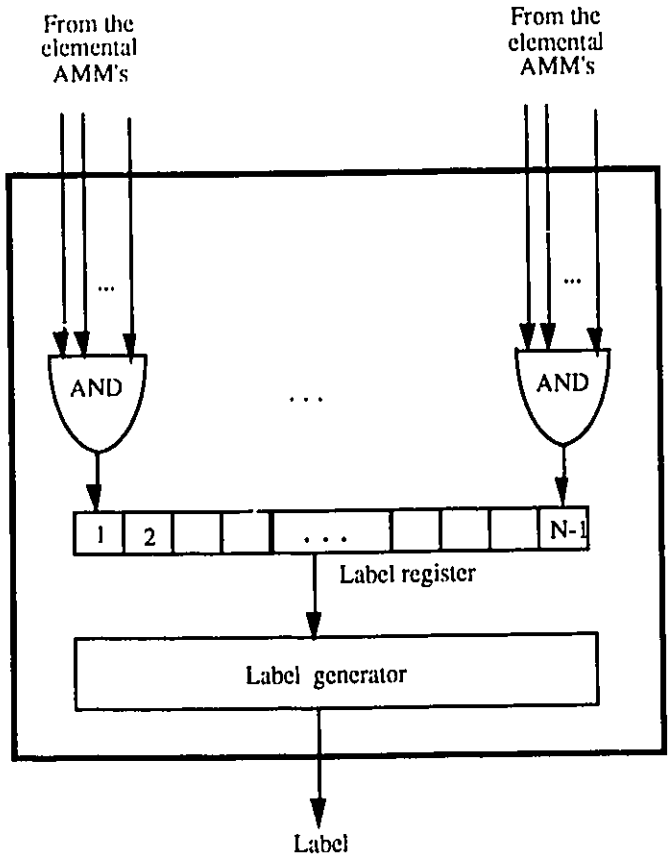


Figure 5.9: Label generation module

PROCEDURE LRU approximation (**INPUT**: $RP(t)$, $UB(t)$;
OUTPUT: $RP(t+1)$, $UB(t+1)$, LRU Codeword Index)

BEGIN

Flag = FALSE;

WHILE (Flag is FALSE) **DO**

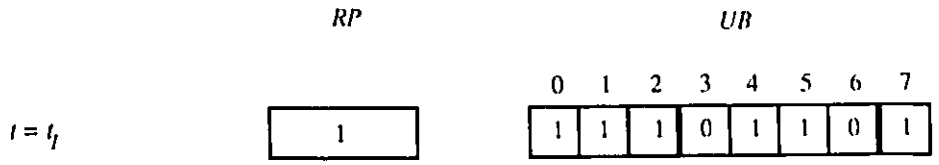
BEGIN

Increment $RP(t)$ by 1;

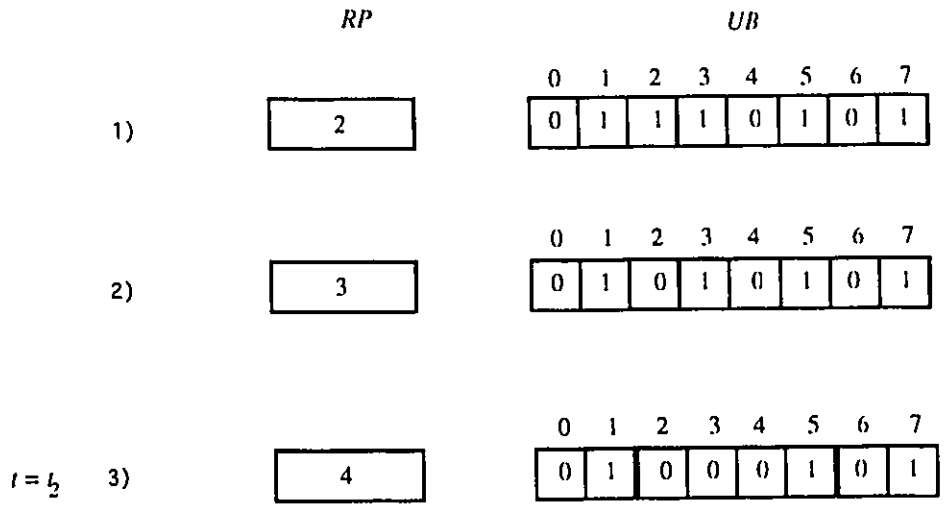
$i = RP(t)$;

IF the codeword has been referenced ($UB_i(t) = 1$)

Set the usage bit of this codeword to 0 ($UB_i(t) = 0$);



a) RP and UB before executing the LRU approximation algorithm.



b) The steps required to determine the LRU codeworc

Figure 5.10: Illustration of the LRU approximation algorithm.

ELSE

BEGIN

LRU Codeword Index=i;

Flag=TRUE;

END ELSE;

END WHILE;

$RP(t+1) = RP(t)$;

$UB(t+1) = UB(t)$;

END.

Here, t refers to the time instance before the execution of the LRU approximation algorithm. The contents of RR are used to set UB_i. We note that given $UB(t)$ and $RP(t)$; $UB(t+1)$, $RP(t+1)$ and the LRU codeword index can be determined. In other words, the LRU Approximation Algorithm can be implemented using a finite state machine as shown in Figure 5.11.

5.5 Associative Memory Architecture for Video Compression

We recall from chapter 4 that in VC-FAVQ an input vector is first encoded in the interframe mode, if it fails the input vector is encoded in the intraframe mode. In the interframe mode a match is sought within a search area in the previous frame. If a match is obtained the corresponding motion vector is used to represent the input vector. In the intraframe mode, the input vector is compared with a set of codewords in a primary codebook. If a match is not obtained in the primary, the input vector is also compared with the codewords in a secondary codebook. If a match is obtained, the input vector is represented by the corresponding label. Otherwise, the input vector is transmitted. The primary and secondary codebooks are organized using the LRU algorithm.

In this section we present the associative memory design for VC-FAVQ which is a

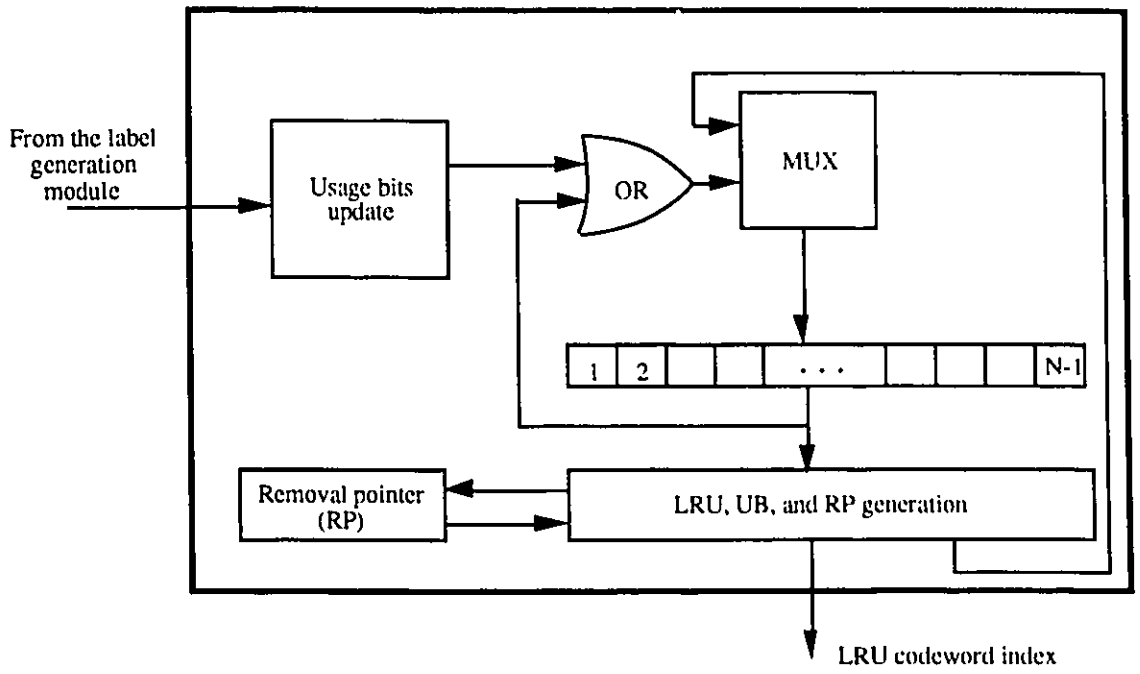


Figure 5.11: Codeword usage module.

combination of the architecture designed for motion estimation and VQ in sections 5.3 and 5.4, respectively. The block schematic of the general associative memory design for VC-FAVQ is shown in Figure 5.12. The design of the individual modules follows.

Motion Estimation Module: To estimate the motion of a reference block of size $L = n \times n$, the motion estimation module is composed of L elemental AMM of size $2^b \times (n + 2p)^2$ and a high speed interface, where b is the gray level resolution and p is the maximum displacement. The high speed interface is required to load the search area and the reference block in parallel into the memory cells and the address registers of the elemental AMM's, respectively.

Motion Vector Generation Module: The structure of the motion vector generation module is shown in Figure 5.8. It combines the search results from the elemental AMM of the motion estimation module into the displacement register using $n \cdot (n + 2p)^2$ AND-gates. The contents of the displacement register is used to test if a match is obtained or not. In case of a match is obtained the motion vector generator generates the

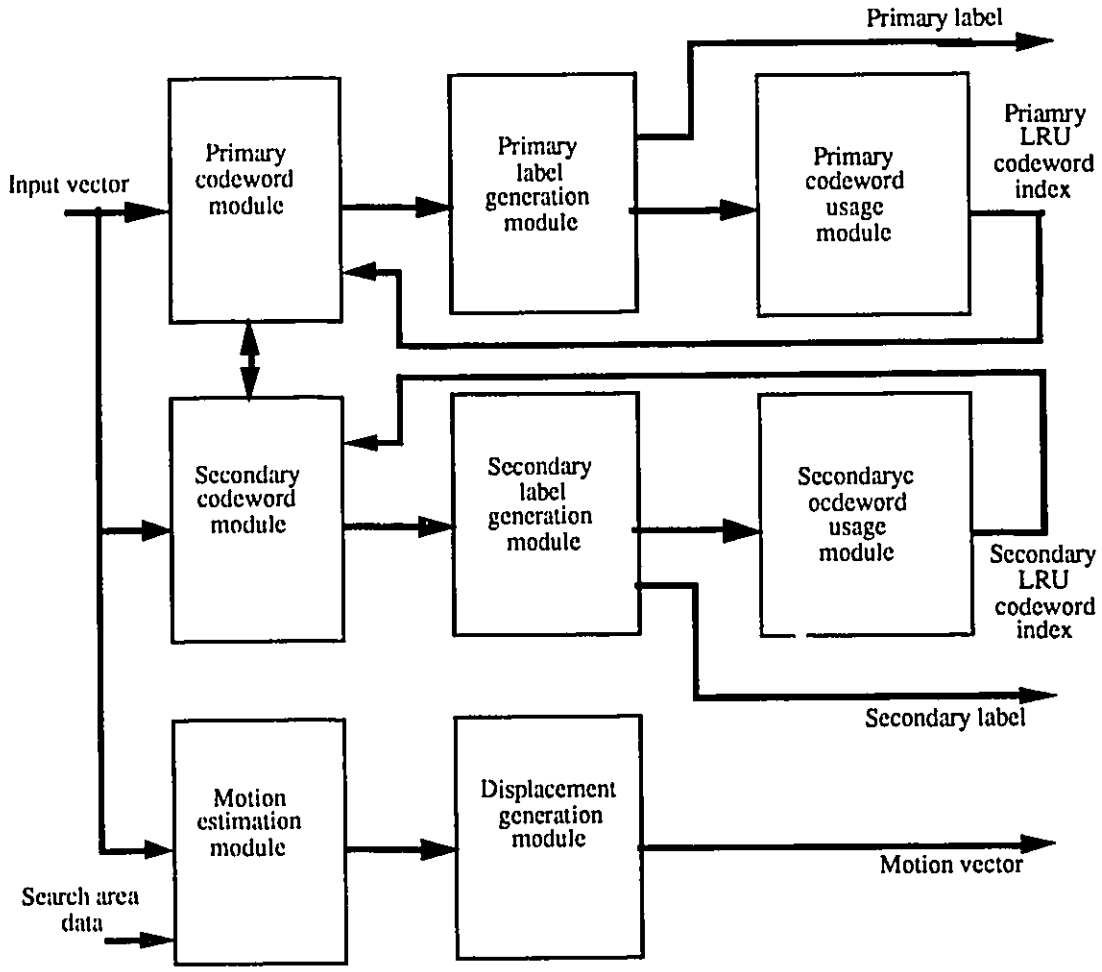


Figure 5.12: Block diagram of the Associative memory architecture.

corresponding motion vector.

Primary Codeword Module: The primary codeword module consists of L elemental AMM of size $2^b \times N_p$, where N_p is the primary codebook size. It also contains a high speed interface to load the primary codewords and the input vector into the associative memory cells and the address registers. The load operation is executed in parallel.

Primary Label Generation Module: The design of the primary label generation module is shown in Figure 5.9 for $N_p = N$ codewords. It combines the results obtained by the elemental AMM's of the primary codeword module using N_p AND-gates. The label register is used to store the result of the combination operation. The label generator examines the contents of the label register if a match is obtained (label register $\neq$ 0), then it generates the corresponding primary label.

Primary Codeword Usage Module: This module is required to keep track of the usage of the primary codewords. The structure of the primary codeword usage module is shown in Figure 5.11 for $N_p = N$ codewords (section 5.4.1).

Secondary Codeword Module: The structure of the secondary codeword module is similar to that of the primary codeword module. However, the size of the elemental AMM's is $2^b \times N_s$, where N_s is the size of the secondary codebook.

Secondary Label Generation Module: The design of the secondary label generation module is similar to that of the primary label generation module except it is designed for N_s codewords rather than N_p codewords.

Primary Codeword Usage Module: The structure of this module is similar to that of the primary codeword usage module. However it size is different as it should keep track for N_s codewords.

5.6 Execution Time

We recall from section 4.4 that the encoding complexity of VC-FAVQ for an input vector ($K = 1$) using a primary codebook of size N_p and a secondary codebook of size N_s is

given by:

$$O\left\{L\left(\eta_1(2p+1)^2+\eta_2(N_p+(2p+1)^2)+(1-\eta_1-\eta_2)(N_s+N_p+(2p+1)^2)\right)\right\} \quad (5.3)$$

where η_1 and η_2 are the probabilities of obtaining a match in the search area and in the primary codebook, respectively. Hence, the execution time T_{SISD}, using an SISD implementation for $K = 1$ is:

$$T_{SISD} = T_e \times L\left\{\eta_1(2p+1)^2+\eta_2\left(N_p+(2p+1)^2\right)+ \quad (5.4)$$

$$(1-\eta_1-\eta_2)\left(N_s+N_p+(2p+1)^2\right)\right\} \quad (5.5)$$

T_e is the time of execution of the elemental distortion operation, The associative memory architecture exploits parallelism in the directions of vector dimension, codewords, and the search area. The speed up factors of the associative architecture are as follows:

- $S_p\left(\eta_1(2p+1)^2+\eta_2(N_p+(2p+1)^2)+(1-\eta_1-\eta_2)(N_s+N_p+(2p+1)^2)\right)$ due to the parallelism in the direction of codewords and search area.
- $S_p(L)$ due to the parallelism in the direction of vector dimension.
- $S_p(T_e/T_a)$, where T_a is the execution time of the elemental operation using the associative memory, result from faster execution of the elemental operation. The elemental operation in the associative memory architecture is a logic operation, while on SISD machine the elemental operation is arithmetic operation.
- $S_p(1/(3n_i))$ where n_i is the number of passes required to execute the "Search within Δ" algorithm. Each pass requires 3 clock cycles (two clock cycles to read the memory location addressed by AR1 and AR2 and one clock cycle to increment/decrement AR1/AR2). The maximum value of n_i is equal to the prespecified threshold Δ.

Thus the maximum *total speed up* achieved is

$$S_p\left(LT_e \times \frac{\eta_1(2p+1)^2+\eta_2(N_p+(2p+1)^2)+(1-\eta_1-\eta_2)(N_s+N_p+(2p+1)^2)}{3T_a n_i}\right) \quad (5.6)$$

Hence, the execution time T_A, for a single frame of K vectors is given by:

$$T_A = T_{SISD}/(total\ speed\ up) \tag{5.7}$$

$$= T_a \times \sum_{i=1}^{K} 3n_i \tag{5.8}$$

We recall from chapter 4 that typical values of the threshold Δ and the vector dimension L are 12 and 16, respectively. With the present technology T_a is in the order of 30 ns [108]. Hence, the maximum execution time for 512×512 and 288×360 frames are 18 and 7 ms, respectively.

We note that the execution time does not depend on the vector dimension, the codebook size, or the search area size. However, a larger vector dimension, codebook, or search area requires more hardware.

5.7 Summary

In this chapter, we have presented an associative memory architecture for real-time video compression. VQ and motion estimation involves a search operation on the codebook and the search area, respectively, to obtain the best match. The search procedure is computationally intensive making real time implementation difficult. The proposed architecture exploits parallelism in the directions of L, the codewords, and the search area resulting in an execution time of $T_a \times \sum_{i=1}^{K} 3n_i$. The simplicity, partitionability (in the direction of L, N, and b), and the modularity of the proposed architecture makes possible VLSI implementation.

Chapter 6

Summary and Future Work

6.1 Summary

In this thesis, a new adaptive algorithm (VC-FAVQ) and a novel architecture for video compression have been presented. The proposed algorithm for video compression combines the concepts of motion estimation (interframe) and adaptive VQ (intraframe) resulting in an excellent coding performance at a reduced complexity. In the interframe mode the input vector is compared with the vectors within a search area in the previous frame. If a match is obtained the corresponding motion vector is used to represent the input vector. In the intraframe mode a match is sought in a primary codebook. If a match is not obtained in the primary, the input vector is also compared with the codewords in a secondary codebook. If a match is obtained the input vector is represented by the corresponding label. Otherwise, the input vector is transmitted. The primary and secondary codebooks are dynamically generated and are organized using the LRU algorithm. Computer simulations demonstrate the superior coding performance compared to other techniques. Excellent quality images (PSNR of 39 dB) were obtained at compression ratio of 13:1. In addition, this technique is simple to implement, highly adaptive and is elegant for constant quality applications.

VQ and motion estimation involves a search operation on the codebook and the search area, respectively, to obtain the best match. The search procedure is computa-

tionally intensive making real time implementation difficult. We have proposed a novel storage concept where image data is stored by association rather than by contents. We have also proposed a design for an LRU algorithm. Based on this, we presented an unified associative memory architecture for real-time implementation of motion estimation and frame adaptive vector quantization for video compression. The proposed architecture exploits parallelism in the directions of vector dimension, the codewords, and the search area resulting in a high speed up. The simplicity, partitionability (in the directions of vector dimension, codewords, and the gray level), and the modularity of the proposed architecture make possible VLSI implementation.

6.2 Future Research Work

Video sequences are highly non-stationary and they exhibit changes from frame to frame and from scene to scene. Moreover, the local activities within the same frame may vary significantly. Hence, the use of a preepecified threshold to code all the input vectors is not optimum. It is possible to achieve a better coding performance by using a variable threshold. For example, the threshold value may be varied according to the variance of each input vector. Another possibility is the use of a hybrid codebook. The hybrid codebook is composed of two sub-codebooks, one is generated on the fly and the other is fixed. This should lower the bit rate somewhat.

Future work in the area of the proposed associative memory architecture could concentrate on implementing motion estimation and adaptive VQ in the same associative memory module. This is possible because both the codewords and the search area are parts of the previous frame. Hence, the previous frame can be stored in the associative memory and the search operation can be carried out. This will eliminate the need for high speed interfaces to load the search area in the associative memory modules and avoid the possibility of a communication bottleneck. The VLSI implementation of the associative memory architecture is also a possible future area of work.

Bibliography

[1] A. R. Netravali, "On Quantizers for DPCM Coding of Picture Signals", *IEEE Transactions on Information Theory*, Vol. IT 23, No. 3, pp. 360-370, May 1977.

[2] A. K. Jain, " Image Data Compression: A Review", *Proceedings of the IEEE*, Vol. 69, No. 3, pp. 349-389, March 1981.

[3] J. R. Jain and A. K. Jain, "Displacement Measurement and Its Application in Interframe Image Coding", *IEEE Trans. on Communications*, Vol. COM-29, No. 12, pp. 1799-1808, December 1981.

[4] A. N. Netravali and J. O. Limb, " Transform Picture Coding", *Proceedings of the IEEE*, Vol. 68, No. 3, pp. 366-407, March 1980.

[5] K. R. Rao and P. Yip, "Discrete Cosine Transform Algorithms, Advantages, and Applications", *Academic Press*, Second edition 1990.

[6] W. K. Pratt, "Digital Image Processing", *Academic Press*, 1990.

[7] J. A. Roese, W. K. Pratt, and G. S. Robinson, "Interframe Cosine Transform Image Coding", *IEEE Trans. on Communications*, Vol. COM-25, No. 11, pp. 1329-1338, November 1977.

[8] W. H. Chen and C. H. Smith, "Adaptive Coding of Monochrome and Color Images", *IEEE Trans. on Communications*, Vol. COM-25, No. 11, pp. 1285-1292, November 1977.

[9] W. K. Pratt, "A Comparison of Digital Image Transforms", *Proc. Mervin J. Kelly Commun. Conf.*, Vol. COM-25, No. 11, pp. 17-4-1-17-4-5,1970.

[10] P. A. Wintz, " Transform Picture Coding", *Proceedings of the IEEE*, Vol. 60, No. 7, pp. 809-820, July 1972.

[11] A. Habibi and P. A. Wintz, "Image Coding by Linear Transformation and Block Quantization", *IEEE Trans. on Communications*, Vol. COM-19, No. 1, pp. 50-62, February 1971.

[12] A. N. Netravali and J. D. Robbins, "Motion Compensated Television Coding:Part I", *Bell Syst. Tech. J.*, Vol. 58, pp. 631-670, March 1979.

[13] J. C. Candy, M. A. Franke, M. A. Haskel, and F. W. Mounts, "Transmitting television as clustters of frame to frame differences", *Bell Syst. Tech. J.*, Vol. 50, No. 6 pp. 1889-1917, 1971.

[14] R. F. W. Pease and J. O. Limb, "Exchange of Spatial and Temporal Resolution in Television Coding", *Bell Syst. Tech. J.*, Vol. 50, pp. 191-200, January 1971.

[15] A. Habibi, "Survey of Adaptive Image Coding Techniques", *IEEE Trans. on Communications*, Vol. COM-25, No. 11, pp. 1275-1284, November 1977.

[16] R. Gray, "Source Coding Theory", *Kluwer Academic Publishers*, 1990.

[17] N. S. Jayan, "Digital Coding of Wave Forms Principles and Applications to Speech and Video",*Prentice Hall*, 1984.

[18] J. Makhoul, S. Roucos, and H. Gish, "Vector Quantization in Speech Coding",*Proceedings of the IEEE*,Vol. 73, No. 11, pp. , November 1985.

[19] L. D. Davisson, " Rate-Distortion Theory and Application", *Proceedings of the IEEE*, Vol. 60, No. 7, pp. 800-808, July 1972.

[20] S. P. Lloyd, "Least Squares Quantization PCM", *IEEE Transactions on Information Theory*, Vol. IT-28, No. 2, pp. 129-137, March 1982.

[21] R. Gray, "Vector Quantization", *IEEE ASSP Mag.*, Vol. 1, pp. 4-29, April 1984.

[22] N.M. Nasrabadi and R. A. King, "Image Coding Using Vector Quantization: A Review", *IEEE Trans. on Communications*, Vol. COM-36, No. 8, pp. 957-971, August 1988.

[23] Y. Linde, A. Buzo, and R. Gray, "An Algorithm for Vector Quantizer Design", *IEEE Trans. on Communications*, Vol. COM-28, pp. 84-95, January 1980.

[24] Y. He, Q. Zhang, Y. Ye, and Z. Li,"Vector Quantization by Neural Network",*SPIE Medical Imaging IV: Image Capture and Display*, Vol. 1232, pp. 359-362, 1990.

[25] J. Li and C. N. Manikopoulos,"Multi Stage Vector Quantization based on the Feature Maps",*SPIE Visual communications and Image Processing IV*, Vol. 1199, pp. 1046-1055, November 1989.

[26] J. Vaisey and A. GershoN, "Simulated Annealing and Codebook Design", *IEEE 1988 International Conference on Acoustics, Speech, and Signal Processing*, Vol. 2, pp. 1176-1179, April 1988.

[27] J. K. Flanagan and D. R. Morrell, "Vector Quantization Codebook Generation using Simulated Annealing", *IEEE 1988 International Conference on Acoustics, Speech, and Signal Processing*, pp. 1759-1762, 1988.

[28] W. H. Equitz, "Interframe Hierarchical Vector Quantization", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 37, No. 10, pp. 1568-1575, October 1989.

[29] A. Gersho and B. Ramamurthi, "Image Coding using Vector Quantization", *IEEE 1982 International Conference on Acoustics, Speech, and Signal Processing*, pp. 428-431, May 1982.

[30] S. Gupta and A. Gersho,"Image Vector Quantization with Block Adaptive Scalar Prediction", *SPIE Visual Communications and Image Processing: Visual Communications*, Vol. 1605, pp. 179-189, 1991.

[31] C. W. Rutledge, "Variable Block DPCM: Vector Predictive of Color Images", *Proceedings of the IEEE* , Vol. 1, pp. 130-135, June 1987.

[32] B. Ramamurthi and A. Gersh, "Classified Vector Quantization of Images", *IEEE Trans. on Communications*, Vol. COM-34, No. 11, pp. 1105-1115, November 1986.

[33] D. S. Kim and S. U. Lee,"Classified Vector Quantizer based On Minimum Distance Partitioning", *SPIE Visual Communications and Image Processing: Visual Communications*, Vol. 1605, pp. 190-201, 1991.

[34] D. S. Kim, and S. U. Lee, "Image Vector Quantization based on Classification in the DCT Domain", *IEEE Trans. on Communications*, Vol. 39, No. 4, pp. 549-556, April 1991.

[35] J. W. Kim, and S. U. Lee, "A Transform Domain Classified Vector Quantization for Image Coding", *IEEE Trans. on Circuits and Systems*, Vol. 2, No. 1, pp. 549-556, March 1992.

[36] J. Y. Nam and K. R. Rao, "Image Coding Using a classified DCT/VQ Based on Two-Channel Conjugate Vector Quantization ", *IEEE Trans. on Circuits and Systems for Video Technology*, Vol. 1, No. 4, pp. 325-336, December 1991.

[37] I. Dinstein, K. Rose and A. Heiman, "Variable Block Size Transform Image Coder", *IEEE Trans. on Communications*, Vol. 38, No. 11, pp. 2073-2078, November 1990.

[38] N. M. Nasrabadi and Y. Feng, "A Dynamic Finite State Vector Quantization Scheme", *IEEE 1990 International Conference on Acoustics, Speech, and Signal Processing* , pp. 2261-2264, April 1990.

[39] T. Kim, "Side Match and Overlap Match Vector Quantizer for Images", *IEEE Transactions on Image Processing*. Vol. 1, No. 2, pp. 170-185, April 1992.

[40] A. Gersho and M. Yano, "Adaptive Vector Quantization by Progressive Codevector Replacement", *IEEE 1982 International Conference on Acoustics, Speech, and Signal Processing*, pp. 133-136, March 1985.

[41] M. Goldberg, P. R. Boucher and S. Shlien, "Adaptive Image Compression Using Vector Quantization", *IEEE Trans. on Communications*, Vol. COM-34, No. 8, pp. 180-187, February 1986.

[42] S. Panchanathan and M. Goldberg, "Adaptive Algorithms for Image Coding Using Vector Quantization", *Signal Processing: Image Communication*, Elesvier Science Publishers, pp. 81-92, 1991.

[43] K. Zeger and A. Bist, "Universal Adaptive Vector Quantization with Application to Image Compression", *IEEE 1992*, Vol. COM-34, No. 8, pp. III-381 - III-384, 1992.

[44] A. Buzo, A. H. Gray, R. M. Gray and J. D. Markel, "Speech Coding based upon Vector Quantization", *IEEE Trans. on ASSP*, Vol. ASSP-28, pp. 562-574, October 1980.

[45] E. A. Riskin and R. M. Gray, "A Greedy Tree Growing Algorithm for the Design of Variable Rate Vector Quantizers", *IEEE Trans. on Signal Processing*, Vol. 39, No. 11, pp. 2500-2507, November 1991.

[46] V. Ramasubramanian and K. Paliwal, "Fast K-Dimensional Tree Algorithms for Nearest Neighbour Search with Applications to Vector Quantization", *IEEE Trans. on Signal Processing*, Vol. 40, No. 3, pp. 518-531, March 1992.

[47] B. H. Juang and A. H. Gray, "Multiple Stage Vector Quantization for Speech Coding", *IEEE 1982 International Conference on Acoustics, Speech, and Signal Processing*, pp. 597-600, April 1985.

[48] L. Wang and M. Goldberg, "Block Transform Image Coding by Multistage Vector Quantization with Optimal Bit Allocation", *IEEE Transactions on Communications*, Vol. 39, No. 9, pp. 1360-1369, September 1991.

[49] F. Kossentini , M. J. Smith and C. F. Barnes, "Image Coding with Variable Rate RVQ", *IEEE 1992 International Conference on Acoustics, Speech, and Signal Processing*, Vol. 40, No. 11, pp. III-369 - III-372, 1992.

[50] W. Y. Chan, S. Gupta, and A. Gersho, "Enhanced Multistage Vector Quantization by Joint Codebook Design", *IEEE Transactions on Communications*, Vol. 40, No. 11, pp. 1693-1697, November 1992.

[51] C. L. Yeh, "Image Compression Using Non adaptive Vector Quantization with Adaptive Overhead Transmission for Codevector Replacement", *Proc. GLOBECOM 87*, pp. 35.7.5-35.7.5, 1987.

[52] S. Panchanathan and M. Goldberg, "A Mini-Max Algorithm for Image Adaptive Vector Quantization", *IEE Proceedings : Part I - Communications, Speech and Vision*, Vol. 138, No. 1, pp. 53-60, February 1991.

[53] N.M. Nasrabadi, Y. Feng, "Image Compression Using Address-Vector Quantization", *IEEE Trans. on Communications*, Vol. 38, No. 12, pp. 2166-2173, December 1990.

[54] N.M. Nasrabadi, Y. Feng, "A Multilayer Address-Vector Quantization Technique", *IEEE Trans. on Circuits and Systems*, Vol. 37, No. 7, pp. 912-921, July 1990.

[55] N.M. Nasrabadi, S. E. Lin, and Y. Feng, "Interframe Hierarchical Vector Quantization", *IEEE 1989 International Conference on Acoustics, Speech, and Signal Processing*, May 1989.

[56] J. S. Vitter, "Two Papers on Dynamic Huffman Codes", *The 26th Annual IEEE Symposium on Foundations of Computer Science*, October 1985.

[57] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic Coding",*Communications of the ACM*,Vol. 30, No. 6, pp. 520-540, June 1987.

[58] T. Murakami, K. Asai, and E. Yamazaki, "Image Sequence Coding", *Electronics Letters*, Vol. 18, No. 23, pp. 1005-1006, November1982.

[59] L. Corte-real and A. P. Alves, "Vector Quantization of Image Sequences Using Variable Size and Variable Shape Blocks", *Electronics Letters*, Vol. 26, No. 18, pp. 1483-1484, August 1990.

[60] J. Huguet and L. Torres, "Vector Quantization in Image Sequence Coding", *Signal Processing V: Theories and Applications*, pp. 1079-1081, 1990.

[61] W. T. Chen, R. F. Chang, and J. S. Wang, "Image Sequence Coding Using Adaptive Finite-State Vector Quantization", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 2, No. 1, pp. 15-24, March 1992.

[62] Q. Guo, N. M. Nasrabadi, and N. Mohesenian, "An Interframe Dynamic FSVQ Codec for Video Sequence Coding", *IEEE 1992 International Conference on Acoustics, Speech, and Signal Processing*, pp. 501-504, 1992.

[63] N.M. Nasrabadi,"Interframe Hierarchical Address-Vector Quantization", *Visual Communications and Image Processing: SPIE'90*, Vol. 1360, pp. 558-574, 1990.

[64] N.M. Nasrabadi, C. Y. Choo, and J. U. Roy, "Interframe Hierarchical Address-Vector Quantization", *IEEE Journal on Selected Areas on Communications*, Vol. 10, No. 5, pp. 960-967, June 1992.

[65] M. Goldberg and H. Sun, "Image Sequence Coding Using Vector Quantization", *IEEE Trans. on Communications*, Vol. COM-34, No. 8, pp. 703-710, July 1986.

[66] M. Goldberg and H. Sun, "Frame Adaptive Vector Quantization for Image Sequence Coding", *IEEE Trans. on Communications*, Vol. 36, No. 5, pp. 629-635, May 1986.

[67] P. Monet and C. Labit, "Codebook Replenishment in Classified Pruned Tree-Structured Vector Quantization of Image Sequences",*IEEE 1990 International Conference on Acoustics, Speech, and Signal Processing* pp. 2285-2288, April 1990.

[68] F. Lavagetto and S. Zappatore, "Unbalanced Tree Structure for Frame Adaptive Vector Quantization of Image Sequences",*SPIE: Image Processing Algorithms and Techniques*, Vol. 1244, pp. 291-304, 1990.

[69] C. L. Yeh, "Color Image Sequence Compression Using Adaptive Binary-Tree Vector Quantization with Codebook Replenishment", *Proc. Internat. Conf. Acoust. Speech Signal Process. 87*, Dallas, Texas, pp. 1059-1062, April 1987.

[70] D. Chen and A. C. Bovik, "Visual Pattern Image Coding", *IEEE Trans. on Communications*, Vol. 38, No.12, pp. 1662-1671, December 1990.

[71] Z. Xie and T. G. Stockham, "Previsualized Image Vector Quantization with Optimized Pre- and Postprocessors", *IEEE Trans. on Communications*, Vol. 39, No.11, pp. 1662-1671, November 1991.

[72] ISO, "JPEG Digital Compression and Coding of Continuous Tone Still Images",Draft ISO 10918, 1991.

[73] G. K. Wallace, " The JPEG Still Picture Compression Standard", *Communication of the ACM*, Vol. 34 , No. 4, pp. 31-45, April 1991.

[74] CCITT, "Video Codec for Audiovisual Services at $p \times 64$ kbits/s",*CCITT Recommendation H.261*, International Telegraph and Telephone Consultative Committee (CCITT), CDM XV-R 37-E, August 1990.

[75] ISO, "Coding of Moving Pictures and Associated Audio", Committee Draft of Standard ISO 11172, ISO/MPEG 90/176, December 1990.

[76] D. L. Gall, " MPEG: A Video Compression Standard for Multimedia Applications", *Communication of the ACM*, Vol. 34 , No. 4, pp. 47-58, April 1991.

[77] A. N. Choudhary and J. H. Patel, "Parallel Architectures and Parallel Algorithms for Integrated Vision Systems",*Kluwer Academic Publishers*, 1990.

[78] J. Kittler and M. J. B. Duff, "Image Processing System Architectures",*Research Studies Press*, 1985

[79] M. J. B. Duff, "Computing Structures for Image Processing",*Academic Press*, 1985.

[80] H. T. Kung, "Why Systolic Arrays?", *IEEE Computer*, pp. 37-46, January 1982.

[81] S. Y. Kung, "VLSI Array Processors", *IEEE ASSP Magazine*, pp. 4-22, July 1985.

[82] K. Dezhogosha, M. M. Jamali and Subhas C. Kwatra, "A VLSI Architecture for Real Time Image Coding Using a Vector Quantization Based Algorith", *IEEE Transactions on Signal Processing*, Vol. 40, No. 1, pp. 181-189, January 1992.

[83] G. A. Davidson, T. Stanhope, R. Aravind and A. Gersho, "Real Time Speech Compression with A VLSI Vector Quantization Processor", *Proceedings of the IEEE Conference Acoust., Speech, Signal Processing*, Vol. 4, pp. 1437-1440, March 1988.

[84] G. A. Davidson, P. R. Cappello and A. Gersho, "Systolic Architectures for Vector Quantization", *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-36, pp. 1651-1664, October 1988.

[85] R. Dianysian and R. L. Baker, "A VLSI Chip Set for Real Time Vector Quantization of Image Sequences", *IEEE Int. Symp. Circuits and Systems*, Vol. 1, pp. 221-224, May 1987.

[86] H. Abut, B. P. M. Tao and J. L. Smith, "Vector Quantizer Architecture for Speech and Image Coding",*Proc. IEEE conf. Acoust., Speech, Signal Processing*, Vol. 2, pp.756-759, April 1987.

[87] B. P. M. Tao, H. Abut and R. M. Gray, "Hardware Realization of Waveform Vector Quantizers", *IEEE J. Selected Areas in Commun.*, Vol.SAC-2, pp. 343-352, March 1984.

[88] M. Yan and J. V. McCanny, "A Bit-Level Systolic Architecture for Implementing a VQ Tree Search",*Journal of VLSI Signal Processing*, Vol. 2, pp. 149-158, 1990.

[89] M. Yan, J. V. McCanny and Y. Hu, "VLSI Architectures for Digital Image Coding",*IEEE 1990 International Conference on Acoustics, Speech, and Signal Processing*, Vol. 2, pp. 913-916, April 1990.

[90] P. A. Ramamoorthy B. Potu and T. Tran, "Bit-Serial VLSI Implementation of Vector Quantizer for Real-Time Image oding",*IEEE Transactions on Circuits and systems*, Vol. 36, No. 10, pp. 1281-1289, October.1989.

[91] A. Lafage and F. Jutand, "VLSI Architectures for Vector Quantization for Real-Time Image oding",*IEEE Transactions on Circuits and systems*, Vol. 36, No. 10, pp. 1281-1289, October1989.

[92] S. Panchanathan and M. Goldberg,"A Content-addressable Memory Architecture for Image Coding using Vector Quantization", *IEEE Trans. on Signal Processing*, Vol. 39, No. 9, pp. 2066-2078, September 1991.

[93] S. Panchanathan and M. Goldberg,"A Systolic Array Architecture for Image Coding Using Adaptive Vector Quantization", *IEEE Trans. on Circuits and Systems for Video Technology*, Vol. 1, No. 2, pp. 222-229, June 1991.

[94] T. Wehberg and H. Volkers, "Architecture for Programmable Real Time Processor for Digital Video Signals Adapted to motion Estimation Algorithms", *SPIE: Visual Communications and Image Processing '88*, Vol. 1001, pp. 908-916, November 1988.

[95] F. May," Full Motion 64 Kbit/s Video Codec with 8 DSPs", *International Workshop on 64 Kbit/s Coding of Moving Video*, Hannover, FRG, June 1988.

[96] I. Tamitani, H. Harasaki, T. Nishitani, Y. Endo, M. Yamashina, and T. Enomoto, " A Real Time Video Signal Processor Suitable for Motion Picture Coding Applications",*IEEE Transactions on Circuits and Systems*, Vol. 36, No. 10, pp. 1259-1266, October 1989.

[97] R. Dianysian, R. L. Baker AND J. L. SALINAS, "A VLSI Architecture for Template Matching and Motion Estimation", *ISCAS'88*, Finland, June 1988.

[98] K. M. Yang, L. Wu, H. Chong, and M. T. Sun, "VLSI Implementation of Motion Compensation Full Search Block Matching Algorithm", *SPIE: Visual Communications and Image Processing '88*, Vol. 1001, pp. 892-899, November 1988.

[99] T. Komarek and P. Pirsch,"Array Architectures for Block Matching Algorithms", *IEEE trans. on Circuits and Systems*, Vol. 36, No. 10, pp. 1301-1308, October 1989.

[100] L. D. Vos and M. Stegherr, "Parameterizable VLSI Architecture for the Full-Search Block Matching Algorithm", *IEEE trans. on Circuits and Systems*, Vol. 36, No. 10, pp. 1309-1316, October 1989.

[101] K. M. Yang, M. T. Sun and L. Wu, "A Family of VLSI Designs for the Motion Compensation Block Matching Algorithm", *IEEE trans. on Circuits and Systems*, Vol. 36, No. 10, pp. 1317-1325, October 1989.

[102] C. H. Chou and Y. C. Chen, "A VLSI Architecture for Real Time and Flexible Image Template Matching ", *IEEE trans. on Circuits and Systems*, Vol. 36, No. 10, pp. 1336-1342, October 1989.

[103] C. H. Hsieh and T. P. Lin, "VLSI Architecture for Block Matching Motion Estimation Algorithm", *IEEE trans. on Circuits and Systems for Video Technology*, Vol. 2, No. 2, pp. 169-175, June 1992.

[104] S. Madnick and J. Donvan, "Operating Systems", McGraw Hill 1978, pp. 155-156.

[105] Z. Hussain, "Digital Image Processing: Practical Applications of Parallel Processing Techniques", Ellis Horwood 1991.

[106] K. Hwang and F. A. Briggs, "Computer Architecture and Parallel Processing", McGraw Hill 1987.

[107] T. Kohonen, "Content Addressable Memories", Spring-Verlag, Second Edition, 1987.

[108] P. Dewilde and J. Vandewalle, "Computer Systems and Engineering", Kluwar Academic Publishers, pp. 65-99, 1992.

[109] F. Idris and S. Panchanathan, "Associative Memory Architecture for Video Compression", submitted to the *Fifth IEEE Symposium on Parallel and Distributed Processing*, 1993.

[110] F. Idris and S. Panchanathan, "Image Sequence Coding Using Frame Adaptive Vector Quantization", *Visual Communications and Image Processing '93*, to appear, November 1993.

[111] F. Idris and S. Panchanathan, "Adaptive Vector Quantizer for Image Coding", *Picture Coding Symposium '93*, pp. 5.2-5.3, March 1993.