Colour image representation by scalar variables

COLOUR IMAGE REPRESENTATION BY SCALAR VARIABLES

by

Hua Wang

A Doctoral Thesis

Submitted in partial fulfillment of the requirements for the award of the degree of Doctor of Philosophy of the University of Technology, Loughborough.

September, 1992

Supervisors : Mr. J. W. Burren and Professor J. W. R. Griffiths

Department of Electronic and Electrical Engineering,
Loughborough University,
(Visiting in the Rutherford Appleton Lab.)
England.

Abstract

A number of studies have shown that it is possible to use a colour codebook, which has a limited number of colours (typically 100-200), to replace the colour gamut and obtain a good quality reconstructed colour image. Thus colour images can be displayed on less expensive devices retaining high quality and can be stored in less space. However, a colour codebook is normally randomly arranged and the coded image, which is referred to as the index image, has no structure. This prevents the use of this kind of colour image representation in any further image processing.

The objective of the research described in this thesis is to explore the possibility of making the index image meaningful, that is, the index image can retain the structure existing in the original full colour image, such as correlation and edges. In this way, a three band colour image represented by colour vectors can be transformed into a one band index image represented by scalar variables.

To achieve the scalar representation of colour images, the colour codebook must be ordered to satisfy the following two conditions: (1) codewords representing similar colours must be close together in the codebook and (2) close codewords in the codebook must represent similar colours. Some effective methods are proposed for ordering the colour codebook. First, several grouping strategies are suggested for grouping the codewords representing similar colours together. Second, an ordering function is designed, which gives a quantity measurement of the satisfaction of the two conditions of an ordered codebook. The codebook ordering is then iteratively refined by the ordering function. Finally, techniques, such as artificial codeword insertion, are developed to refine the codebook ordering further.

A number of algorithms for colour codebook ordering have been tried to retain as much structure

in the index image as possible. The efficiency of the algorithms for ordering a colour codebook has been tested by applying some image processing techniques to the index image. A VQ/DCT colour image coding scheme has been developed to test the possibility of compressing and decompressing the index image. Edge detection is applied to the index image to test how well the edges existing in the original colour image can be retained in the index image.

Experiments demonstrate that the index image can retain a lot of structure existing in the original colour image if the codebook is ordered by an appreciate ordering algorithm, such as the PNN-based/ordering function method together with artificial codeword insertion. Then further image processing techniques, such as image compression and edge detection, can be applied to the index image. In this way, colour image processing can be realized by index image processing in the same way as monochrome image processing. In this sense, a three-band colour image represented by colour vectors is transformed into a single band index image represented by scalar variables.

Acknowledgments

I would like to acknowledge the financial assistance received from the Chinese government and the British council, which made this research possible. Acknowledgment also goes to the Rutherford Appleton Lab which provided equipments for the research.

I would like to thank my supervisors Mr. J. W. Burren and Professor J. W. R. Griffiths for their guidance, encouragement and much invaluable advice and discussion throughout the research. Mr. Burren also extensively reviewed this thesis, suggesting a lot of improvements, and teaching me the art of best organizing technical material.

There are several people I would like to thank a lot. Among them are: Dr. L. Zhang, who helped me in the design of the DCT board and provided much helpful advice in the research; Dr. M. K. Carter, who helped me with the experimental colour images; Ms. J. Haswell, who helped me in drawing the codebook distribution in the RBG colour space.

My heartiest gratitude goes to my parents and my husband Min for their support and constant encouragement. Min also read the draft thesis and suggested many improvements.

Contents

Chapter 1

Introduction

1.1 Background

Conventionally, the pixel in a colour image is represented by a colour vector which is composed of the three colour components of the colour space used. For example, in the RGB colour space, the colour vector is (r, g, b). The colour image then contains three band images corresponding to the three colour components. This kind of vector representation of a colour image brings about some problems in colour image processing. First, the amount of data in a full colour image is large since there is three times the image information compared to a monochrome image. Normally, the pixel in a full colour image is represented by 24 bits (about 16 million possible colours), 8 bits for each colour component. For example, the amount of data in a 512×512 full colour image is 6 Megabits (Mbits). A wideband channel is required to transmit the data and a large memory to store it. Second, the processing for colour images is more expensive than that for monochrome images, such as full colour image digitizing and displaying.

Third, the processing of colour images is more complex than that for monochrome images. The first step in colour image processing is normally to choose a proper colour space, in which the processing can be efficiently carried out. The selection of the colour space depends on the application. For example, since the RGB colour space corresponds directly with additive colour devices, such as the displays of computers, it is often used for its easy processing and programming avoiding any pre-transformation. The YUV colour space is used mainly in video communication systems. In the

YUV space, most energy is compacted on the Y component which is referred to as the luminance component and less energy in the U and V components which are referred to as the chrominance components. The HSV colour space is widely used in image analysis, since it approximates the perceptual properties of Hue, Saturation and Value. After the colour space is chosen, the colour image is transformed into the chosen colour space and then the processing techniques are applied to the three image bands separately in the same way as they might be applied to a monochrome image. For example, in the conventional edge detection for colour images [Robi77], first a proper colour space which is the most suitable for colour edge extraction is chosen and then the edge detection is performed on the individual components and finally the detected edges are suitably combined depending on the definition of the colour space. In the conventional colour image compression [Limb77] [Jain81], the colour image is firstly transformed into a proper colour space which can compact energy well, normally the YUV space, and then the compression techniques are applied to the Y, U and V images separately. The processing complexity, the memory requirement and the system expense are tripled compared with that of monochrome images. For example, if an image compression technique, such as the DCT, is implemented by a hardware board, three such boards are required for colour image compression processing, each of which is used for one band image.

Additionally, the independent processing on the three bands does not take into account the interaction of the three colour components. For example, in the colour image compression, the correlation among the three colour components is not employed. Furthermore, most image processing techniques have been originally designed and developed for monochrome images and do not consider the interaction among the three components when they are used in colour image processing. Therefore, they are not as efficient when applied to colour images as to monochrome images. The large amount of data, the complexity of the processing and the cost of the hardware involved make colour imaging systems expensive and impractical for some applications.

Some previous work has been carried out on colour image quantization for full colour image display [Brau87] [Bala91] [Gold 90] [Heck82] [Macd90]. Research in the field of the human visual system shows that human eyes can only distinguish a relatively small number of colours from the

colour gamut. For example, experiments show that the eye can distinguish about 128 hues of green, 64 hues of red and 16 hues of blue [Helm90]. Additionally, for a colour image, only a finite number of colours from the colour gamut are used [Pres91]. For a 512×512 colour image, the colours used are, at most, about 256, 000 colours, i.e. a different colour for each pixel. For these reasons, it is possible to use a limited number of colours to replace the colour gamut and achieve a very good quality of colour image display which has no significant difference from the original full colour image. That is to display a particular colour image by approximating the true colour of each pixel in the image by a colour from a colour codebook. This process is referred to as colour quantization. In colour quantization, a colour codebook is firstly generated by vector quantization algorithms. The colour codebook contains a limited number of colours from the colour gamut. Next each 24 bit pixel of the colour image is mapped into the address of its closest mapped colour in the colour codebook. The newly formed image is called an index image. When the index image is displayed, the colour indicated by the address in the codebook is displayed rather than the original colour. The quality of the reconstructed colour image is acceptable for many applications if the colour codebook is well generated. In this way, cheap display devices which can only display a small number of colours at a time, typically less than 256, can be used to display good quality colour images and less memory space is required to store the colour image to be displayed.

However, the index image generated in the above way is meaningless because the colour codebook has no structure. The structure existing in the original full colour image, such as the correlation, edges etc., cannot be retained in the index image by the unstructured colour codebook. Further image processing techniques cannot be applied to the index image. Therefore, though the colour image vector quantization can save a lot of colours in colour image representation, which is particularly useful in colour image display and storage, it is not helpful in the colour image processing.

1.2 Research Objective

The objective of the research addressed in this thesis is to explore the possibility of making the index image meaningful. That is to structure the codebook so that the index image can retain the structure existing in the original full colour image and further image processing techniques can be carried out on the index image. The meaningful index image is a one band image and can be processed in a similar way to a monochrome image. In this sense, the original three-band colour image represented by colour vectors can be transformed into a one-band index image represented by scalar variables, which are the indexes of the mapped colours in the codebook. Colour image processing can then be realized by applying image processing techniques to the index image in similar way as that in the monochrome image. To achieve the above objective, the research described in this thesis has been carried out. The steps to be carried out are envisaged as follows.

First, an appropriate colour codebook is generated. A good quality colour codebook should represent the original full colour image and give a good quality reconstructed colour image using a minimum number of codewords. The Pairwise Nearest Neighbour (PNN) algorithm [Equi89] and the Linde-Buzo-Gray (LBG) algorithm [Lind80] are chosen to generate the colour codebook, where the PNN algorithm is used to generate an initial colour codebook and the LBG algorithm is used to refine the initial codebook.

Second, the colour codebook is ordered. The codebook generated in the above way is normally randomly arranged. The codewords which are close in the codebook do not normally represent similar colours and similar colours may be far apart in the codebook. Using this randomly arranged codebook in the colour image vector quantization results in a meaningless index image. In order to retain as much structure in the index image as possible, it is desirable to order the colour codebook to satisfy the following two ordering conditions as far as possible: (1) codewords representing similar colours must be close together in the codebook and (2) close codewords in the codebook must represent similar colours. The colour codebook ordering is carried out in the RGB colour space. The first reason is for its simple processing. Because the image data captured and stored in the

framestore are normally in the RGB colour space, there is no pre-processing on the image data required before the ordering and the image vector quantization. The other reason for this is that it is easy to approximately determine the colour appearance of a codeword by the Red, Green, Blue tristimulous values though it may not be exact. For example, if the Red component of a codeword is much bigger than the Green and the Blue components, the codeword represents a reddish colour. The information about the approximate colour appearance of a codeword is very useful in the codeword rough classification and is taken advantage of in the design of the colour codebook ordering algorithms.

Several ordering methods for colour codebook ordering, such as the centroid method, the PNN-based method etc., are proposed and tested. These methods can group the codewords representing similar colours together in the codebook. An ordering function, which provides a quantitative measure of how well the arrangement of the colour codewords conforms with the two ordering conditions, is designed and it is used to iteratively refine the colour codebook ordering by reducing the ordering function value. Next techniques, such as artificial codeword insertion etc., are developed for the further refinement of the colour codebook ordering.

Finally, the 24 bit colour image is vector quantized by the ordered colour codebook into an 8 bit meaningful index image and the index image is tested to determine whether the structure existing in the original colour image is retained. The test is implemented by applying two image processing techniques, namely, the image compression and the edge detection, to the index image. If the results of the image processing on the index image are satisfactory, it indicates that the index image can retain most of the structure and the ordering techniques designed are effective.

1.3 Thesis Organization

The research addressed in this thesis is to explore the possibility of representing colour images by

scalar variables. To realize the scalar representation of colour images, it is crucial to generate a high quality colour codebook and structure the codebook to satisfy the two ordering conditions as far as possible. The colour codebook generation and ordering will be discussed in chapter 2 and chapter 3. In chapter 2, a number of algorithms for the codebook generation are introduced. Two algorithms, namely, the PNN algorithm and the LBG algorithm, are especially discussed because they are used in the later experiments. Chapter 3 mainly deals with the colour codebook ordering. In this chapter, the background, the possibility, the necessity and the difficulties of the colour codebook ordering are analyzed. The two ordering conditions which the codebook ordering should satisfy are presented. Then some efficient colour codebook ordering strategies, such as the PNN-based method, the ordering function, the artificial codeword insertion etc., are proposed.

Chapter 4 is about image quality evaluation. It begins with the basic knowledge of the human visual system, which is used in image quality evaluation and is helpful in the design of the colour codebook ordering algorithms. Two image quality evaluation techniques, namely, the objective and the subjective, are discussed.

In chapter 5, image compression techniques are discussed. Since image compression processing is an important test method for the colour codebook ordering in the later experiments, the compression techniques are discussed in detail. First, image compression techniques are reviewed. Then the discrete cosine transform which is used in the test is especially discussed. As a comparison for the colour image compression on the index image, the conventional colour image compression is discussed. Finally, a still image compression standard is briefly introduced.

Chapter 6 describes the experimental hardware environment on which all the experiments are carried out. The experimental hardware system is mainly composed of an IBM PC, a B004 Transputer board, a MicroEye TC colour image capture and framestore board and a DCT board. The DCT board which is designed to implement the DCT processing with high precision and speed, is particularly discussed in this chapter.

Chapter 7 describes the experiments for testing the colour codebook ordering techniques and discusses the experimental results. In the tests, the ordering techniques presented in chapter 3 are compared in three ways: (1) computing the ordering function on the ordered codebook, (2) applying edge detection to the index image and (3) applying the DCT image compression to the index image. Finally, chapter 8 concludes the thesis by discussing the experimental results and the possible future work.

Chapter 2

The Codebook Generation in Vector Quantization

The vector quantization technique is employed to approximate the colours used in colour images by a colour codebook, which is a small set of colours from the colour gamut. In this chapter, vector quantization is first briefly introduced. Since the final coding quality in vector quantization depends strongly on the codebook, in the remaining part of this chapter we focus on the discussion of the algorithms for the codebook generation, especially the LBG algorithm (§ 2.2) and the PNN algorithm (§ 2.3). The PNN algorithm and the LBG algorithm are used to generate the colour codebook in the experiments addressed in this thesis.

2.1 Vector Quantization

Vector quantization [Gray84] [Nasr88] is to approximate the vectors to be coded by the vectors from a codebook which is a small sample set of vectors representing the vectors to be coded. Before vector quantization, the data to be coded is decomposed into vectors. For example, the vectors for monochrome images can be small, spatially contiguous, non-overlapping square blocks of pixels. The vectors for colour images can be the three colour components or their transform primaries. Thereafter, a codebook is generated from the vectors to be coded and a vector in the codebook is referred to as a codeword. Vector quantization contains two mappings, the encoder and the decoder. The encoder maps each image vector into its closest codeword, and the index of the mapped code-

word in the codebook represents the vector as the final code. The decoder maps the index into the corresponding codeword in the codebook. The block diagram of vector quantization is shown in figure 2.1. Here, $\mathbf{c}_i$ is the *i*th codeword in the codebook, N_c is the number of codewords in the codebook and n is the number of the vector dimensions.

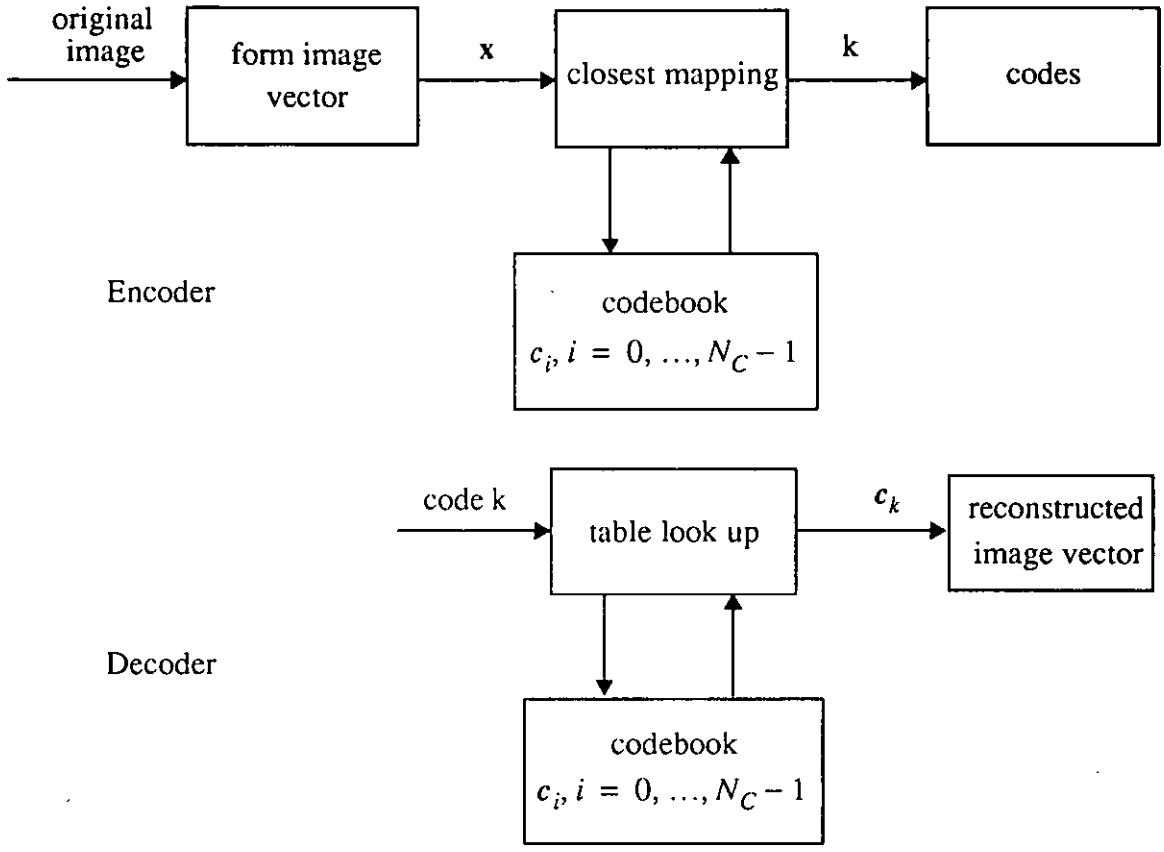


Figure 2.1 The block diagram of vector quantization

The closest matching in the encoder is realized by a minimum distortion rule. For an image vector $\mathbf{x}$, its closest codeword $\mathbf{c}_k$ is the codeword which can introduce the minimum distortion when it represents vector $\mathbf{x}$, i.e. choose codeword $\mathbf{c}_k$ to map image vector x so that $d(\mathbf{x}, \mathbf{c}_k) \leq d(\mathbf{x}, \mathbf{c}_j)$ for $j = 0, 1, 2, \ldots, N_c-1$, and, where $d(\mathbf{x}, \mathbf{c}_j)$ denotes the distortion introduced in replacing the original vector $\mathbf{x}$ by codeword $\mathbf{c}_k$. Ideally, the distortion measure should be mathematically tractable and subjectively meaningful, so that the quantitative distortion values correspond to perceived quality. The most common distortion measure used in image VQ is the mean squared error (MSE), which corresponds to the square of the Euclidean distance between two vectors as follows:

$$d(x, c_i) = \frac{1}{n} \sum_{j=1}^{n} (x_j - c_{ij})^2 \qquad (2.1)$$

The MSE is widely used because of its simplicity, though it does not necessarily correlate well with perceived quality. In practical applications, the weighted MSE may be more useful and it is given as:

$$d_w(x, c_i) = \frac{1}{n} \sum_{j=1}^{n} w_j (x_j - c_{ij})^2 \qquad (2.2)$$

where w_j is the weight factor applied to the jth vector component difference.

The codebook plays an important role in vector quantization because it determines the quality of the coding. Once the codebook is generated, vector quantization is a straightforward process of looking up the codebook to find the closest mapping from the input vector to the codeword. Codebooks are typically generated from a training set of image vectors. A local codebook is generated when the training set is the image to be coded. A local codebook can be a very good codebook but it must be regenerated for different images. This is a computational intensive task and cannot usually be performed in real time. Additionally, a local codebook must be transmitted to the receiver or stored as overhead information. A global codebook can be generated by using several images as a training set. If the images to be encoded belong to the same class of imagery (in terms of detail, resolution, image feature, colour etc.), a global codebook can produce good performance. But if the images to be encoded differ greatly, the performance of a global codebook may be substantially degraded. In such a case, the global codebook should be developed using as large and diverse a training set as possible to achieve reasonable average performance.

The algorithms usually used to generate the codebook are the Linde-Buzo-Gray (LBG) algorithm [Lind80] and the Pairwise Nearest Neighbour (PNN) algorithm [Equi89], both of which will be discussed in detail in the following two sections. Vector quantization is normally discussed in the

literature in the context of data compression, which will be discussed in chapter 5.

2.2 Review of Codebook Generation in Image Vector Quantization

Codebook generation is a process in which a set of vector representatives with the same dimensions as the image vectors is selected. The codebook can represent images with a good quality and the size of the codebook is much smaller than that of the input vector space. In the example of colour images, if each colour component is represented by 8 bits, there are about 17 million possible colours in the colour space, while the number of colours in a codebook is normally less than 256. Consequently, using the small size codebook to represent the original image will introduce distortion in the reconstructed image. A good quality codebook can approximate the image well with small distortion in the reconstructed colour image using minimum number of codewords, or a given number of codewords in some applications. The quality of the codebook is mainly dependent on the codebook generation algorithms. The performance of the codebook generation algorithm is measured by the algorithm complexity and the quality of the resulting codebook. In the research discussed in this thesis, the quality of the codebook is taken as the most important consideration.

In1980, Linde et al. [Lind80] proposed a codebook generation algorithm, which is referred to as the Linde-Buzo-Gray (LBG) algorithm. It is derived from the clustering technique. In the LBG algorithm, an initial codebook is first generated from a set of training vectors and then the codebook is iteratively refined by minimizing the distortion introduced in quantizing the training vectors by the codebook. Since the proposition of the LBG algorithm, it is almost exclusively used in the image vector quantization. One reason for this is that it is the first formal algorithm which is proposed for vector quantization. Secondly, it is simple to implement and the performance is quite satisfactory. Furthermore, it does not require any knowledge of the statistics of the input vectors. However, it suffers from several defects. First it depends on the initial codebook, so it can only produce a codebook locally optimum to the initial codebook. Secondly, its execution time is uncertain, be-

cause the iteration times cannot be determined in advance. Experiments show that its computation is very intensive and grows quickly as the size of the codebook gets larger, as the size of the training set gets larger, and as the vector dimension increases. Finally, the size of the codebook must be decided in advance and the size of the codebook is normally heuristically determined rather than by the requirements of the image. Thus it is possible that the size of the codebook is sometimes unnecessarily large, which wastes a lot of system resources, or sometimes too small to produce a good quality of the reconstructed image.

Some researchers have proposed that a codebook can be generated by pattern recognition "clustering" techniques. These techniques, however, suffer from some defects. First they cannot specify how many clusters will result ahead of the processing. Second, many typical clustering algorithms are not less complicated than the LBG algorithm. Finally the cluster points are generated to classify the points rather than to minimize the reconstructed distortion.

In 1989, Equitz [Equi89] presented a new method for codebook generation, which is known as the Pairwise Nearest Neighbour (PNN) algorithm. It works in a different way from the LBG algorithm. In the LBG algorithm, the size of the codebook stays the same in each iteration, while in the PNN algorithm the size will be reduced by one after each iteration. In the PNN algorithm, the codebook is generated by grouping the training vectors into clusters. The initial clusters are the training vectors and each cluster contains one training vector. There are N_t initial clusters. Then two close clusters are merged into one cluster and the number of the clusters is reduced by one. This merging process is repeated until the desired number of codewords is achieved or the distortion is within the acceptable limit. The collection of the centroids of each cluster is then used as the codebook. The PNN algorithm has some advantages over the LBG algorithm. First it is independent of the selection of an initial codebook. Second, it is not necessary to determine the size of the codebook before the codebook generation. Therefore, it is possible to minimize the number of codewords needed subject to a maximum allowable distortion or to minimize the distortion subject to predetermined number of codewords. The PNN algorithm can also be used to generate an initial codebook for the LBG algorithm, which can achieve better performance than either algorithm

separately and make the LBG algorithm converge after less iteration times.

In the field of the research in the colour image display and storage, some techniques are developed for colour codebook generation. Heckbert [Heck82] presented an algorithm to generate the colour codebook either by choosing the colours occurring with the highest frequency from the colour histogram of a colour image as the initial codebook, or by first choosing the colours which represent an equal number of pixels in the original colour images and then applying the LBG algorithm on the initial colour codebook. Braudaway [Brau87] proposed a procedure for optimum choice of a small number of colours from a large colour palette for colour images. He first generates an initial codebook and then refines the codebook by the LBG algorithm. In the initial codebook generation, the colour space is divided into a set of several equal size cubes, then the cube containing the biggest number of training vectors is chosen and the centroid of the vectors in the chosen cube is used as a codeword. Thereafter, the number of vectors in each cube is modified so that the number of the chosen cube is set to zero. The above process is repeated until the required number of codewords is obtained or all cubes contain no vector. Balasubramanian et al. [Bala91] made some improvements to the PNN algorithm by considering the features of the human visual system. The basic idea is that, since the human visual system is more tolerant to quantization noise in high activity regions, the activity measure is designed to signify whether a colour is high activity or low activity in a colour image. Then the activity measure in each cluster is computed. The distance between two clusters is weighted according to the activity measure, where the weight is big if the activity measure is small and vice versa. In this way, more codewords are used to quantize low activity regions and less codewords are used for high activity regions.

Though the algorithms discussed above, especially the LBG algorithm and the PNN algorithm, are quite efficient for codebook generation, there are some problems associated with them. First, there is no technique which can generate an optimum codebook. The LBG algorithm, for example, can only generate a codebook locally optimum to an initial codebook. The PNN algorithm cannot generate an optimum codebook either. Additionally, the codebook generation is a quite time consuming process. Though some work has been done to speed up the codebook generation by sacrificing

the quality of the codebook or by using a larger memory, it is still a time consuming process. Finally, the distortion measure used in vector quantization is normally the mean squared error for its simplicity, but this is not a good measure of colour difference in all regions of colour space.

2.3 The Linde-Buzo-Gray (LBG) Algorithm

The LBG algorithm [Lind80] is composed of two parts, the initial codebook generation and the initial codebook refinement. In subsection 2.3.1, several techniques for initial codebook generation are introduced. Then in subsection 2.3.2, the LBG algorithm is described.

2.3.1 The Selection of an Initial Codebook

The initial codebook is a crucial factor to decide the final codebook generated by the LBG algorithm, because the codebook is generated by iteratively refining an initial codebook. The codebook generated by the LBG algorithm is locally optimum to the initial codebook. There may be several locally optimum codebooks but many of these optimum codebooks may have poor performance. It is, therefore, important to begin with a good initial codebook in the LBG algorithm. Several techniques have been proposed to generate the initial codebook. One simple technique is to select the initial codebook randomly from the training vectors. The initial codebook generated in this way may not be well separated and is then not typical to represent the training vectors. A better approach is to choose N_c uniformly spaced vectors in the training set as an initial codebook, i.e. y_1, y_{k+1}, y_{2k+1},..., $y_{(Nc-1)k+1}$, where y_i is the ith training vector and N_c is the size of the codebook. The uniform selection depends on the statistics of the training set. The random and uniform selections are easy to perform but usually require more LBG iterations to reach convergence. Alternatively, the initial codebook can be selected according to the histogram of the training vectors. This method can generate a good quality initial codebook but it is more complicated and needs more computa-

tion. In the following part of this subsection, two more sophisticated methods for generating good quality initial codebooks are discussed. They are the **split method** and the **cube method**. The later method is used especially for colour codebook generation. The PNN algorithm is, of course, also an effective way to generate a good quality initial codebook. It will be discussed in detail in section 2.4.

The **split method** [Lind80] begins with one vector, normally the centroid of the entire training vectors, as a temporary initial codebook. Each vector in the temporary codebook is split into its two close vectors, i.e. split each vector $\mathbf{c}_i$ into $\mathbf{c}_i + \varepsilon$ and $\mathbf{c}_i - \varepsilon$, where ε is a fixed perturbation vector. After the splitting process, the size of the codebook is doubled. If the required number of codewords is obtained, the splitting process halts and the temporary codebook is the resulting initial codebook. Otherwise, use the temporary codebook to vector quantize the training vectors. Then each temporary codeword is replaced by the centroid of all the training vectors which are quantized by it. The splitting process is repeated on the newly formed temporary codebook in the above way. The size of the initial codebook generated in this way is a power of 2, i.e. 1, 2, 4, 8,..., 2^M.

The **cube selection** method [Brau87] has been proposed for generating an initial colour codebook. This is motivated by the idea of displaying high quality colour images on less expensive device which can only display a few colours at a time. It is applied to the RGB colour space and suppose each colour component is represented by 8 bits. First, it partitions the colour space into equal $32 \times 32 \times 32$ small cubes. This is achieved by dividing R, G, B into 32 uniform regions each, and each cube is assigned a count to signify the number of training vectors contained inside the cube. The cube which has the biggest count is chosen. Then the centroid of the vectors contained in the chosen cube is selected as a codeword and added to the initial codebook. Thereafter, the count of each cube is modified by a reduction function, which sets the count of the chosen cube to zero and reduces the counts in those cubes which are close to the chosen cube. The reduction degree is determined by the distance to the chosen cube, i.e. the smaller the distance the bigger the reduction. The aim of the cube count reduction is to select the colours which are well separated in the training set. The above process is repeated until the desired number of colours are obtained or the counts of all

cubes are zero. This method is simple to implement and takes into account the colour distribution of the training vectors.

2.3.2 Codebook Generation by the LBG Algorithm

The LBG algorithm [Lind80] generates the codebook by iteratively refining an initial codebook under the minimum distortion rule from a training set, so that the codebook can best represent the training set. It uses a long training set, which is statistically representative of the data to be quantized and is normally composed of image vectors from one or several images in the image vector quantization. Suppose the training set is $T = \{y_i;\ i= 0, 1,..., N_t - 1\}$. An initial codebook $A_0 = \{c_i^0,\ i =0, 1, 2,..., N_c - 1\}$ is generated by the algorithms discussed in the last section or the PNN algorithm discussed in the next section. Then the training vectors are partitioned into N_c groups in the way such that those training vectors which are vector quantized by the same temporary codeword are placed in the same group. Then the average distortion D_m (see equation 2.4) introduced by the quantization is computed. If the fractional change in the average distortion from the previous iteration is smaller than a positive small quantity ε, that is,

$$\frac{D_{m-1} - D_m}{D_{m-1}} \leq \varepsilon \qquad (2.3)$$

then the codebook generation process halts. Otherwise, the temporary codebook is replaced by the collection of the centroids of the vectors in each group and the above process is repeated.

The steps in the LBG algorithm are described as follows:

1. Initialization: generate a N_c-level initial codebook, $A_0 = \{c_i^0,\ i=0, 1, 2,..., N_c - 1\}$; choose a small positive quantity for the algorithm termination; set the initial average distortion D_0 to a large number, i.e. $D_0 = \infty$; and set the iteration counter m to 1.

2. Partition the training vectors by the following steps: first map each training vector to the nearest codeword in the temperate codebook, then those vectors which are encoded by the same codeword are put in the same partition group. That is, $P(T) = \{S_i, i = 1,..., N_c - 1\}$: If $y \in S_i$, then $d(y, c_i^m) \le d(y, c_k^m)$, for all k, $k = 0, 1, 2,..., N_c-1$.
3. Compute the average distortion introduced in the partition as:

$$D_m = \frac{1}{N_t} \sum_{j=0}^{N_t-1} \left\{ min_{c_i^m \in A_m} d(y_j, c_i^m) \right\} \qquad (2.4)$$

4. If the fractional change in the average distortion from the previous iteration is less than the small quantity ε, see Eq.(2.3), then the convergence has been achieved and the algorithm halts. Otherwise, go to step 5 to repeat the process.
5. Update the temporary codebook by the collection of the centroids of the vectors in each partition group. Replace m by m+1, go back to step 1.

It is possible that a partition group may be empty during the training set partitioning. An arbitrary vector is assigned as a centroid of the empty group and the algorithm continues. Alternative rules are possible. For instance, the group S_i is removed and then the algorithm continues with a (N_c-1)-level codebook. Alternatively, assign S_i the *i*th centroid from the previous iteration. In practice, a simple alternative is that, if the codebook generated by the algorithm has an empty group, simply retry the algorithm with a different initial codebook.

The convergence of the LBG algorithm has been demonstrated by a lot of experiments. In the above algorithm, if the threshold ε is set too small, the convergence may take many iterations. An alternative termination criterion can be a predetermined maximum number of iterations. Then the algorithm halts when the number of iteration exceeds the predetermined number.

For each iteration in the LBG codebook generation, there are two processes: a partition process of complexity $O(N_tN_cn)$, where n is the number of the vector dimension, and a centroid computation process of complexity $O(N_cn)$. As the centroid computation complexity is much smaller than the

partition complexity, it can be ignored. If the algorithm iterates I times, then the computation complexity of the codebook generation is $O(IN_tN_cn)$. The computation is proportion to the size of the training set, the size of the codebook, the vector dimension and the iteration times of the algorithm.

A fast version of the LBG algorithm [Chen91] has been proposed. It reduces computational complexity at the price of memory space. Because the LBG algorithm spends most of the time in the full search for the nearest neighbour in the codebook in the partition, the time saving can be achieved by just partial search rather than full search. It is known that the Euclidean distance satisfies the triangle inequality, i.e. $d(x, z) \leq d(x, y) + d(y, z)$ for any vector **y**. Suppose $\mathbf{c}_1$, $\mathbf{c}_2$ are two codewords, using the triangle inequality, it is easy to obtain the result: if $d(\mathbf{c}_1, \mathbf{c}_2) > 2d(\mathbf{y}, \mathbf{c}_1)$ then $d(\mathbf{y}, \mathbf{c}_2) > d(\mathbf{y}, \mathbf{c}_1)$. In this case, the distance $d(y, c_2)$ is not necessary to compute in the search for **y**'s nearest neighbour in the codebook. The condition is called the elimination condition, since if it is true, the computation of $d(\mathbf{y}, \mathbf{c}_2)$ can be eliminated. For squared error distortion measure, the elimination condition is $d(\mathbf{c}_1, \mathbf{c}_2) > 4d(\mathbf{y}, \mathbf{c}_1)$. The above result can be used in the LBG algorithm to speed up the nearest neighbour search in the training set partition. At the beginning, a table showing the distance between any pair of codewords is built. The search for the training vector **y**'s nearest neighbour works in the following way. First compute a distance between the training vector **y** and a codeword, use the eliminate condition to eliminate all unnecessary distortion computation and confine the further searching to the remaining codewords. This eliminate and confine process is repeated until x's nearest neighbour is found or no remaining codewords are left.

2.4 The Pairwise Nearest Neighbour (PNN) Algorithm

The codebook generation is actually a process in which the training set is first grouped into clusters and then each cluster is represented by a codeword which can introduce minimum distortion. The main difference among the codebook generation algorithms lies in the way of grouping the training vectors. The LBG algorithm, for example, groups the vectors of the training set by quantizing each

vector in the training set using a temperate codebook, that is, those vectors which have the same nearest codeword are put in the same group. While the PNN algorithm discussed below groups the training vectors by putting close vectors in a group so that in each group the variance of the vectors is small.

The PNN algorithm [Equi89] begins with a separate cluster for each vector in the training set and merges two clusters at a time until the desired codebook size is achieved. Two clusters, which are merged at any stage of the iteration, are the closest clusters among the current clusters in terms of a distortion measure so that the error introduced by replacing the two vectors with a single vector is minimized. The new cluster is represented by the centroid of the two vectors. An example in figure 2.2 illustrates the process of the first merge. It begins with a training set having six 2-dimensional vectors and each vector is considered as a cluster. The two components of each vector are represented as x and y coordinates on the graph. Each cluster centroid is represented in the diagram by a small empty circle and each cluster centroid has a number beside to signify the number of vectors inside the cluster. The first merge is to merge the two closest clusters and represent them by their centroid. The new cluster centroid has a number "2" beside it, signifying that it has two training vectors. After the first merging, there are five clusters. In the PNN algorithm, the squared error or the weighted squared error distortion measure is used as the error distortion and distance measurements, as shown in Eq.(2.6).

At each merging, choosing merging candidate clusters should consider not only the distance between the two clusters but also the minimum error introduced by the merge, that is, few training vectors are affected in each time merging. Given K clusters, it is always possible to optimally move to K-1 clusters by merging the two clusters, which results the best trade-off between merging close clusters and affecting few training vectors. If the members of a cluster can be approximated by their centroid, then this step-by-step optimal merge will lead to a good overall clustering. Considering the example in figure 2.3, which represents a typical merge in the PNN algorithm. In the merge, the two clusters containing four and one training vectors are merged, rather than the two clusters containing four and one hundred training vectors, though the later two clusters appear "closer".

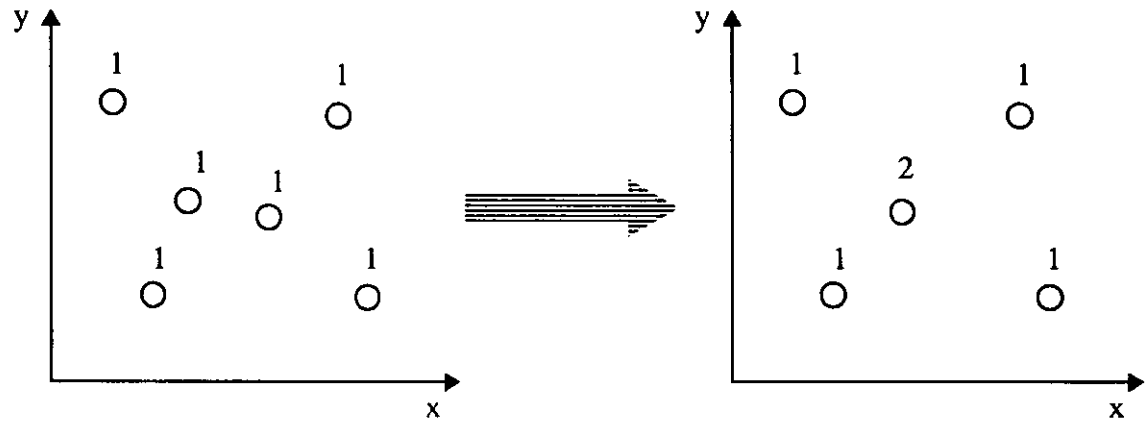


Figure 2.2 First merge in the PNN algorithm

This is because the larger error will be introduced to each training vectors in the later case.

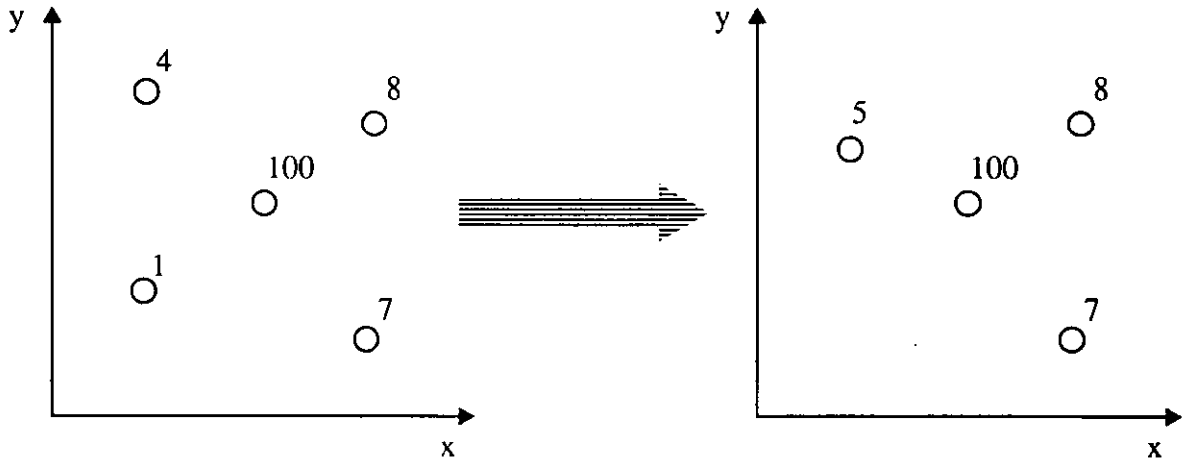


Figure 2.3 Typical merge in the PNN algorithm

The pair of clusters which will introduce the least error after the merge can be calculated as follows, where the squared error is used as the distortion measure and the following notations are used:

C_i = *i*th cluster of training vectors

C_{ij} = cluster formed by merging *i*th and *j*th clusters

n_i = number of training vectors in *i*th cluster C_i

n_{ij} = number of training vectors in cluster C_{ij}

$\bar{y}_i$ = centroid (mean) of the training vectors in *i*th cluster C_i

$\bar{y}_{ij}$ = centroid (mean) of the training vectors in cluster C_{ij}

d_i^2 = average squared error between $\bar{y}_i$ and the training vectors in cluster C_i

d_{ij}^2 = average squared error between $\bar{y}_{ij}$ and the training vectors in cluster C_{ij}

After the merge of two clusters, the distortion introduced is,

$$n_{ij}d_{ij}^2 = \sum_{y \in C_{ij}} |y - \bar{y}_{ij}|^2 = \sum_{y \in C_i} |y - \bar{y}_{ij}|^2 + \sum_{y \in C_j} |y - \bar{y}_{ij}|^2 \qquad (2.5)$$

It can be proven [Equi89] that,

$$n_{ij}d_{ij}^2 = n_i d_i^2 + n_j d_j^2 + \frac{n_i n_j}{n_i + n_j} |\bar{y}_i - \bar{y}_j|^2 \qquad (2.6)$$

The relation in Eq.(2.6) is interpreted as the squared error introduced by merging the *i*th cluster C_i and the *j*th cluster C_j. The PNN algorithm is to choose two clusters which can minimize the squared error introduced after their merging. It can be seen that, after a merging process, only factors n_k and $\bar{y}_k$ for the *k*th cluster, k=0, 1,..., (N_c-1), are required to be kept track. In fact C_k can be considered to be a vector of weight n_k at the centroid of the *k*th cluster. In this way, the distortion introduced by merging two clusters can be considered a weighted distance between the two centroids.

There are two possible termination criteria for the merging. The first one is that the algorithm stops when a predetermined number of clusters is obtained. Alternatively, it stops when the average error introduced by the merging is smaller than a predetermined appropriate threshold. The first termination criterion is often used, especially when the PNN algorithm is used as an initial codebook generator for the LBG algorithm. The second termination criterion is very useful when the PNN algorithm is used to generate minimum number of codewords to give an acceptable distortion, such as selecting the minimum number of colours to display high quality colour images. These termination criteria correspond relatively to minimizing distortion, subject to a predetermined size of the

codebook, and to minimizing the size of the codebook, subject to a distortion constraint.

The above process can be summed up as the following steps, where the squared error in Eq.(2.6) is used as the error distortion measure:

1. Initialization: begin with N_c clusters and each cluster contains one codeword corresponding to a training vector and here N_c is equal to N_t.
2. Search two clusters C_i and C_j which have the nearest weighted distance, i.e. $n_{ij}d_{ij}^2 \leq n_{kt}d_{kt}^2$ for all k and t, where $i \neq j$ and $k \neq t$, then merge these two clusters into one cluster and compute the centroid vector and the size of the newly formed cluster.
3. If the termination criterion is met, the algorithm halts and the collection of the centroids of each cluster is selected as the codebook. Otherwise, the number of clusters is reduced by one and go back to step two to repeat the merging process.

A fast version of the PNN algorithm is also proposed in [Equi89]. This speeds execution at the expense of codebook quality. In the PNN algorithm, most time is consumed in the search for the closest pairs of centroids among the current clusters. The complexity of each merge is on the order of $O(N_t \log N_t)$. The speed-up can be achieved if at each step two sub-optimal pairs of clusters rather than the absolute closest pair of clusters are merged as long as close clusters will be merged eventually. The approach to accomplish this speed-up is to use a k-d tree to partition the clusters into disjoint small sets, so that the search for the closest pair of clusters is confined to small sets. In this way, tremendous computational saving is achieved at the price of the sub-optimum quality of the codebook. The complexity of the fast PNN algorithm is on the order of $O(N_t)$, that is, its complexity is linear in the size of the training set.

Chapter 3

Colour Codebook Ordering

Chapter 2 discussed the codebook generation techniques, such as the LBG algorithm, and the PNN algorithm. Though these techniques can generate a good quality colour codebook for colour image vector quantization, the codebook is normally randomly arranged. Using this randomly arranged colour codebook in vector quantization, the index image has no structure and no further image processing techniques can be applied to the index image.

This chapter proposes some effective techniques which can order the colour codebook so that the index image can retain as much structure as possible. Section 3.2 introduces some methods which can roughly order the colour codebook. Sections 3.3, 3.4 and 3.5 present some strategies for refining the colour codebook ordering.

3.1 Colour Image Vector Quantization

Colour image vector quantization is to approximate the colours used in a full colour image by a colour codebook, which contains only a few number of colours, typically less than 256. That is to use a small number of colours to replace the colour gamut and obtain a good quality reconstructed colour image. In colour image vector quantization, the image vector is formed by the three colour components, such as (r, g, b) in the RGB colour space. The colour codebook is then a set of colours

which are the representatives of the colours used in a colour image. The colour codebook can be generated by the PNN algorithm and the LBG algorithm, both of which have been discussed in the previous chapter. The encoder in colour image vector quantization maps each image vector to its closest codeword in the colour codebook and the index of the mapped codeword becomes the pixel value. The coded image is referred to as the index image. The decoder maps each pixel in the index image to its corresponding codeword in the colour codebook. The reconstructed colour image is then represented by the colours in the colour codebook.

3.2 Background for Colour Codebook Ordering

The colour codebook generated by the algorithms discussed in the last chapter is normally randomly arranged. The index of a given codeword only indicates the codeword location in the codebook and the indexes of the codewords are independent of one another. The locations of the codewords in the colour codebook are not necessarily correlated with their locations in colour space. This means that, the codewords which have close index values, do not necessarily represent close colours in colour space and the codewords which represent close colours do not necessarily have close index values. The codewords are used separately in colour image vector quantization without any consideration of their relationship, such as the colour likeness or difference they represent. As a result, there is no structure in the index image because the pixels in the index image are unrelated data. On one hand, adjacent pixels in the index image do not retain the correlation existing in the original full colour image. On the other hand, the change of the adjacent pixel values in the index image does not give any information of the colour change in the original colour image and cannot indicate the edges existing in the original colour image. Whereas in a black and white image, for example, the value of a pixel represents the luminance intensity and the change of adjacent pixel values indicates the change of their luminance. There exists structure, such as correlation, edges etc., in a meaningful black and white image. Therefore, the image processing techniques, which can be applied, for example, to the black and white image, cannot be applied to the index image.

The goal of the colour codebook ordering is to structure the codebook so that the index image can retain as much of the structure existing in the original colour image as possible. This may be possible for two reasons. First, since a typical colour image, especially a natural colour image, uses only a relatively small number of colours in the colour gamut, the size of the codebook is normally rather small, typically 100 to 256. Second, the colours represented by a typical colour codebook are normally distributed in the form of clusters rather than uniformly in colour space. If the colours of a colour codebook are distributed uniformly in colour space, then colour codebook ordering is impossible. This is because in this case the ordering is equivalent to the space transformation from a three dimensional space to a one dimensional space.

It is possible to retain most of the structure existing in the original full colour image in the index image as long as the codebook is ordered in an appropriate way. In a typical colour image, there is a lot of colour correlation. This means that, in most parts of the colour image, adjacent pixels represent the same or similar colours. If the codewords which represent similar colours in colour space are put close in the codebook, then the colour correlation in the original full colour image can be retained in the index image. On the other hand, in a typical colour image, there are edges which are the boundaries between two regions with distinct properties. The edge is represented by the changes of properties, such as luminance, or chrominance or both, between adjacent pixels, and the change at edges of objects in the image may be gradual or abrupt. The edges existing in the original colour image can be retained in the index image, if the codewords which have big distance in colour space are put far away in the codebook and the bigger the distance is, the farther away they are in the codebook. Using a colour codebook ordered in this way, the index image can retain most of the structure existing in the colour image. The values of the pixels in the index image here not only indicate the indexes of the corresponding codewords in the colour codebook, but also can be used to describe most of the structure existing in the original full colour image. The ordered colour codebook makes the index image have a similar meaning to that of the original colour image. Consequently, image processing techniques can be applied to the index image.

It can be concluded from the above discussion that an ordered codebook should satisfy the following two conditions so that the index image can retain most of the structure:

<1> If $d(\mathbf{c}_i, \mathbf{c}_j)$ is small, then $|i - j|$ is small

$i, j = 0, 1, 2, \ldots, N_c - 1$, and $i \neq j$

<2> If $|i - j|$ is small, then $d(\mathbf{c}_i, \mathbf{c}_j)$ is small

$i, j = 0, 1, 2, \ldots, N_c - 1$, and $i \neq j$

where i and j are the indexes of codewords in the colour codebook, $\mathbf{c}_i$ and $\mathbf{c}_j$ are the ith and the jth codewords respectively, N_c is the size of the colour codebook, d() is the colour distance of two codewords in colour space. A small value of $d(\mathbf{c}_i, \mathbf{c}_j)$ means that the codewords $\mathbf{c}_i$ and $\mathbf{c}_j$ represent close colours. Here, we suppose that the codewords which are close in colour space represents similar colours. Normally, the Euclidean distance between two codewords is used to roughly compute their colour difference. It should be noted that the Euclidean distance does not always conform with the perceived colour difference.

The above two conditions are referred to as the two ordering conditions and they should be satisfied by the colour codewords arrangement as far as possible so that more structure can be retained in the index image. The first condition requires that the codewords which are close in colour space should be put close in the codebook. If it is not met, i.e. the codewords which represent close colours are put far away in the codebook, two problems will be caused. First, the colour correlation or colour redundancy in the original colour image cannot be retained in the index image. Second, the index image may have some incorrect edges which do not exist in the original colour image. The second condition requires that the codewords which are close in the codebook should be close in colour space. If this condition is not met, i.e. the codewords which are close in the codebook are far away in colour space, two problems will be caused. First, some edges existing in the original colour image cannot be retained in the index image. Second, the reconstructed colour image is very vulnerable to any errors in the index image. Even a slight distortion of the pixel value in the index

image will probably result in a poor quality reconstructed colour image. Though, in practice, it is not feasible to order the colour codebook to satisfy the two ordering conditions completely, it is desirable to satisfy the two conditions as far as possible.

While the description of the colour codebook ordering is concise, it is a difficult and challenging job to order a colour codebook to satisfy the two ordering conditions as far as possible. In the remaining part of this section, six typical difficulties associated with the colour codebook ordering are discussed as follows:

1. To any point in colour space, there are more than two points which have the same distance from it, whereas to a given codeword in a colour codebook, there are at most two codewords which are located at the same distance from it. This is because colour space is three dimensional while the codebook is one dimensional. In colour space, all the points on the surface of a sphere have the same distance from the centered point. If the diameter of the sphere is very short, all these points on the surface are close to the centered point. But it is impossible to make sure that all these close points are put adjacent or close to the centered point in the codebook. Though in practice, not all the points on the surface of the sphere will exist in a colour codebook, the case that a codeword has several codewords which have the same short distance from it often happens in a colour codebook. For example, the codewords which represent white, grey or dark, normally have several codewords which are at similar short distances from them. A codeword which has the same distance from more than one codeword is referred to as a junction codeword. Junction codewords, especially the codewords representing white, grey and dark, can be separated from the other codewords and are processed in different ways.

2. The Mean Squared Error (MSE) is not a precise measurement for determining the difference of colour codewords. Since a codeword is a three dimension vector, which is composed of magnitude and direction, the MSE only computes the magnitude difference of two codewords and does not take into account their direction difference. For

example, suppose there are four codewords, $\mathbf{c} = (r, g, b)$, $\mathbf{c}_1 = (r + \Delta, g, b)$, $\mathbf{c}_2 = (r, g+\Delta, b)$, $\mathbf{c}_3 = (r, g, b + \Delta)$, where Δ is a small positive quantity. Though the MSE values of codewords $\mathbf{c}$ and $\mathbf{c}_1$, codewords $\mathbf{c}$ and $\mathbf{c}_2$, codewords $\mathbf{c}$ and $\mathbf{c}_3$ are the same, codewords $\mathbf{c}_1$, $\mathbf{c}_2$, $\mathbf{c}_3$ are different from codeword $\mathbf{c}$ in different colour components and they represent colours which may look quite different. So the MSE only gives the magnitude change of two codewords but does not indicate how they change. Additionally, when the MSE is very big, it makes little sense to compare the difference of colour codewords. For example, suppose the MSE distance between codewords $\mathbf{c}_i$ and $\mathbf{c}_j$, and codewords $\mathbf{c}_i$ and $\mathbf{c}_k$ are both very big, there is no point in comparing these two distances to conclude the result that $\mathbf{c}_i$ is closer to $\mathbf{c}_j$ or closer to $\mathbf{c}_k$, especially when $\mathbf{c}_i$, $\mathbf{c}_j$ and $\mathbf{c}_k$ represent quite different colours. Finally, the MSE used as a colour difference measurement is not subjectively meaningful. The value of the MSE does not necessarily conform with the perceived colour difference. The same MSE values may have different perceived colour difference. Therefore, the MSE does not exactly indicate whether two codewords are close or far away, or which codeword is closer to a given codeword.

3. The distribution of codewords in colour space normally is not linear but quite complicated. One typical example is shown in figure 3.1(a). It can be seen that there is a junction which is composed of three close codewords, namely $\mathbf{c}_5$, $\mathbf{c}_6$, and $\mathbf{c}_{10}$. Based on the two ordering conditions, in the codebook, codewords $\mathbf{c}_1$, $\mathbf{c}_2$, $\mathbf{c}_3$, $\mathbf{c}_4$ and $\mathbf{c}_5$ should be close together; codewords $\mathbf{c}_6$, $\mathbf{c}_7$, $\mathbf{c}_8$ and $\mathbf{c}_9$ should be close together; codewords $\mathbf{c}_{10}$, $\mathbf{c}_{11}$ and $\mathbf{c}_{12}$ should be close together and codewords $\mathbf{c}_5$, $\mathbf{c}_6$, and $\mathbf{c}_{10}$ should also be close together. Three possible kinds of ordering are illustrated in figures 3.1(b), 3.1(c) and 3.1(d). In the ordering of figure 3.1(b), in the codebook the codewords are in the order: $\mathbf{c}_1\ \mathbf{c}_2\ \mathbf{c}_3\ \mathbf{c}_4\ \mathbf{c}_5\ \mathbf{c}_6\ \mathbf{c}_7\ \mathbf{c}_8\ \mathbf{c}_9\ \mathbf{c}_{10}\ \mathbf{c}_{11}\ \mathbf{c}_{12}$. In this ordering, though codeword $\mathbf{c}_5$ is very close to codeword $\mathbf{c}_{10}$, they are not close in the codebook, while codewords $\mathbf{c}_9$ and $\mathbf{c}_{10}$ are far away but they are adjacent in the codebook. In the ordering of figure 3.1(c), in the codebook the codewords are in the order: $\mathbf{c}_1\ \mathbf{c}_2\ \mathbf{c}_3\ \mathbf{c}_4\ \mathbf{c}_5\ \mathbf{c}_{10}\ \mathbf{c}_{11}\ \mathbf{c}_{12}\ \mathbf{c}_6\ \mathbf{c}_7\ \mathbf{c}_8\ \mathbf{c}_9$. This ordering is similar to the ordering in figure 3.1(b) but also has the problem that code-

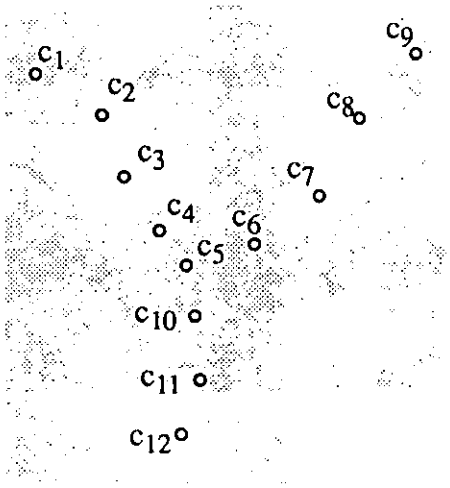


Figure 3.1(a)

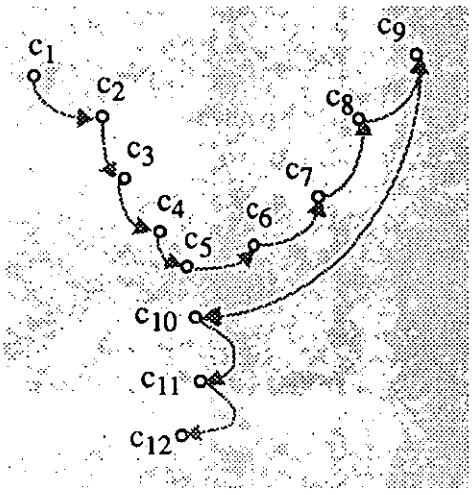


Figure 3.1(b)

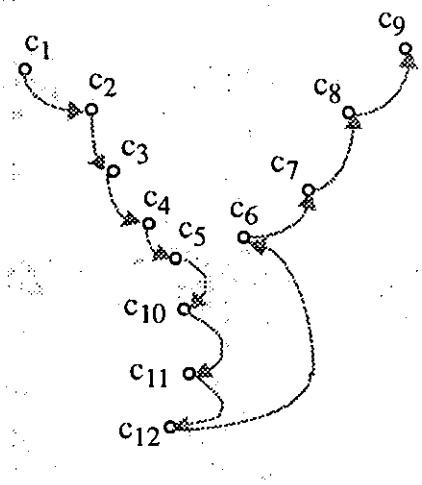


Figure 3.1(c)

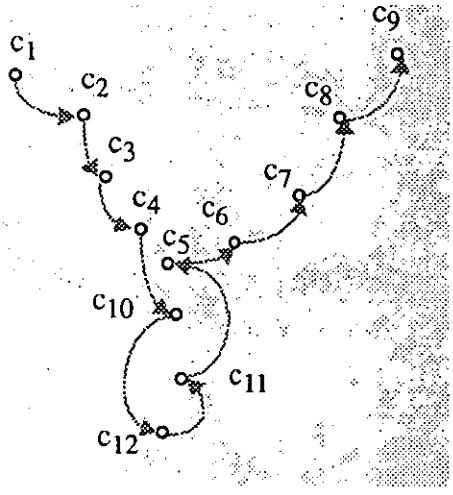


Figure 3.1(d)

Figure 3.1 An illustration of codebook ordering

Figure 3.1(a): the distribution of codewords; Figure 3.1(b), figure 3.1(c) and figure 3.1(d): three different kinds of ordering

word c_5 is very close to codeword c_6 but they are not close in the codebook, while codewords $\mathbf{c}_{12}$ and $\mathbf{c}_9$ are far away but they are adjacent in the codebook. In the ordering of figure 3.1(d), in the codebook the codewords are in the order: $\mathbf{c}_1\ \mathbf{c}_2\ \mathbf{c}_3\ \mathbf{c}_4\ \mathbf{c}_{10}\ \mathbf{c}_{12}\ \mathbf{c}_{11}\ \mathbf{c}_5\ \mathbf{c}_6\ \mathbf{c}_7\ \mathbf{c}_8\ \mathbf{c}_9$. In this ordering, codeword c_4 and codeword c_6 are very close, but they are not close in the codebook. This example shows that due to the complicated distribution

of colour codewords in colour space, codebook ordering which completely satisfies the two ordering conditions is sometimes impossible. Some strategies, like artificial codeword insertion and replacement, are developed to improve the ordering and they are discussed below.

4. Colour codebook ordering involves colour perception. In the colour codebook ordering, it is required that the codewords which represent close colours are put close in the codebook and the codewords which are close in the codebook should represent close colours. But colour perception is subjective and relative. The appearance of a colour to the human eye changes as the background against which the colour is viewed changes. What we see does not necessarily match the colour tristimulous values[1] of the colour. A well known example is that, when several squares with identical colour tristimulous values are put on a background which varies in colour and intensity, these squares can look different. The other simple example is that green colour looks greener when it is close to red and red colour looks redder when it is close to green. Colour perception depends on many factors, such as light sources, the viewing environment, the background, the response of the eye and the brain. Additionally, in the two ordering conditions, we use the small distance between two codewords in colour space to approximate the case that they represent close colours. But the close codewords in colour space, in the sense of small distance between them, may not look close. For example, codewords representing white, light green, light pink, light blue may be close in colour space, but they look different. This is especially so when they are put close together. Furthermore, some colours may look similar when they are viewed separately, but they look different when they are put together. All these features of colour perception make colour codebook ordering more complicated.

5. The second ordering condition cannot always be satisfied. A colour codebook normally

1. The amount of the three primary colours (blue, green and red) that form the colour being examined or matched. A tristimulous colourimeter can analyze a colour and indicates the amounts present of its constitute primary colours.

contains more than one cluster, each of which represents quite different colours. The codewords in the same cluster are close together in colour space while the codewords in different clusters are far away. If the first ordering condition is satisfied, that is, the codewords in the same cluster are put close in the codebook, then the codewords in the jointing position between two adjacent clusters cannot satisfy the second condition. For example, consider two adjacent clusters, the last codeword of the first cluster and the first codeword of the second cluster are far away in colour space but they are adjacent in the colour codebook. This problem cannot be solved by the ordering itself and other strategies must be introduced, such as artificial codeword insertion.

6. The selection of a proper threshold is very hard. Many thresholds used in colour codebook ordering are predetermined heuristically. One example is the definition of close and far away of two colour codewords in colour space. Two heuristic thresholds T and t are selected in advance. When the distance between two colour codewords is bigger than threshold T, then they are said to be far away and when the distance between two colour codewords is smaller than threshold t, then they are said to be close. The correctness of the decision whether two colour codewords are close or far away is dependent on the right selection of the two thresholds, which cannot be deduced from some formulas. The improper selection of these thresholds will give poor results and will eventually affect the final colour codebook ordering. Making decisions by thresholds is far from an exact science and can only be made on a "try-it-and-see-how-it-work" approach.

There is no function which can give an exact quantitative evaluation of the colour codebook ordering. The judgement on colour codebook ordering is based on how well the arrangement of the codewords satisfies the two ordering conditions. There is not such a function which can provide an exact evaluation of the codebook ordering because of the existence of the junction codewords in a colour codebook. A junction codeword is close to several codewords and these close codewords may be put in different clusters. According to the two ordering conditions, the junction codeword should

be put close to all these clusters. But in a linear codebook, the clusters which are close in colour space cannot always be put close together. In a certain ordering, the junction codeword is close to some clusters but far away from the others. Therefore, different arrangements of the junction codewords may have the same function values. The function value cannot indicate which of the clusters is the best one to which the junction codeword should be close.

In this case, the quality of colour codebook ordering depends on the colour image being coded. In the example shown in figure 3.1, if any region of the colour image to be coded is vector quantized either by codewords $\mathbf{c}_1$, $\mathbf{c}_2$, $\mathbf{c}_3$, $\mathbf{c}_4$ and $\mathbf{c}_5$, or codewords $\mathbf{c}_6$, $\mathbf{c}_7$, $\mathbf{c}_8$ and $\mathbf{c}_9$, or codewords $\mathbf{c}_{10}$, $\mathbf{c}_{11}$ and $\mathbf{c}_{12}$, rather than by codewords $\mathbf{c}_5$, $\mathbf{c}_6$ and $\mathbf{c}_{10}$ mixed, then the ordering illustrated in figure 3.1(b), figure 3.1(c) and figure 3.1(d) are all good, though they do not completely meet the two ordering conditions. This is because the problems caused by the disagreement of the two ordering conditions do not exist in the index image. However, if some regions in the colour image are vector quantized by codewords $\mathbf{c}_1$, $\mathbf{c}_2$, $\mathbf{c}_3$, $\mathbf{c}_4$, $\mathbf{c}_5$, $\mathbf{c}_6$, $\mathbf{c}_7$ and $\mathbf{c}_9$ mixed or codewords $\mathbf{c}_{10}$, $\mathbf{c}_{11}$ and $\mathbf{c}_{12}$ mixed, then the first ordering shown in figure 3.1(b) is better than the other two kinds of ordering shown in figures 3.1(c) and 3.1(d).

Normally, the colour codebook ordering is tested by checking the index image. If much of the structure existing in the original colour image can be retained in the index image, then the ordering is good. The test can also be realized by applying further image processing techniques to the index image, such as image compression, edge detection etc., and then checking the reconstructed colour image or the processed index image.

The remaining sections of this chapter deal with the techniques for colour codebook ordering. Section 3.3 proposes some methods for colour codebook ordering. In these methods, codewords are firstly grouped into clusters, in each of which codewords are close in colour space. Then the codewords in each cluster are ordered and finally the clusters are ordered. In section 3.4, an ordering function which can roughly measure the degree of colour codebook ordering is presented. The function can also be used to refine the ordering. The final two sections describe two techniques,

namely artificial codeword insertion and codeword replacement, which can be used to refine the ordering further.

3.3 Strategies for Colour Codebook Ordering

In this section, two algorithms for the first stage of colour codebook ordering are proposed. They are both based on the fact that the codewords used for a typical colour image are distributed in the form of clusters rather than uniformly in colour space. Therefore, the colour codewords can be firstly divided into several disjoint clusters, each of which contains codewords representing similar colours. Then the codebook is ordered by ordering the clusters and the codewords in each cluster.

It can be seen that these methods are composed of three processing steps, namely, generating clusters, ordering clusters and ordering codewords in each cluster. Cluster generation is a key step in the ordering and will be discussed in detail in the next two subsections, §3.3.1 and §3.3.2. The clusters can be ordered in the decreasing or increasing order of the magnitude of Y of the cluster centroid. Alternatively, the clusters can be ordered by the following steps: (1) choose the cluster with the biggest luminance value Y as the first ordered cluster; (2) find the cluster in which a codeword is the closest to the centroid of the just ordered cluster as the next ordered cluster; and (3) repeat step 2 on the unprocessed clusters until all the clusters are ordered. Finally, the codewords in each cluster are ordered. The ordering of codewords in each cluster can be realized by first finding the codeword which is the closest to the last codeword in the preceding cluster, then ordering the remaining codewords in the cluster according to their closeness to the codeword just ordered. The first codeword chosen in this way can smooth the joint place of two adjacent clusters. An alternative way is to choose the brightest codeword in each cluster as the first codeword.

3.3.1 Centroid Method

The centroid method is a simple method for clustering the codewords. Each cluster is formed by the following steps: at the beginning, a seed codeword is chosen as a one codeword initial cluster; then the codeword which is the closest to the cluster centroid is selected, if the codeword is close to the cluster centroid, then it is appended to the cluster and this process is repeated until no close codeword can be appended to the cluster. Figure 3.2 illustrates the procedure for the formation of a cluster. Let codeword $\mathbf{c}_1$ be the seed codeword and it is the initial cluster centroid. Since codeword $\mathbf{c}_3$ is the closest codeword to the current cluster centroid, i.e. codeword $\mathbf{c}_1$, and it is also very close to the current cluster centroid, it is appended to the cluster. The cluster now contains two codewords $\mathbf{c}_1$ and $\mathbf{c}_3$ and the cluster centroid is the centroid of codewords $\mathbf{c}_1$ and $\mathbf{c}_3$. Likewise, codewords $\mathbf{c}_2$, $\mathbf{c}_4$, $\mathbf{c}_5$, $\mathbf{c}_6$ and $\mathbf{c}_7$ are appended to the cluster successively. The number inside the parentheses indicates the appending order of the codeword to the cluster. For example, the number beside codeword $\mathbf{c}_5$ indicates it is the third codeword to be appended to the cluster. Note that this method is dependent on the selection of the seed codeword and different clusters may be resulted from choosing different seed codewords. The seed codeword is normally chosen by some features, such as the biggest luminance value Y, for the reason of simplicity.

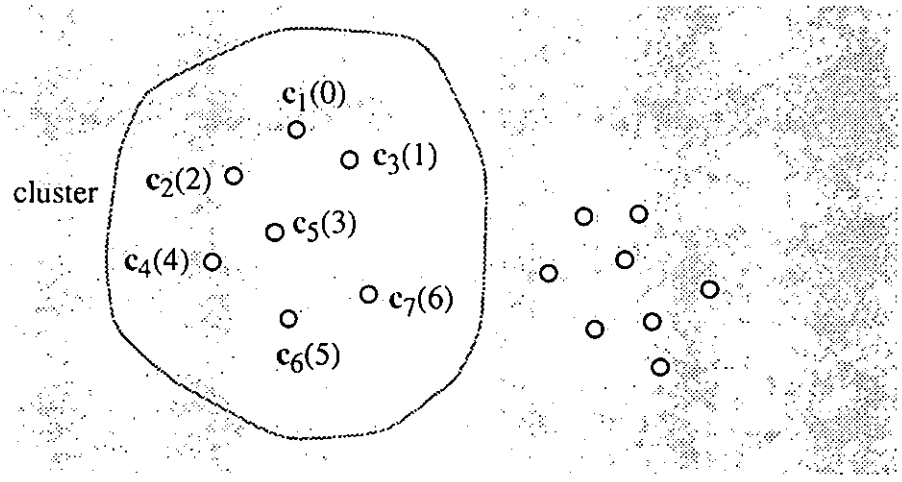


Figure 3.2 An illustration of a cluster formation

A specific algorithm for ordering a colour codebook by the centroid method is proposed. The fol-

lowing notations in the algorithm are used:

A: a set of unprocessed codewords which is initialized to be the colour codebook

$\mathbf{c}_i$: a codeword in A

C_i: a set of codewords in the *i*th cluster

n_i: number of codewords in cluster C_i

$\bar{c}_i$: the centroid of cluster C_i and is given as:

$$\bar{c}_i = \left(\frac{\sum_{c_j \in C_i} r_{c_j}}{n_i}, \frac{\sum_{c_j \in C_i} g_{c_j}}{n_i}, \frac{\sum_{c_j \in C_i} b_{c_j}}{n_i} \right) \qquad (3.1)$$

The algorithm is given as follows:

1. Find the codeword with the biggest Y, where Y is the luminance of the codeword, and take this codeword as the first member of cluster C_1 and continue with steps 2, 3 and 4 to form cluster C_1.
2. Search for codeword $\mathbf{c}_j$ in A, which is the closest to cluster centroid $\bar{c}_i$, that is,

$$d(\bar{c}_i, c_j) = min_{c_k \in A}(d(\bar{c}_i, c_k)) \qquad (3.2)$$

3. If codeword $\mathbf{c}_j$ is not close to the cluster centroid, that is, $d(\mathbf{c}_i, \mathbf{c}_j) > T$, where T is a positive appropriate threshold, then the formation of this cluster is completed and go to step 5 to form a new cluster, otherwise, add codeword $\mathbf{c}_j$ to cluster C_i and remove codeword $\mathbf{c}_j$ from A, that is,

$$C_i = C_i + \{\mathbf{c}_j\}$$
$$A = A - \{\mathbf{c}_j\}$$

   Then compute the new centroid of cluster C_i, see Eq.(3.1).
4. If there is no unprocessed codeword, that is, $A = \varnothing$, the algorithm halts, otherwise go back to step 2.
5. Codeword $\mathbf{c}_j$ is then used as the seed codeword of a new cluster C_{i+1}, and codeword $\mathbf{c}_j$ is removed from A, i.e. $A = A - \{\mathbf{c}_j\}$. If all the codewords are processed, that is A

$= \varnothing$, the algorithm halts, otherwise go back to step 2.

Actually, it can be seen from the above algorithm that, during the process of the clustering, clusters have been ordered, because the seed codeword of a cluster is the closest codeword to the centroid of the preceding ordered cluster. The codewords within each cluster can be ordered by the magnitudes of the Y of the codewords.

3.3.2 The PNN-Based Method

The Pairwise Nearest Neighbour (PNN) algorithm, see section 2.4, is an effective method to group data into clusters and it can be used as a way for generating clusters in colour codebook ordering. After the clusters are formed, the clusters and the codewords in each cluster can be ordered in the same way as that discussed in the last subsection.

The PNN-based codebook ordering algorithm is described as follows:

1. Initialization: begin with N clusters, each of which contains one codeword, that is to set $\hat{A}_0(N)$, here N is equal to the number of codewords N_c in the codebook.
2. Search two clusters C_i and C_j which have the nearest weighted distance, i.e. $n_{ij}d_{ij}^2 \leq n_{kl}d_{kl}^2$ for all k and t, where $i \neq j$ and $k \neq t$, then merge these two clusters into one cluster and compute the centroid vector and the size of the newly formed cluster, where $n_{ij}d_{ij}^2$ is the same as that defined in Eq.(2.6).
3. If the termination criterion is met, the algorithm halts. Otherwise the number of clusters is reduced by one and go back to step 2 to repeat the merging process.

Two termination criteria can be used. First, when the number of clusters is equal to a pre-determined number, then the algorithm halts. Alternatively, compute the distance between the two cen-

troids of the clusters to be merged, if the distance is bigger than a pre-determined threshold, then the algorithm halts.

3.3.3 Summary

The above two methods for colour codebook ordering are both based on the fact that the codewords used for a typical colour image are not uniformly distributed in colour space, but in the form of clusters. Colour codebook ordering can then be achieved by grouping codewords into clusters, ordering clusters, and ordering codewords in each cluster. The two methods are different in the way of grouping codewords. The centroid method is to form the cluster by appending codewords to a cluster while the PNN-based method is to form the clusters by iteratively merging two close clusters. The centroid method is simple and straightforward. The main process in it is to find the codeword to be appended. The PNN-based method makes use of the existing algorithm developed for the codebook generation in vector quantization, see section 2.4.

Both methods have the problem of choosing a suitable threshold for terminating the formation of the clusters. In the centroid method, the termination criterion used is that when the distance between the codeword to be appended and the centroid of the cluster being formed is bigger than a pre-determined threshold, then the codeword is not appended to the cluster and the formation of this cluster is finished. A constant threshold is applied globally throughout the formation of all clusters. However, a pre-determined constant threshold is not precise because clusters do not necessarily have the same scattering, some of them may be closely compacted and some of them may be loose. Therefore, the constant threshold may separate codewords which should be in the same cluster into different clusters. A variable threshold can be used locally depending on the cluster scattering but the variable threshold needs more computation.

In the PNN-based method, there are two ways for terminating the cluster formation. The first one

is to use a pre-determined cluster number and the second one is to use a pre-determined distortion threshold. Both of them have the following two problems. First, the pre-determined factor should be different for different images. Second, it can only be obtained heuristically and may not be very exact.

The centroid method also has problems associated with the 'seed' codeword. The cluster formation depends on the selection of the seed codeword. For example, suppose the codewords are distributed in the form as shown in figure 3.3. If codeword A is chosen as the seed codeword, then codewords B, C, D may be excluded from the cluster, while if the codeword B is chosen as the seed codeword, all the codewords can be included in the cluster, and actually all these codewords are close to one another. The improper selection of a seed codeword will result in improper clustering, that is, some codewords which are close to the codewords in a cluster and should be contained in that cluster are excluded from the cluster.

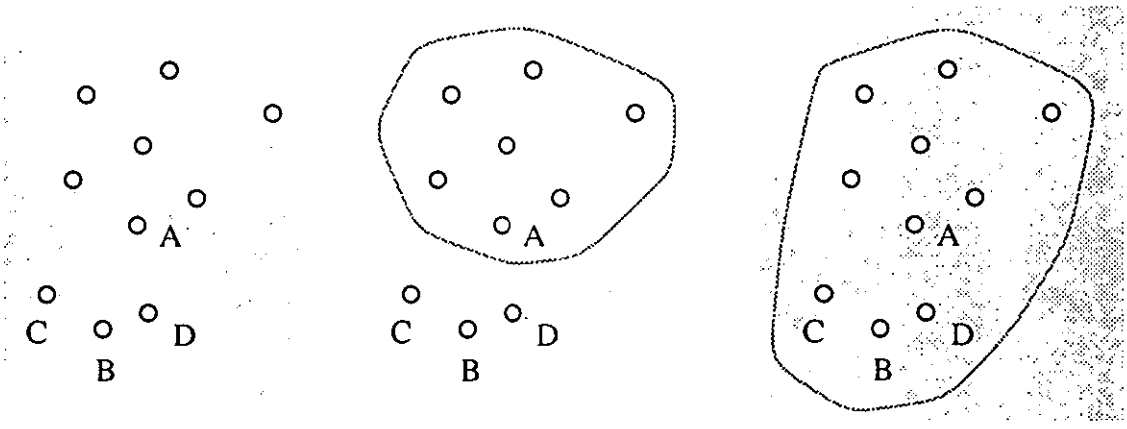


Figure 3.3(a) Figure 3.3(b) Figure 3.3(c)

Figure 3.3 An illustration of a cluster formation by choosing different seed codewords Figure 3.3(a) the distribution of the codewords; figure 3.3(b) codeword A is chosen as the seed codeword; figure 3.3(c) codeword B is chosen as the seed codeword

The two methods order colour codebook to a large extent but the ordering is not good enough. They are used as the first step in colour codebook ordering. Improvements to the ordering can be achieved by the techniques introduced in the next three sections.

3.4 Codebook Ordering Refinement

3.4.1 Ordering Function

It is important to know how well a colour codebook is ordered. Though the ordered colour codebook can be tested by examining whether the index image retains most of the structure existing in the original full colour image or by applying some image processing techniques to the index image, these kinds of test are not straightforward as they are applied to the index image rather than to the colour codebook. The ordered codebook cannot be judged until the colour image is vector quantized by it. It takes time and needs a lot of computation. Besides, the result of the test is rather rough. First, the result is descriptive rather than quantitative. Second, not all the improvement on the codebook ordering can be obviously reflected on the index image. This kind of testing may be useful for making a final evaluation of the ordered codebook but they are not very useful in the process of codebook ordering.

The ordering function is designed to provide a functional description of an ordered colour codebook. A colour codebook is said to be well ordered, if the arrangement of the codewords satisfies the two ordering conditions, that is, close codewords in colour space are close in the codebook and far away codewords in colour space are far away in the codebook. The ordering function then describes the relationship between an ordered colour codebook and its agreement with the two ordering conditions. Ideally, the ordering function can provide correct evaluation of the ordered codebook and is easy to compute. The ordering function proposed below is an important reference in evaluating an ordered codebook and is easy to compute. Additionally, since the ordering function provides a quantitative measurement for colour codebook ordering, it can be directly used in optimizing the ordering system.

Before the presentation of the ordering function, the variables used in the function are discussed. For each codeword, two variables are used to describe its relationship with the other codewords in colour space and in the codebook. In the RGB colour space, a codeword is represented by the three

colour components, namely Red, Green and Blue. Its relation with the other codewords can be described by the Euclidean distance between them. Consider reference codeword $\mathbf{c}_i$, $\mathbf{c}_i = (r_i, g_i, b_i)$, its relation with codeword $\mathbf{c}_j$, $\mathbf{c}_j = (r_j, g_j, b_j)$ in the RGB colour space is then:

$$d(c_i, c_j) = \sqrt{(r_i - r_j)^2 + (g_i - g_j)^2 + (b_i - b_j)^2} \qquad (3.3)$$

In the codebook, a codeword is represented by its index and its relation with the other codewords can be described by their index distances, i.e. their relative positions in the codebook. Consider codewords $\mathbf{c}_i$, $\mathbf{c}_j$,where i and j are the indexes in the codebook, the index distance is:

$$d(i, j) = |i - j| \qquad (3.4)$$

Since the colour distance $d(\mathbf{c}_i, \mathbf{c}_j)$ and the index distance d(i, j) have different meanings and ranges, they cannot be compared. Distance $d(\mathbf{c}_i, \mathbf{c}_j)$ is the Euclidean distance of two codewords and ranges from 0 to $\sqrt{2^{2B_r} + 2^{2B_g} + 2^{2B_b}}$, where B_r, B_g and B_b are the numbers of bits used to represent the Red, Green and Blue components respectively. Distance d(i, j) is the index distance of two codewords in the codebook and is a pseudo distance of two codewords. It ranges from 0 to N_c-1, where N_c is the number of codewords in the codebook.

In order to compare the two distances, one in colour space and one in the codebook, it is desirable to make them have the identical meaning. The index distance between reference codeword $\mathbf{c}_i$ with the other codewords can be taken as a set of ordinal numbers, i.e. {d(i, j), j=0, 1,..., N-1}, which indicate the positional closeness of the codewords to the reference codeword $\mathbf{c}_i$ in the codebook. The distance of reference codeword $\mathbf{c}_i$ with the other codewords in colour space can be transformed into a similar pseudo distance. This can be realized by sorting the codewords in the increasing order of the distances from the codewords to the reference codeword $\mathbf{c}_i$. In this arrangement of the sorting, it is likely that more than one codeword will have the same distance from the reference codeword. In this case, they are ordered in a random way. It can be seen that reference codeword $\mathbf{c}_i$ is sorted as the first, and the closest codeword to $\mathbf{c}_i$ is the second and etc. In this way, to each code-

word, its position in this order is its ordinal number which indicates its closeness to the reference codeword in colour space. Therefore, each codeword $\mathbf{c}_j$ has two ordinal numbers to a given reference codeword $\mathbf{c}_i$, one is in colour space denoted by $I_j^{(i)}$ and the other is in the codebook space denoted by $I'^{(i)}_j$. The ordinal number is also called close ordinal number.

One example is shown in figure 3.5 to explain these two close ordinal numbers. Suppose the codewords are arranged in the codebook in the order $\mathbf{c}_i$ $\mathbf{c}_{i+1}$ $\mathbf{c}_{i+2}$ $\mathbf{c}_{i+3}$ $\mathbf{c}_{i+4}$. Let codeword $\mathbf{c}_i$ be the reference codeword. In the codebook, the close ordinal numbers of codewords $\mathbf{c}_i$ $\mathbf{c}_{i+1}$ $\mathbf{c}_{i+2}$ $\mathbf{c}_{i+3}$ $\mathbf{c}_{i+4}$ to reference codeword $\mathbf{c}_i$ are 0, 1, 2, 3, 4 respectively, while in colour space, the close ordinal numbers of codewords $\mathbf{c}_i$ $\mathbf{c}_{i+1}$ $\mathbf{c}_{i+2}$ $\mathbf{c}_{i+3}$ $\mathbf{c}_{i+4}$ to reference codeword $\mathbf{c}_i$ are 0, 2, 4, 3, 1 respectively, which is decided by their distances from codeword $\mathbf{c}_i$.

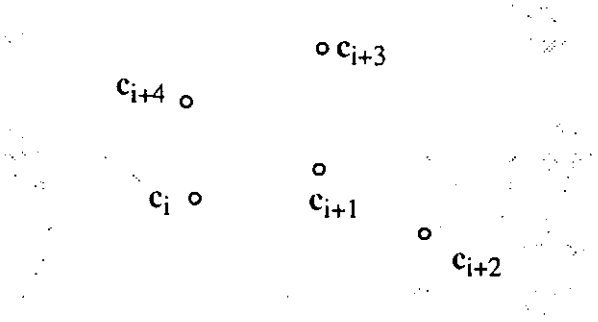


Figure 6.4 An example of the two close ordinal numbers

The ordering function of codebook B, i.e. F(B), is then defined as the function of the two close ordinal numbers, namely, $I_j^{(i)}$ and $I'^{(i)}_j$. It is the sum of the ordering function of all the codewords, i.e. f(i), i = 0, 1, 2,..., N_c -1, where f(i) is the sum of the squared difference of the two ordinal numbers of all the codewords to reference codeword $\mathbf{c}_i$, that is:

$$F(B) = \sum_{i=0}^{N_c-1} f(i) \qquad (3.5)$$

where
$$f(i) = \sum_{j=0}^{N_c-1} (I_j^{(i)} - I'^{(i)}_j)^2 \qquad (3.6)$$

The ordering function F(B), designed in the above way, can be used to measure how good the ordering of a colour codebook is. If a codebook is well ordered, the two ordering conditions are well satisfied. The first ordering condition is that close codewords in the codebook should be close in colour space. This indicates that, to any reference codeword $\mathbf{c}_i$, if codeword $\mathbf{c}_j$ has small $I'^{(i)}_j$, it should also have small $I_j^{(i)}$. In this case, $(I_j^{(i)} - I'^{(i)}_j)^2$ is small. The second ordering condition is that close codewords in colour space should be close in the codebook. This means that, to any reference codeword $\mathbf{c}_i$, if codeword $\mathbf{c}_j$ has small $I_j^{(i)}$, then it should have small $I'^{(i)}_j$. In this case, $(I_j^{(i)} - I'^{(i)}_j)^2$ is small. Therefore, if the two ordering conditions are satisfied, the ordering function F(B) is small. Likewise, it is obvious that if ordering function F(B) is small, then the two ordering conditions are well satisfied. The extreme case is that the ordering function value is zero. In this case, the two ordering conditions are satisfied by all the codewords. But in most cases, it is not possible to order the codebook so that F(B) is zero or be very small because of the complex and non-linear distribution of the codewords in colour space which has been discussed in section 3.1.

Since the value of $I_j^{(i)}$ ranges from 0 to N_c-1, it can be replaced by j, j=0, 1, 2,..., N_c - 1, in Eq.(3.6). Then in the function, $I'^{(i)}_j$ is replaced by $\tilde{I}_j^{(i)}$, which is defined as follows: suppose codeword $\mathbf{c}_k$ is the *j*th closest codeword to reference codeword $\mathbf{c}_i$ in colour space, its ordinal number to reference codeword $\mathbf{c}_i$ in the codebook is $\tilde{I}_j^{(i)}$, i.e., the index distance in the codebook between codeword $\mathbf{c}_k$ and $\mathbf{c}_i$. The ordering function in Eq.(3.5) and Eq.(3.6) can be simplified as,

$$F(B) = \sum_{i=0}^{N_c-1} f(i) = \sum_{i=0}^{N_c-1} \left[\sum_{j=0}^{N_c-1} \left(\tilde{I}_j^{(i)} - j \right)^2 \right] \qquad (3.7)$$

The amount of the computation in function F(B) is quite large and the computation complexity is $O(N_c^2)$. For every codeword, the difference between the two close ordinal numbers is computed

(N_c-1) times. However, much of the computation has no sense. When the close ordinal number in colour space is very large, it is not necessary that, only the two close ordinal numbers are the same, that is, $\left(\tilde{I}_j^{(i)} - j\right)^2 = 0$, the ordering is the best. In this case, as long as the close ordinal number in the codebook is bigger than the close ordinal number in colour space, then the ordering is fine. For example, for codeword $\mathbf{c}_i$, suppose its jth closest codeword in colour space is $\mathbf{c}_j$, and j is very big, say 40. Then it is not important if $\mathbf{c}_j$ has a distance in the codebook that is large but different from 40. Whatever the actual distance, the second ordering condition is met. Moreover, when the close ordinal number in colour space is large, it does not make sense to use the close ordinal number to compare the colour difference. For example, suppose codeword $\mathbf{c}_j$ is the 86th closest to codeword $\mathbf{c}_i$, and codeword $\mathbf{c}_k$ is the 87th closest to codeword $\mathbf{c}_i$ in colour space, then it may not be correct to make the conclusion that codeword $\mathbf{c}_j$ is closer to codeword $\mathbf{c}_i$ than codeword $\mathbf{c}_k$. In this case, it is not very important to put codeword $\mathbf{c}_k$ a little closer to $\mathbf{c}_i$ than $\mathbf{c}_j$ in the codebook to get better ordering. Therefore, when j is large, $\left(\tilde{I}_j^{(i)} - j\right)^2$ has little meaning and is not precise. Accordingly, ordering function F(B) can be approximated by the following:

$$F(B) = \sum_{i=0}^{N_c-1} f(i) = \sum_{i=0}^{N_c-1} \sum_{j=0}^{n'_i} \left(\tilde{I}_j^{(i)} - j\right)^2 \qquad (3.8)$$

where n'_i is the number of the close codewords to reference codeword $\mathbf{c}_i$ in colour space and $n'_i \le N_c - 1$. The number of close codewords n' can be decided in the following: if codewords $\mathbf{c}_k$ and $\mathbf{c}_{k+1}$ are the n' and $(n'+1)$ th closet codewords to reference codeword $\mathbf{c}_i$ and they have the relationship, $D(c_i, c_k) \le T$, and $D(c_i, c_{k+1}) > T$, where T is a pre-determined threshold, then the number of close codewords to reference codeword $\mathbf{c}_i$ is n'. In this way, to reference codeword $\mathbf{c}_i$, only its close codewords in colour space are considered in the computation of function f(i). This means that only those codewords which have small close ordinal numbers in colour space to reference codeword c_i are considered in the function.

The changed function Eq.(3.8) can describe how well the ordered codebook satisfies the first ordering condition, but it does not reflect how well the ordered codebook satisfies the second ordering condition. The following example can explain this case. Suppose some codewords are distributed

in the way shown in figure 3.5. Now consider codeword $\mathbf{c}_i$ as the reference codeword. Suppose the numbers beside the points are their close ordinal numbers in colour space to codeword $\mathbf{c}_i$ and the numbers inside the brackets beside the points are their close ordinal numbers in the codebook to codeword $\mathbf{c}_i$. In this example, codeword $\mathbf{c}_j$ has a rather big distance from codeword $\mathbf{c}_i$ and its close ordinal number in colour space to $\mathbf{c}_i$ is 40. On the other hand, it is very close to $\mathbf{c}_i$ in the codebook and its close ordinal number in the codebook to $\mathbf{c}_i$ is 1. The two close ordinal numbers are quite different and the second ordering condition is not satisfied in this arrangement. However, in the computation of the ordering function in Eq.(3.8), since $\mathbf{c}_j$ is not close to $\mathbf{c}_i$ in colour space, it is not considered in the computation of f(i) and f(i) is very small, f(i) = 1+ 1 + 1 + 1 = 4. Obviously the function is not exact in describing the ordering.

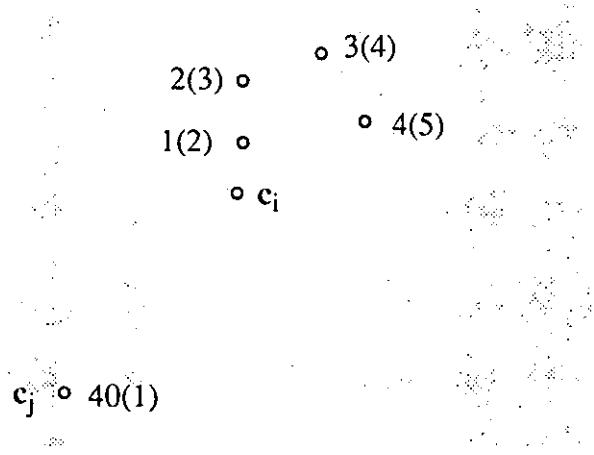


Figure 3.5 An example of the ordering function

To improve the ordering function so that the satisfaction of the second ordering condition is taken into account, the codewords which have big close ordinal numbers in colour space but small close ordinal numbers in the codebook with reference to codeword $\mathbf{c}_i$ should be reflected in the function f(i). Hence, the function f(i) can be modified as the following:

$$f(i) = \sum_{j=0}^{n'} \left[\left| \tilde{l}_j^{(i)} - j \right| + \sum_{k=0}^{n''} w_k \right]^2 \qquad (3.9)$$

where w_k is a weight, n'' is the number of those far away codewords which have big close ordinal numbers in colour space to the reference codeword $\mathbf{c}_i$ but are located in the codebook between

codeword $\mathbf{c}_i$ and the *j*th closest codeword to the reference codeword in colour space. Weight $\mathbf{w}_k$ can be constant, or a decreasing function against the close ordinal number in the codebook of the far away codewords to reference codeword $\mathbf{c}_i$, that is the closer the far away codeword from the codeword $\mathbf{c}_i$, the bigger the weight is. The ordering function F(B) is now:

$$F(B) = \sum_{i=0}^{N-1} \sum_{j=0}^{n'} \left[\left| \tilde{I}_j^{(i)} - j \right| + \sum_{k=0}^{n''} w_k \right]^2 \qquad (3.10)$$

Using this function in Eq.(3.10) to compute the above example, the result is f(*i*) = 4(w+1). Thus f(*i*) can be very large when the weight is big.

One problem in the ordering function is that the MSE is not an exact distance measurement for colours. It has been stressed in the first section of this chapter that since colour space is a three dimensional space, the MSE on its own is not enough to tell the difference of two codewords. For example, suppose codeword $\mathbf{c}_w$ represents white colour, and codewords $\mathbf{c}_r$, $\mathbf{c}_g$, $\mathbf{c}_b$ represent reddish, greenish and bluish colours respectively and they have the relation: $d(c_w, c_r) = d(c_w, c_g) = d(c_w, c_b)$. Codeword $\mathbf{c}_w$ is a junction codeword for it has more than one codeword which has the same small distance from it. Then the distance in this case does not indicate which codeword among codewords $\mathbf{c}_r$, $\mathbf{c}_g$, $\mathbf{c}_b$ is closer to codeword $\mathbf{c}_w$. Actually, it makes no sense to point out which is closer to codeword $\mathbf{c}_w$, because they represent different colours. One solution is to put them in four different clusters rather than in one cluster even though the distances between them are very small. To realize this, before computing the distance, colour classification is roughly made. Every codeword is marked by a classification mark, which is composed of three bits, e.g., mark_r, mark_g and mark_b respectively. If there is no other colour component whose value is bigger than the Red component value to a certain quantity, then mark_r is set to be 1, otherwise mark_r is 0. The same computation is applied to mark_g and mark_b. For the achromatic colours, the three bits are all 1, that is $\text{mark}_r = \text{mark}_g = \text{mark}_b = 1$. In this way, the codewords are roughly classified. Before the distance computation of two codewords, their marks are compared. If the marks are different, then their distance is taken to be very big.

The ordering function F(B) is the summation of f(i), i=0, 1,..., N_c-1, and it describes the overall ordering of a colour codebook rather than local ordering. A small function value indicates that the overall ordering is very good but cannot guarantee that all the codewords are in their optimum positions. It is likely that the function value is very small but some codewords may be not in their optimum positions. Also, it is likely that several different orderings have the same ordering function value. The other problem associated with the ordering function is that the two close ordinal numbers are slightly different. The close ordinal number in colour space is a monotonic variable while the close ordinal number in the codebook is not. A codeword may have two codewords which have the same close ordinal numbers in the codebook from it, one of which is on its right side and one of which is on its left side. This brings some distortion to the result of comparing the two close ordinal numbers.

3.4.2 Refinement of Colour Codebook Ordering by Reducing the Ordering Function

The colour codebook ordering can be refined by reducing the ordering function value. On one side, a small ordering function value indicates consistency between the close ordinal number in colour space and its corresponding close ordinal number in the codebook. Also consistency between the two close ordinal numbers results in a small ordering function value. On the other side, if the two close ordinal numbers are consistent, the two ordering conditions can be satisfied by the ordered codebook and if the two ordering conditions are satisfied, the two close ordinal numbers are consistent. Therefore, a small ordering function value indicates good satisfaction of the two ordering conditions and good satisfaction of the two ordering conditions by the codebook corresponds to a small ordering function value. The ordering refinement can be realized by reducing the ordering function. To reduce the ordering function is to make the two close ordinal numbers more conformable. The close ordinal number in colour space is fixed and cannot be changed, but the close ordinal number in the codebook can be changed by modifying the codeword position in the codebook. Therefore, the ordering function value can be reduced by adjusting the codeword arrangement in

the codebook.

The process of adjusting the codeword arrangement in a codebook is to adjust each codeword position to the position where the maximum reduction of the ordering function value can be achieved so that the ordering function value of codebook ordering is reduced. This process is repeated until the modification of codeword positions cannot bring any reduction of the ordering function value. That is, when the above process terminates, all the codewords in the codebook are in their optimum positions with respect to the current codebook ordering. There are two important steps associated with the codeword position adjustment process. One is the order that a codeword is chosen for the codeword position adjustment process. One simple approach is to choose codewords in the order of the arrangement of the codewords in a codebook. That is the codeword selection order conforms with the codeword index in the codebook. In this case, for example, the first codeword to be processed is the first codeword in the codebook and the second codeword to be processed is the second codeword in the codebook. This kind of selection order is quite straightforward and very simple, but in this selection order, the first codeword to be processed is not necessarily the codeword which can bring the maximum reduction of the ordering function value. Since the ordering function can functionally describe the codebook ordering, smaller ordering function values indicate better codebook ordering and bigger reductions of the ordering function value mean that more refinement of the codebook ordering has been achieved. Additionally, it is always hoped that, at each time, the process of codeword position adjustment is carried out on a well ordered codebook. Alternatively, the codeword to be processed can be chosen in this way: always choose the codeword which can bring the maximum reduction of the ordering function value. In this way, subsequent codeword adjustment can always be carried out on a better arrangement of codewords in the codebook. The second method of codeword selection can achieve better performance in codebook ordering than the first one but needs more computation.

The second crucial step in the process of codeword position adjustment is how to find the optimum position in the codebook where a codeword is to be moved to. An optimum position of a codeword is the position where, when the codeword is put there, the ordering function value is minimum. One simple way is to search exhaustively all the possible positions in the codebook. In a codebook with N_c codewords, there are (N_c - 1) possible positions that a codeword can be moved to. Then, to each codeword, the ordering function has to be computed (N_c - 1) times to find the optimum position. This requires quite a large amount of computation. From the discussion about codebook ordering in the last section, it can be seen that the refinement of codebook ordering by the ordering function is carried out after the codebook has been initially ordered by the centroid method or the PNN-based method discussed in section 3.3. The centroid method and the PNN-based method both can group the codewords in a codebook into several disjoint clusters, in each of which the codewords represent close colours. Therefore, the process of searching for the optimum position for a codeword can be confined within the positions within the cluster to which the processed codeword belongs or in its several close clusters. In this way, the search range of positions can be reduced to a large extent.

One way of minimizing the ordering function value is to adjust the position of the codeword in the codebook which has the biggest f(i) to the place where its new f(i) is a minimum. This process is repeated until no reduction of the function value can be obtained. However, the codeword with the biggest f(i) is not necessarily the codeword which is in the most unsuitable position in the codebook. The codeword which is in the most unsuitable position is the one which, when it is adjusted to its optimum position, produces the biggest reduction in the function value. An alternative approach is to adjust all the codewords to their optimum positions. This is described in the following algorithm. Suppose B is the codebook:

1. Compute the initial ordering function $F_0(B)$ on the codebook.
2. Change the place of codeword c_i to its optimum place where the ordering function $F(B')$ is a minimum. This process is applied to all the codewords in the codebook.
3. If the ordering function on the changed codebook $F(B')$ is equal to $F_0(B)$, the algorithm halts, otherwise go back to step 2 and the codebook is replaced by the changed codebook B' and $F_0(B)$ is replaced by $F(B')$.

There is a large amount of computation in the above processing because for any codeword there are N possible changed positions in the codebook and the ordering function had to be computed N times to decide the best position. An alternative way is to change the codeword to the place where f(i) is minimum rather than the ordering function F(B). The amount of the computation is then reduced at the price of the quality of the codebook ordering.

A further refinement can be achieved by adjusting several consecutive codewords at the same time rather than only one codeword. Sometimes it is likely that a set of close codewords are grouped in several clusters which are not adjacent in the codebook and every codeword is in its local optimum position. Consider close codewords $c_1\ c_2\ c_3\ c_4\ c_5\ c_6\ c_7\ c_8$. Suppose they are arranged in the codebook as: $c_1\ c_2\ c_3\ c_4$ x x x... x x x $c_5\ c_6\ c_7\ c_8$, where they are separated by some codewords and they

are all in their local optimum positions. Refining the ordering with the above method cannot adjust them together. Since they are all in their local optimum positions, no codeword can be adjusted. For example, if codeword $\mathbf{c}_1$ is moved adjacent to codeword $\mathbf{c}_5$, it will be far away from close codewords $\mathbf{c}_2$ $\mathbf{c}_3$ $\mathbf{c}_4$, and this may not bring any function value reduction. One approach to this is to move codewords $\mathbf{c}_1$ $\mathbf{c}_2$ $\mathbf{c}_3$ $\mathbf{c}_4$ or $\mathbf{c}_5$ $\mathbf{c}_6$ $\mathbf{c}_7$ $\mathbf{c}_8$ at the same time. In this way, they can be adjusted together. To do this, the codebook is first divided into several segments, in each of which codewords are close together, then adjust the consecutive codewords in each segment to reduce the ordering function value.

The codebook ordering refinement can be achieved by reducing the ordering function in the above way but this does not necessarily generate the optimum ordering. The reason for this is that a codeword or several consecutive codewords can only be adjusted to their local optimum places with respect to the arrangement at that time, but the arrangement of the codewords at that time is not necessarily the best one. The extent to which the codebook ordering can be optimized by the ordering function depends on the correctness of the ordering function and the ability to exploit the function to refine the ordering. Nevertheless, using the ordering function defined above to refine the ordering in the above way does put more close codewords together and improve the ordering.

3.5 Artificial Codeword Insertion

After the codebook has been ordered and refined, there are still likely to be a number of rough places, particularly the joint positions of two adjacent clusters. One example is shown in figure 3.6. Suppose codewords are distributed in the way of figure 3.6. Two clusters are formed, one is $\{\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4, \mathbf{c}_5\}$ and the other cluster is $\{\mathbf{c}_6, \mathbf{c}_7, \mathbf{c}_8, \mathbf{c}_9\}$. In the codebook, these codewords can be arranged in the order: $\mathbf{c}_1$ $\mathbf{c}_2$ $\mathbf{c}_3$ $\mathbf{c}_4$ $\mathbf{c}_5$ $\mathbf{c}_6$ $\mathbf{c}_7$ $\mathbf{c}_8$ $\mathbf{c}_9$. In this arrangement, the distance between codewords $\mathbf{c}_5$ and $\mathbf{c}_6$ is very big but they are adjacent in the codebook. This arrangement of codewords does not satisfy the second ordering condition. This situation can be improved by inserting some "arti-

ficial codewords" between the two adjacent codewords which are far away in colour space, so that they are located with some distance rather adjacent in the codebook to satisfy the second ordering condition.

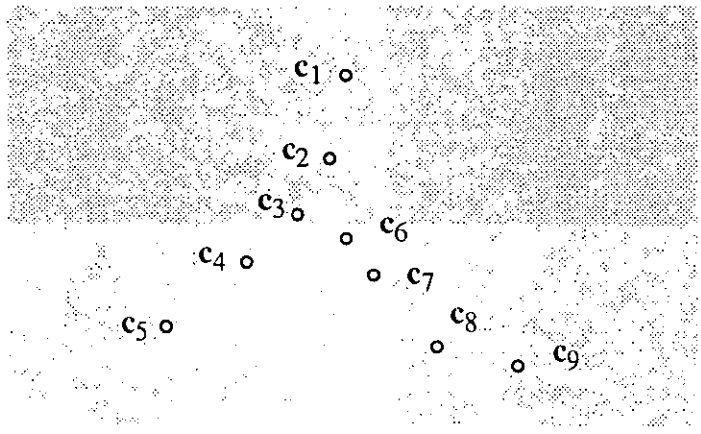


Figure 3.6 An illustration of the artificial codeword insertion

An effective way of codeword insertion is to insert some copies of those adjacent codewords which are distant from each other in colour space. In the above example, artificial codewords are inserted between codewords c_5 and c_6, that is: $c_1\ c_2\ c_3\ c_4\ c_5\ c_5 ...\ c_5\ c_6\ c_6 ...\ c_6\ c_7\ c_8\ c_9\ c_{10}$. The number of the copies inserted depends on the distance of the two codewords in colour space. Normally, the bigger the distance between two adjacent codewords, the more insertions are required. The joint place of two adjacent clusters may need more insertions, especially the clusters in which codewords represent achromatic colours. This is because the codewords in two different clusters normally represent different colours though sometimes the distance between the last codeword of the first cluster and the first codeword of the second cluster is small.

The indexes of artificial codewords are not used in the index image, but the artificial codewords can enlarge the distance between the adjacent codewords which are distant in colour space. In this way, the edges caused by the adjacent codewords which are far away in colour space can be retained in the index image. Additionally, artificial codeword insertion is necessary and helpful when there is some distortion in the index image and it can decrease the degradation generated by this

distortion in the reconstructed colour image. For example, after the DCT processing on the index image, the original index image cannot be recovered exactly. If there is no artificial codeword insertion, any error of the index pixel which happens to be around the rough place may cause obvious visual distortion on the final recovered colour image.

Though the insertion of some artificial codewords into the rough places can improve the codebook ordering, it also introduces some problems in the ordering. If two close codewords are on the two sides of the inserted place, the artificial codeword insertion will enlarge the distance between the two close codewords in the codebook. For example, in figure 3.6 suppose the codewords are arranged in the order: $\mathbf{c}_1\ \mathbf{c}_2\ \mathbf{c}_3\ \mathbf{c}_4\ \mathbf{c}_5\ \mathbf{c}_6\ \mathbf{c}_7\ \mathbf{c}_8\ \mathbf{c}_9$. Since codewords $\mathbf{c}_5$ and $\mathbf{c}_6$ are adjacent in the codebook but distant in colour space, artificial codewords are inserted between them. However, the insertion also enlarges the distance between codewords $\mathbf{c}_3$ and $\mathbf{c}_6$ which are close in colour space. This is normally happened inside a cluster. Therefore, less artificial codewords are inserted between the codewords which are in the same cluster even though their distance may be large. Artificial codeword insertion is used in the experiments discussed in chapter 7.

3.6 Codeword Replacement

Due to the complexity of the colour codeword distribution, sometimes there are still some similar colour codewords which are far away in the codebook, no matter how the codebook is ordered. This case often happens when the codewords which are around a junction codeword are ordered. Suppose a set of codewords are distributed in the way shown in figure 3.7. One possible codeword ordering is: $\mathbf{c}_1\ \mathbf{c}_2\ \mathbf{c}_3\ \mathbf{c}_4\ \mathbf{c}_5\ \mathbf{c}_6\ \mathbf{c}_7\ \mathbf{c}_8\ \mathbf{c}_9\ \mathbf{c}_{10}\ \mathbf{c}_{11}\ \mathbf{c}_{12}$. Codeword $\mathbf{c}_4$ and codeword $\mathbf{c}_9$ are close in colour space but they are not close in the codebook. This ordering can be improved by replacing codeword $\mathbf{c}_9$ by one of its close vectors which is a little farther away from codeword $\mathbf{c}_4$. For example, codeword $\mathbf{c}_9$ can be replaced by the centroid of codewords $\mathbf{c}_9$ and $\mathbf{c}_{10}$.

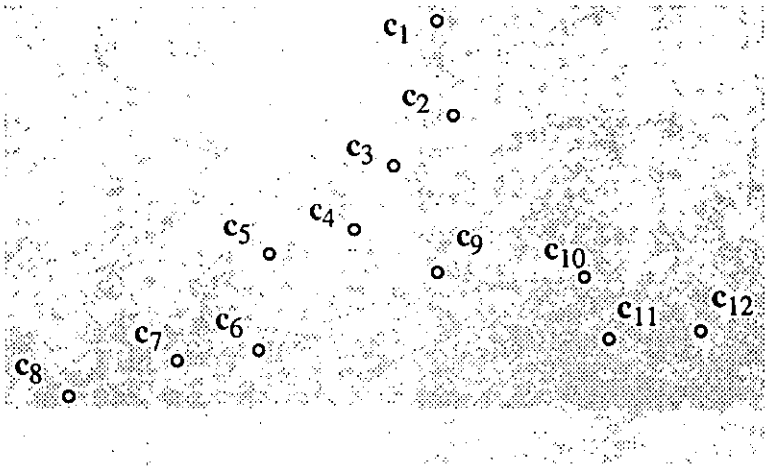


Figure 3.7 An example of the codeword replacement

Codeword replacement can decrease the disagreement with the first ordering condition in the codebook ordering, but it will change the codewords. The changed codeword is not necessary a good codeword for vector quantization and will degrade the quality of the reconstructed colour image. Though the strategy of codeword replacement can refine the ordering from the point of the view of better satisfaction of the first ordering condition, care is needed to confine its application to specific and special cases. This strategy has been tried in the experiments, but it is not used in the final codebook ordering discussed in chapter 7.

Chapter 4

Image Quality Evaluation

Image quality is an important factor for evaluating most image processing systems. Since the knowledge of the human visual system can be exploited in better understanding and designing the techniques for image processing and image quality evaluation, the first section of this chapter discusses the human visual system, especially some characteristics and interesting visual phenomena of both brightness and colour perception. The next two sections introduce some methods of subjective (§ 4.2) and objective (§ 4.3) evaluation of image quality.

4.1 Human Visual System

Since the human eye is one of the important end users to most image processing systems, understanding of the human visual system is helpful not only in the evaluation of processed images but also in better understanding, designing and optimizing an image processing system. A typical example is the visual communication system. Based on a knowledge of the human visual system, it only represents and transmits information which can be perceived by human eyes to save transmission bandwidth and reduce the amount of required memory. The human visual system is a very complicated system which contains the eye, the optic nerve and part of the brain. Its properties depend on many aspects, such as the physical characteristics of the eye and the characteristics of both the eye and the brain. Though the human visual system is still far from completely understood, a

lot of useful experimental results have been obtained. In this section, first the structure of the human eye is briefly discussed. Since some results from the research on the visibility threshold are helpful in image quality evaluation, next the visibility threshold for brightness will be discussed. Finally, colour perception and some colour visual phenomena are discussed.

The human eye is a unit which converts visual information into nerve impulses for the brain to form a perceived image. The main components in the eye [Gonz87] [Netr88] which play important roles in visual perception are: cornea, lens, and retina. The cornea is a tough, transparent tissue that covers the anterior surface of the eye. The retina is the innermost membrane of the eye and covers the posterior portion of the eye. Light from an external object is focused by the cornea and lens to form an image of the object on the retina. The retina consists of a densely packed population of photoreceptors and some connecting nerve cells. The photoreceptors are translating the incoming light into nervous impulses and are of two classes, namely cones and rods. The cones and rods play different roles in visual perception. The cones are mainly located in the central part of the retina, which is called the fovea and are responsible for spatial acuity and colour vision at normal daylight levels. Cone vision is known as photic or bright-light vision. In contrast, the rods have a large area of distribution over the retinal surface. The rods are responsible for low light vision and provide a general and overall picture of the viewed field. For example, an object which is viewed brightly coloured in daylight appears colourless in moonlight because only rods are stimulated. Rod vision is known as scotopic or dim-light vision.

In the neural presentation of an image, experiments indicate that there are five channels, two of which are spatial frequency channels and three of which are colour coding channels [Gran79]. The two spatial frequency channels contain a spatially low pass channel and a bandpass channel. The spatially low pass channel carries information about the degree of contrast across the image, while the bandpass channel carries edge information from the image. The three colour coding channels contain one achromatic and two chromatic channels. It is known that cones are of three types which are sensitive to red, green and blue colours and are commonly known as red, green and blue cones. Studies show that, in particular, the achromatic channel carries the information from the red and

green cones; one chromatic channel is a red/green channel by differencing data from the red and green cones; and the other chromatic channel is a yellow/blue channel by differencing data from the luminance channel (yellow = red + green) and the blue cones.

The visibility threshold of a stimulous is an important quantity in image quality evaluation, especially when the visibility of coding impairment is concerned. The visibility threshold of a stimulous is defined as the magnitude of the stimulous at which it becomes just visible or just invisible [Limb79] [Netr88]. If a threshold at which an impairment becomes just visible is known, then it is possible to estimate the quality of images accurately by specifying whether the distortion reaches the threshold. The visibility threshold of a stimulous depends on many factors. Two factors which are directly used in image coding are discussed here, namely average background luminance level against which the stimulous is presented, and luminance changes adjacent to the stimulous. These two factors affect the visibility threshold in two aspects, spatial and temporal. Since the still image is the main study object in the research discussed in this thesis, only the spatial aspect of those factors is considered. For simplicity, the two factors are assumed to be separable, though in reality they are not independent.

Experiments for determining the dependence of the visibility threshold on a background with constant luminance can be designed as shown in figure 4.1. The background luminance is denoted by L_b. The area outside the background is called the surround area and has luminance L_s. The area inside the inner circle is called the stimulous area and the luminance in the stimulous area is perturbed. The magnitude ΔL of the perturbation which is just visible is determined in the experiments. Experiments show that the visibility threshold ΔL increases as the background luminance L_b increases. This means that, when the background becomes brighter, the change of stimulous which can be just perceived is larger. Experiments also illustrate that though the visibility threshold depends on both the background luminance and the surround luminance, it has a stronger dependence on the background luminance and a weaker dependence on the surround luminance, that is, it is more dependent on the luminance of the close area than that of the relatively far away area. In the special case, when the surround luminance and the background luminance are the same, the vis-

ibility threshold ΔL increases almost linearly with background luminance L_b, i.e., the ratio $\Delta L/L_b$ is a constant. This is known as Weber's law and the quantity $\Delta L/L_b$ is called the Weber ratio. The Weber ratio and the surround luminance have the relation illustrated in figure 4.2. It can be seen that the Weber ratio is nearly constant over a large range of the surround luminance. When the surround luminance is very small, the Webber ratio decreases as the surround luminance increases and when the surround luminance is very big, the Weber ratio increases as the surround luminance increases. Weber's law demonstrates that high visibility thresholds will occur in the regions of a picture that are either very dark or very bright, while the lower threshold will occur in the medium to dark-grey regions and the visibility threshold in this case is proportional to the background luminance. Therefore, applying Weber's law to image coding, the impairment is easier to perceive in the medium to dark-grey regions but harder to perceive in very dark or very bright regions.

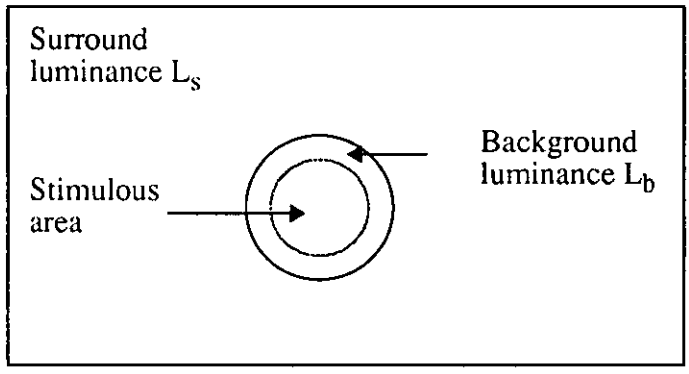


Figure 4.1 Display for the experiments to determine the dependence of the visibility threshold on the background and surround luminance

Experiments show that the visibility threshold is also affected by the luminance change which is adjacent to the test stimulous. It is known that the visibility threshold is increased by a non-uniform luminance background. This is referred to as spatial visual masking of the test stimulous by a non-uniform background. A typical experiment for demonstrating the spatial masking is that a thin vertical line of light is presented at various locations relative to a vertical edge and the visibility threshold of the line stimulous is measured. Some experimental results can be obtained. First, for the edge of both low and high contrast, the visibility threshold rises as the edge is approached from either

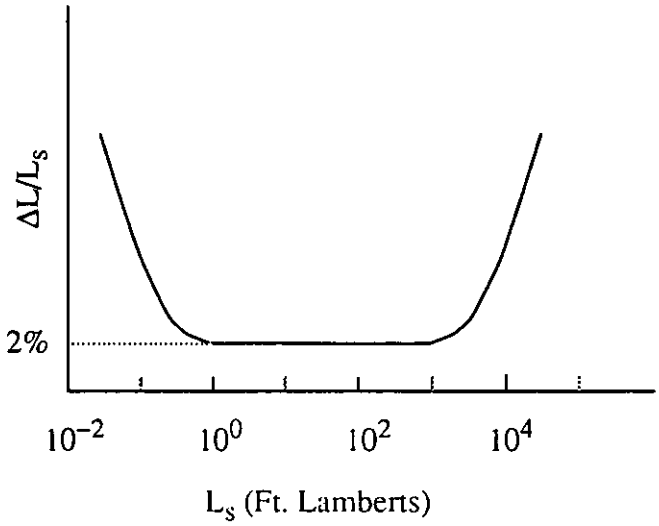


Figure 4.2 The Weber ratio vs. the surround luminance

side; and second the visibility threshold is increased a little if the luminance transition in the background edge is small while the visibility threshold is increased significantly if the edge is very sharp. The result regarding the spatial masking which can be used in the image coding is that distortion is harder to perceive in a non-uniform background than in a uniform background and is much harder to perceive near big spatial detail than brief detail.

The results related to the visibility threshold discussed in the above can be employed in the design of the function for image quality evaluation. Suppose the Weighted Mean Squared Error (WMSE) is used as a rough indication of the quality of a processed image. The weights in the WMSE can be designed according to the results obtained from the study on the visibility threshold. For example, if the luminance value of a pixel is very large or very small, its weight in the WMSE computation is set small. Additionally, for a pixel, the activities of its surrounding pixels are computed. If its neighbouring pixels are very active, i.e. there are big spatial details around it, its weight in WMSE computation is set small, otherwise the weight is set big.

Colour perception is more complex than brightness perception. It is known that the cones on the retina are responsible for colour perception by means of light-sensitive chemicals called photo-pigments within them. There are three different types of cones which are sensitive to three areas of the

visible spectrum, with peaks at approximately 445 (blue), 535 (green) and 570 (red) nanometers. These three cones are commonly known as red, green and blue cones. Each type of receptor integrates the energy in the light from an object at various wavelengths in proportion to their sensitivity for that wavelength. The perception of a colour has three attributes, namely, brightness, hue and saturation, which are generally used to distinguish one colour from the others. Brightness refers to intensity and is the degree of bright or dim of an object appears to have. Hue is associated with the dominant wavelength in a mixture of light waves and it is the attribute of visual sensation where a stimulous appears to be a dominant colour, such as red, green, yellow, blue etc. Hue is a distinctive characteristic of any chromatic colour that distinguishes it from the other hues. Saturation is the attribute of a perceived colour that tells how different the colour is from an achromatic colour (white or grey). Since each colour can be considered to have two components, chromatic and achromatic, saturation can be characterized as the proportion of the chromatic and achromatic components. The degree of colour saturation is inversely proportional to the amount of white light in the colour. For example, the pure spectrum colours are fully saturated while colours, such as pink (red and white) is less saturated. Hue and saturation together are called chromaticity.

Colour appearance is very complicated because colour is the perceptual experience of the human observer. Though it has been studied for long time, a standard procedure for measuring colours does not exist. The perceived appearance of a colour stimulous depends on many factors, such as the background, the shape of the observed objects, the spectral power distribution of the source light etc. Next, some experimental results are studied.

The appearance of a colour is dependent on the colour of its background [Gers85] [Macd90]. First, the background colour appears to be subtracted from the foreground colour. For example, suppose two identical size squares which have the same purple colour are viewed against a blue background and a red background respectively. It can be observed that the foreground purple appears more as a blue purple on the red background. Additionally, if the foreground and the background colours are opposite each another, that is they exhibit a high degree of contrast in hue, such as blue and orange, the foreground colour takes on the characteristic of the complement of the background co-

lour. For example, suppose the foreground colour is orange and the background colour is purple. The foreground orange colour seems to be intensified. Also the lightness/darkness of the background colour affects the appearance of the foreground colour. A dark background makes the foreground colour appear lighter than it is, while a light background makes the foreground colour appear darker. The saturation of the background colour has the similar effect on the foreground colour. A medium bright colour can appear more intense by being placed on a very dull background and it can also appear less intense by being placed on a more intense or more vivid background. Some other well-known effects are discussed in the followings.

The perceived colour of an area also depends on the size of the area. The colour of small area appears more saturated. Likewise, the colour of a patch affects its perceived area. The area of a saturated colour appears larger than the area of a de-saturated colour.

It has been mentioned previously that the achromatic channel carries information which is the summation from the red and green cones. Consequently, the visual system is most sensitive to the center of the spectrum and the sensitivity decreases towards the spectral extremes. This means small changes in the middle range of the spectrum (green) are easier to detect than those in the short and long ranges of the spectrums (blue and red), that is blue and red must be of greater intensity than green to be perceived.

The subjective feature of colour perception has been mentioned in chapter 3 related to colour codebook ordering. In colour codebook ordering, it is required that the codewords which represent close colours should be put close in the codebook and the codewords which are also in the codebook should represent close colours. However, because of the subjective feature of colour perception, it is quite hard or impossible to decide exactly whether two codewords represent close colours, or which codeword represents the closest colour to the colour represented by the reference codeword, just based on the tristimulous values of the codewords. This makes the colour codebook ordering complicated and degrades the ordering algorithms for colour codebook ordering.

4.2 Subjective Testing

One way of testing image quality is to judge the quality by human beings. A group of observers examine a set of images and make subjective decisions about the quality. The results of subjective testing normally depend on the observers background, the motivation for examining the images etc. There are two commonly used methods, namely the category-judgement method and the comparison method [Netr88]. In the research addressed in this thesis, the quality of the processed image is not evaluated by the methods discussed below. They are evaluated by the MSE or the SNR and visually tested by several people.

4.2.1 Category-Judgement Method (Rating Scale Method)

A panel of observers view an original image and a sequence of processed images, and assign each processed image to one of several categories. The categories may be based either on overall quality or on visibility of impairments as in table 4.1 (a) and (b) respectively.

(a)		(b)		(c)	
5	Excellent	5	Imperceptible	3	Much better
4	Good	4	Perceptible	2	Better
3	Fair	3	Slight annoying	1	Slight better
2	Poor	2	Annoying	0	Same
1	Bad	1	Very annoying	-1	Slight worse
				-2	Worse
				-3	Much worse

Table 4.1 Quality and impairment ratings

The subjective judgement depends upon many factors such as the highlight luminance, contrast ratio, image size, viewing distance, experience and motivation of the observers, and the range of the picture material etc. Many of the above variables and their effect on subjective assessment have been studied in depth, and international standardization has been generated with recommendations from the CCIR. Consequently, the results which are obtained from different observers at different times can be compared.

The results of the category-judgment procedure are normally presented by computing a Mean Opinion Score (MOS) defined as the function below (Eq.(4.1)), where c_i is the numerical value corresponding to category i, n_i is the number of judgements in that category and k is the number of categories in the scale. The MOS value is used as the evaluation result for a processed image. Normally, the MOS values are quite consistent, provided appropriate observers are chosen and the viewing conditions are proper.

$$MOS = \frac{\sum_{i=1}^{k} n_i c_i}{\sum_{i=1}^{k} n_i} \qquad (4.1)$$

Category judgement methods are popular in broadcast television for monitoring picture quality, since they can be used for setting up and maintaining an appropriate grade of service.

4.2.2 Comparison Method

An alternative method for subjective quality measurement is the comparison method. In this method, the observers compare an impaired or distorted test image with a reference image to which impairment of a standard type has been added. The comparison may be on two monitors arranged side by side or on one monitor where both images are displayed sequentially in time. Impairment is add-

ed to the reference image until both images appear to the observers to be of equal quality. The amount of the added impairment can be either under the subject's control, or alternatively a group of images with variable amount of impairment can be precomputed, stored and displayed in a given sequence.

The comparison between the test image and the reference image can usually be done accurately when the two types of distortions are visually similar, e.g., additive noise of different spectral characteristics. The distortion of the test image can be assigned a quality (or category) by utilizing the previous category-judgement test ratings on the impaired reference images.

A variation of this method is where the subject uses a comparison rating scale, e.g., table 4.1(c), to compare test images with a reference image. The observers then reply to the question "how much better or worse is the test image compared to the reference image?" using the comparison rating scale. The resulting data is then processed to determine the "point of subjective quality" between the distorted test image and the impaired reference image. One problem in these comparison methods is that transitivity among impairments may not always be hold, especially when different impairments have very different appearance. For example, suppose the following subjective equivalence has been established separately: x amount of impairment A is subjectively equivalent to y amount of impairment B; and y amount of impairment B is also subjectively equivalent to z amount of impairment C. It is not always true to conclude that x amount of impairment A is subjectively equivalent to z amount of impairment C.

4.3 Objective Evaluation of Image Quality

The objective evaluation of image quality is to establish a functional relationship between the degree of the impairment felt by observers and the measurable characteristics of the degraded image. However, a function which can predict the quality of the degraded image and is correlated with the

subjective evaluation has not been developed. This is because many aspects in image quality evaluation are a purely subjective matter and the visual processing is not fully understood. For example, viewers always concentrate their attention on particular areas of a displayed image at any one time, and large distortions in other regions of the visual field may be ignored. Or the attention of the casual observer, whose eyes roam, initially at least, at will over the displayed image, will be caught by isolated large impairments, especially if the impairments are caused by individual localized luminance errors which are correlated, i.e. form a pattern with readily recognizable structure. Furthermore, when people evaluate an image, they normally compare it with what such an image "ought to" look like rather than with the original image.

Some models for image quality have been proposed, either in the space domain or frequency domain [Netr88]. They are consistent with many physiological facts and psychophysical experiments, such as non-linearity, visual masking etc. But they cannot be uniquely derived based on experimental data and are too complex to be directly useful at present for optimizing coding algorithms.

The methods used frequently in practical applications are the Mean Squared Error (MSE) or the Signal to Noise Ratio (SNR). Suppose the image resolution is N × N, the MSE between the original image and the distorted image is computed as follows,

$$MSE = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} E\,(f_{ij} - f'_{ij}) \qquad (4.2)$$

where f_{ij}, f'_{ij} are the value of the original and the processed data respectively and E() is expectation. Experimentally, the MSE is often estimated by the average sample mean squared error in the given image defined by:

$$MSE \approx \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} (f_{ij} - f'_{ij})^2 \qquad (4.3)$$

The SNR used in image processing is:

$$SNR = \left(\frac{\frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f_{ij}^2}{\sqrt{MSE}} \right)^{1/2} \qquad (4.4)$$

The MSE and the SNR are very simple to compute and are widely used, particularly in mathematical studies, but they show weak correlation with the judgement of human observers. The reason for this is that they do not consider the aspect of the human observer. For example, the human visual system does not process the image in a point-to-point way but extracts certain spatial and temporal features for neural coding, while the MSE or the SNR are the summation of the distortion of point-by-point without any consideration of their spatial relationship. Therefore, it is quite possible that the processed image has a poor quality but a very small MSE or SNR value.

For color images, one typical method for the distortion computation is, first compute the distortion of three component images separately in the same way as the computation on the black and white image, and then compute the summation of the three distortion values as the final result. Such independent processing of the three component images does not consider the interaction of the three colour components.

Chapter 5

Image Compression

This chapter deals with the technology for image compression. The goal of image compression is to reduce the amount of data in the image in order to minimize the memory for storage and the bandwidth for transmission. A large variety of techniques for image compression has been developed and is reviewed in section 5.1. Image compression techniques can be divided into two main classes, namely information lossless and information lossy techniques. The former is able to reconstruct the original image without any loss of information whereas the later introduces some distortion in the reconstructed image which should not be perceived by human eyes. In this chapter, the Discrete Cosine Transform (DCT), which is an information lossy technique, is discussed in section 5.2. The conventional colour image compression techniques are studied in section 5.3. Finally the Joint Photographic Expert Group (JPEG) still image compression standard is described in section 5.4.

5.1 Review of Image Compression

In image transmission and storage, digital techniques instead of analog are increasingly used due to the rapid growth in the use of digital computers, the declining cost of digital processing and transmitting equipments. This is also because the digital transmission and storage system has many inherent advantages over the analog system, such as easy processing, processing flexibility, easy

and random access in storage, higher signal-to-noise ratio (SNR), possibility of errorless transmission etc. However, images, whether digital or analog, contain a large amount of information and require wideband channels for transmission and big memory for storage, especially digital images. For example, a 4 MHz television signal sampled at Nyquist rate with 8 bit samples could require a bandwidth of 64 MHz in transmitting. Therefore it is highly desirable to compress image data for transmission and storage. A lot of techniques for digital image compression [Jain88] [Netr88] have been developed.

The statistical properties existing in images are the main reasons that images can be compressed. The statistical property upon which intraframe coding techniques rely is the high correlation between neighbouring pixels.This means that adjacent pixels are usually similar to one another and the magnitude of a pixel may be estimated from the values of the pixels around it. Most images, even fairly "active" images which contain a large amount of spatial detail, have quite high values of correlation. The interframe coding is based on the fact that there exists a large amount of frame-to-frame correlation in moving images, which is also called temporal correlation. For example, in moving images, the background is likely to remain stationary in successive frames. The correlations in one frame of an image or successive frames are image data redundancies which can be reduced without apparent degradation of image quality. In this thesis, only the intraframe compression techniques are discussed.

The image compression techniques can be classified into two classes, namely information lossless and information lossy techniques. The former is able to reconstruct the original image without any loss of information, whereas the later introduces some distortion in the reconstructed image and cannot recover the original image exactly. Lossless and lossy compression techniques are used in different applications. For example, medical images often require completely lossless compression because any slight distortion may result in wrong diagnosis. In other applications, such as entertainment, education etc., the reconstructed images need not necessarily be exactly the same as the original ones and lossy compression techniques are then widely used. The lossless techniques normally reach lower compression ratio while the lossy techniques can reach higher compression ra-

tio. Next we will discuss the monochrome image compression techniques for intraframe coding.

5.1.1 Predictive Coding

Predictive coding is a popular technique for image compression. Since in a typical image, the pixel sequence has statistical dependency from one pixel to the next, then the value of a given pixel may be predicted from its preceding coded neighbouring pixels. On the encoding side, the code of a given pixel is the difference between the pixel value and its predicted value. On the decoding side, the original pixel is recovered by the difference signal together with the predictor which is carried out in the same way as that in the encoding.

The predictive coding is relatively simple and easy to implement. It has low complexity both in memory requirement and number of operations. Using adaptive systems, predictive coding can give good quality images in the 1 to 2 bits/pixel range. However, the predictive coding is quite sensitive to channel errors. For example, a code loss will result in the loss of the rest of the codes if no extra counter measures are used, and an error code will introduce distortion to all the codes after it. Also, it depends on the variation in input data statistics.

5.1.2 Transform Coding

The basic motivation behind transform coding [Wint72] [Clar85] is to transform the image from the data domain to a frequency domain by an energy preserving unitary transform. In the frequency domain, the image pixels are de-correlated and the energy is concentrated on a few coefficients so that the high frequency coefficients and the coefficients with less energy, can be removed without any visual effect on the reconstructed image, since they play less important roles in the image re-

construction. The transform could be applied to the entire image but implementation problems make this impractical. First, the amount of the memory and the computation required increase proportionally to M^2, where M is the image dimension. Second, because of the elimination of unimportant coefficients, large transform size often leads to more significant degradation than small size. A typical approach is to divide the image into a number of rectangular blocks or sub-images, normally 8×8 or 16×16, and then an unitary transform is applied to each sub-image.

After the transformation, the image compression in the transform coding is achieved by quantizing the transform coefficients. If all the coefficients are quantized and coded, the compression ratio is quite small. It has been pointed out that the important characteristic of the transform is that most energy of the image is packed into a small number of low frequency coefficients and the coefficients with less energy or the high frequency coefficients play less important roles in the image reconstruction. To achieve higher compression, one possibility is to use a mask covering an area of low frequency coefficients and to discard the remaining coefficients, i.e. set the remaining coefficients to zeroes. Only those coefficients in the mask are quantized and coded. Considerable compression can be achieved depending on the size of the mask. This technique is known as zonal coding. Another possibility is to use a threshold on each transform coefficient and set the coefficients which are below the threshold value to zero. The remaining non-zero coefficients together with their address information are quantized and finally coded efficiently by coding schemes such as the Huffman coding [Huff52] or the arithmetic coding[Witt87]. For better subjective image quality, the quantizer in both cases should be designed to optimize the reconstructed image quality for a given number of bits.

A number of orthogonal transforms can be used in the transform coding and most of them are linear transformations.

1. The Karhunen-Loève Transform (KLT)

The Karhunen-Loève transform [Clar85] is the best linear transformation in the sense of

completely de-correlating the data and maximizing the amount of energy compacted into the lowest-order coefficients. However, it is not certain that the KLT is the absolutely optimum transform since it does not consider other factors such as the human visual system. Additionally, the transform matrix depends on the image data, i.e., the transform matrix is different for different image data. Thus, the KLT transform matrices are also transmitted and stored along with the coded data. Furthermore, the amount of the computation in the transform matrix generation is very large and the KLT has no fast transformation associated with it.

Because of the computation complexity, the large amount of storage requirement and the dependence on the input images, the KLT is seldom used in practice but is employed in theoretical studies of image coding. It gives an indication about the upper-bound of what other transformations, computationally more efficient, should attempt to reach for de-correlating data samples.

2. The Discrete Fourier Transform (DFT)

The discrete Fourier transform [Clar85] is naturally applied to image coding because of its widespread use in other signal processing fields and the fact that it has efficient computational algorithms and fast implementation. It is the only complex transform used in data coding schemes. The DFT is not convenient for general use due to the necessity to process both real and imaginary components, which requires a large number of operations and large storage.

3. The Discrete Cosine Transform (DCT)

The discrete cosine transform [Ahme74] is one of an extensive family of sinusoidal transforms. Compared with other orthogonal transforms, the DCT has the best all-around performance with respect to efficient computation and acceptable perceptual quality for a

given compression rate. It is widely used in image compression and is adapted in the JPEG still-image compression standard. The DCT will be discussed in section 5.2.

4. The Walsh-Hadamard Transform (WHT)

The Walsh-Hadamard transform [Prat81][Clar85] is the simplest transform among various types of orthogonal transforms. The elements in the transform matrix only consists of 1 and -1, and the only multiplication needed is that of the final scaling operation. However, it is too simple to compact energy well.

Transform coding can achieve higher compression ratios than the predictive coding and its compression ratio is normally 10:1. It also has better immunity to channel noise. In the transform coding, a code error in transmission only influences the corresponding block and has no effect on the succeeding blocks because this error is distributed by the reverse transform over the entire block. Visually, a code error in the transform coding is less visible than that in predictive coding. However, transform coding has some defects. First, since the image is divided into blocks, the block to block correlation is not employed in the transform. Furthermore, artificial blocking segments the image arbitrarily without considering its contents. Second, in the transform coding at low bit rate, sometimes the so-called blocking effects and mosquito effects are apparent in the reconstructed image. Blocking effects are perceived in the reconstructed image as visible discontinuity between adjacent blocks. Mosquito effects are the fluctuation of luminance/chrominance within blocks. These are especially visible around the boundaries of moving objects and still background. Both effects are caused by the improper coding of the transform coefficients, such as eliminating too many coefficients or the coarse quantization. Finally, transform coding needs more operations and memory than predictive coding. This is improved due to the rapidly decreasing cost of digital hardware and computer memory, and this may no longer be a disadvantage. Many transforms, such as the DCT, the DFT etc., have been hardware implemented to achieve higher speed performance.

5.1.3 Hybrid Coding

Hybrid coding [Habi74] is a kind of technique which combines transform coding and predictive coding together to generate a better coding scheme. It takes the advantages of transform coding and predictive coding and overcomes their shortcomings to a certain degree. Typically, in a hybrid coding, a two-dimensional image is unitarily transformed in one of its dimension to obtain a sequence of one dimensional sequences. Each of these sequences is then coded independently by a one dimensional predictive technique, such as the DPCM.

Generally, hybrid coding performance lies between transform coding and predictive coding. It is easily adaptable to coding images and changes in data statistics. It is less sensitive to channel errors than predictive coding.

5.1.4 Vector Quantization (VQ)

According to Shannon's rate distortion theory, a better performance is always achievable in theory by coding vectors instead of scalars. Though predictive coding and transform coding usually use the psychovisual as well as statistical redundancy in the image data to reduce the data rate, one deficiency of these coding schemes is that quantization is performed on individual samples of image pixels and lower compression is obtained. Vector quantization [Gers82] [Gray84], however, is one of the coding techniques that could approach the rate distortion limit and achieve high compression rate. The principle of vector quantization has been discussed in section 2.1. Here, vector quantization is discussed in the context of image compression.

In vector quantization, compression is achieved by using a relatively small codebook to approximate the gamut of the image vectors. Let n_p be the number of bits for one pixel, N_c be the size of the codebook and n be the dimensions of the vector. The resulting bit rate of the vector quantization

scheme is $R = (\log_2 N_c)/n$ bits/pixel and the compression ratio is R/n_p, that is $(\log_2 N_c)/nn_p$. In theory, vector quantization can achieve performance close to the rate-distortion bound as $n \rightarrow \infty$, however, large vector dimensions will make codebook storage and searching impractical. Fortunately, reasonable performance can be achieved with modest n, say in the range of 4.

Vector quantization can also be used together with other image compression techniques to form more powerful and efficient image compression techniques, such as predictive vector quantization, transform vector quantization [Mare86][Maen89]etc. In predictive vector quantization coding scheme, the encoder consists of a vector predictor and an error vector quantizer. The error vector is coded by vector quantization. The idea of predictive vector quantization can also be extended to interframe DPCM coders. In transform vector quantization coding scheme, the vector quantization is applied to the transform domain. It is believed that using vector quantization rather than scalar quantization on transform coefficients can achieve better compression. Besides, since some of the high-frequency coefficients are discarded, the computational cost can be reduced. For colour images, a VQ/DCT coding scheme [Wang91] is developed, which will be discussed in the later chapters.

Vector quantization has been successfully applied to digital image coding at low bit rates. For example, for monochrome images, the rate is in the range of 0.5-1.5 bits/pixel. Besides, the particularly simple table looking-up decoding procedure makes vector quantization an attractive method of data compression in practice. However, the simple vector quantization system suffers from some defects. First, edges are poorly reconstructed and the codeword edges make the reconstructed image appear "blocky" because the codebook cannot reproduce all the possible patterns in the image. This situation can be improved by constructing composite codewords, that is to have separate codebooks for the edge information and the texture information [Gers82]. Secondly, vector quantization codes are very sensitive to errors because a code error will result in a wrongly reconstructed vector. Thirdly, the codebook generation and the coding procedure need a lot of computation. Finally, the distortion measure used normally is the Mean Squared Error (MSE) measure which does not correlate well with perceived quality. Better quality for vector quantization could probably be

achieved with a more complicated distortion measure rather than the MSE distortion measure.

5.1.5 Synthetic High Coding

The synthetic high coding [Kunt85][Kunt87] exploits the properties of the human visual system to achieve a considerable amount of redundancy reduction. In the synthetic high coding system, the image is split into two parts: the low-pass image giving the general area brightness without sharp contours and the high-pass image containing sharp edge information. The low-pass image can be represented by very few samples. The high-pass image is characterized by the location, magnitude and argument of each selected edge-point gradient. These are coded for storage and transmission. Therefore, the coded image is represented by four sets of data, low-pass image, contouring direction changes, gradient direction changes and gradient magnitude. The low-pass image is coded by transform coding. Because there is no sharp edge in the low-pass image, the weakness in coding edges in transform coding is no longer a problem here and transform coding can work very efficiently. The edge information can be coded by an optimum Huffman code derived from their distribution. At the decoding side, the image is reconstructed by the low-pass image and a two-dimensional reconstruction filter. The reconstruction filter, whose properties are determined uniquely by the low-pass filter for the low-pass image, is used to synthesize the high frequency part from the edge information. The synthetic high coding can achieve higher image compression ratio, up to 70:1. But it needs a tremendous amount of computation which makes it inappropriate for real-time applications.

5.1.6 Discussion

A large number of image compression techniques have been developed for digital image transmis-

sion and storage. They have been efficiently used in real-time applications due to the development of computers, both in the power and the speed, and micro-electronic technologies. Many coding techniques, which used to be thought too complex for real time applications, such as the DCT, can be hardware implemented by small fast DSP chips.

However, most of image compression techniques, such as predictive coding and transform coding, are mainly based on the statistical properties in the image data to achieve redundancy reduction with less consideration of the properties of the human visual system. With the progress in the study of the brain mechanism of vision, new methods exploiting the vision properties for image compression to achieve higher compression ratio are being studied intensively [Kumt85] [Kunt87]. In this kind of methods, an image is described by contour and texture and then the contour and the texture are encoded. This kind of image description is more natural since it coincides with the psychological conception of vision. The synthetic high coding system is an example. This group of methods can reach high compression ratios, some of them up to 100:1. The main problem associated with them is the large amount of processing. They are still in the research stage.

5.2 The Discrete Cosine Transform (DCT)

The discrete cosine transform [Ahme74] was proposed by Ahmend et al. in 1974. At that time, there was increasing interest in the class of orthogonal transforms, such as the discrete Fourier transform, the Hadamard transform, in the general area of digital signal processing, such as image coding, pattern recognition etc. It is known that the KLT is the optimal transform with respect to performance measure, but it needs a large amount of computation and has no fast transform. Therefore, researchers tried to develop a transform which is close to the performance of the KLT and has fast algorithms. To fill the role, the discrete cosine transform was proposed.

The two-dimensional discrete cosine transform of a data sequence $f(i, j)$ for i, j = 0, 1,..., N - 1, is

defined as:

$$F(u,v) = \frac{4c(u)\,c(v)}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f(i,j)\cos\frac{(2i+1)\,u\pi}{2N}\cos\frac{(2j+1)\,v\pi}{2N} \qquad (5.1)$$

for u, v = 0, 1,..., N-1, where

$$c(\omega) = \begin{cases} \frac{1}{\sqrt{2}} & for \quad \omega = 0 \\ 1 & for \quad \omega = 1, \ldots, N-1 \end{cases}$$

The inverse two-dimensional discrete cosine transform is defined as:

$$f(i,j) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} c(u)\,c(v)\,F(u,v)\cos\frac{(2i+1)\,u\pi}{2N}\cos\frac{(2j+1)\,v\pi}{2N} \qquad (5.2)$$

for i, j = 0, 1,..., N - 1.

It has been shown that the performance of the DCT is nearly identical to the KLT transform for blocks of reasonable large size [Tsen78]. Furthermore, the empirical evidence shows that even for blocks of small size the performances of the DCT and the KLT are close. Additionally, experiments show that the DCT [Mako85] is better than other transforms, such as the Hadamand transform etc. in respect to the minimum block distortion and good energy compaction.

Since the computation in the DCT by software is quite large for time critic applications, fast versions of the DCT [Chen77][Kama82][Makh87] have been proposed. Though speed performance is improved by the fast algorithms, the fast algorithms still require a large amount of computation. Now the DCT can be hardware implemented by digital signal processor to achieve high speed at reasonable cost.

5.3 Conventional Colour Image Compression

Typically, a colour image is composed of three bands, corresponding to the three colour primaries, namely Red, Green and Blue. Colour image compression can be carried out on the Red, Green and Blue bands independently using the techniques for monochrome image compression. However, this simple approach does not achieve very efficient compression. Because there is considerable correlation among the Red, Green and Blue components, especially for natural pictures. This means that pixels with similar values of Red, Green and Blue components occur frequently in the image. This redundancy cannot be removed by the operations on the Red, Green and Blue bands independently. Alternatively, the Red, Green and Blue primaries can be firstly transformed into alternate colour primaries to decorrelate the correlation in the Red, Green and Blue primaries and the compression techniques are then applied to the transformed primary bands independently. In most colour image compression schemes [Eina87] [Limb77][Mitr89], the RGB coordinates are normally transformed into the YUV coordinates using the following transformation:

$$\begin{bmatrix} Y(i,j) \\ U(i,j) \\ V(i,j) \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R(i,j) \\ G(i,j) \\ B(i,j) \end{bmatrix}$$

and its reverse transformation from the YUV back to the RGB is:

$$\begin{bmatrix} R(i,j) \\ G(i,j) \\ B(i,j) \end{bmatrix} = \begin{bmatrix} 1.000 & 0.956 & 0.621 \\ 1.000 & -0.272 & -0.647 \\ 1.000 & -1.106 & 1.703 \end{bmatrix} \begin{bmatrix} Y(i,j) \\ U(i,j) \\ V(i,j) \end{bmatrix}$$

In the Y, U, V representation, most energy from the Red, Green and Blue components is packed on Y (as much as 93%) and significant less energy on U (about 5%) and V (about 2%). The Y component contains luminance information and the U, V components contain chrominance information. Since the U and V components carry less energy, the U and V bands are often spatially averaged and sub-sampled by a factor 2:1 or 4:1 in both horizontal and vertical directions without noticeable effect on the image quality. At the receiver side, the reconstructed U and V bands are interpolated back to the original size. Another colour primary transform which is similar to the YUV primaries but easier to implement in hardware is: Y, R - Y and B - Y components. These two

kinds of colour primaries have no significant difference from the compression point of view.

In conventional colour image compression scheme, the Red, Green and Blue primaries are first transformed to a proper primaries which can compact energy well, such as the YUV primaries. Then the image compression techniques are applied to the three transformed primary images separately in the same way as to a black and white image.

5.4 The JPEG Still Picture Compression Standard

To promote widespread application of digital image transmission and storage in the market place, an image compression standard is required. An international team of technical experts, called the Joint Photographic Experts Group (JPEG), has been working for the past few years on the first international digital image compression standard for continuous-tone (multilevel) still images both gray scale and colour [Huan89][Wall91][Mitc91]. The JPEG aim was to develop a compression standard which meets the following requirements: a) be at or near the state of the art compression performance; b) be applicable to the widest range of applications and consistent with a number of predetermined constraints; c) have tractable computational complexity; d) have the following modes of operation: sequential encoding, progressive encoding, lossless encoding and hierarchial encoding.

The JPEG standard describes a family of image compression techniques. It provides a toolkit of image compression techniques which can be selected for a wide range of applications with different requirements. There are four main modes of operation in the JPEG toolkit, which provide the framework for implementation of both lossy and lossless coding processes. They are sequential DCT-based mode; the progressive DCT-based mode; the sequential lossless mode; and the hierarchical mode. In the remaining part of this section, the main focus is on the DCT-based coding which is a key part in the standard and many of the techniques used in it are also employed in other

modes of operation. The block diagram of the DCT-based coding is shown in figure 5.1.

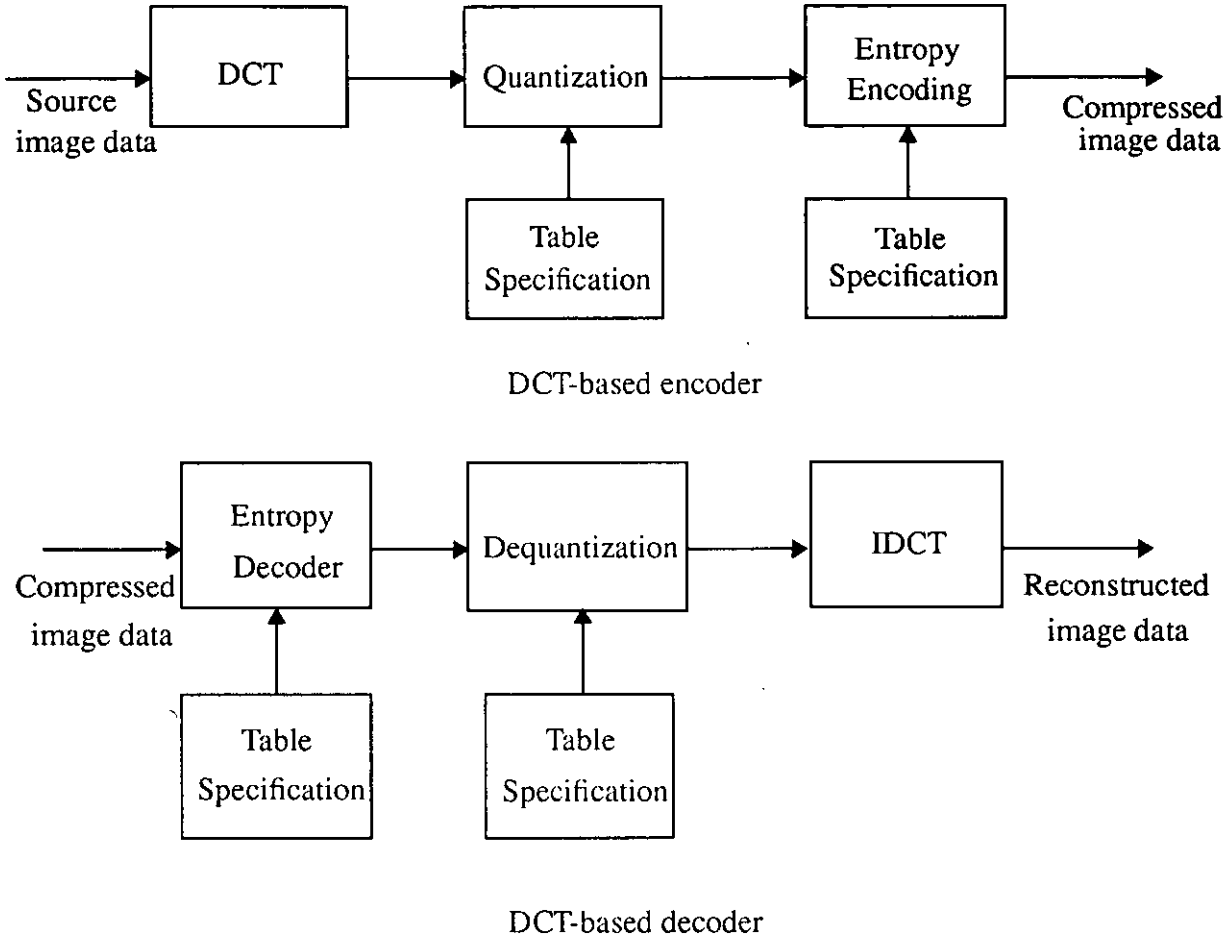


Figure 5.1 The diagram of the DCT-based coding

The main processing steps in the DCT-based coding are explained as follows:

- **8×8 FDCT and IDCT**

The FDCT (Forward Discrete Cosine Transform) is applied to streams of blocks of 8×8 pixels and outputs 64 DCT coefficients. The coefficient with zero frequency in both dimensions is called the DC coefficients and the remaining 63 coefficients are called AC coefficients which represent the values of different frequency changes. The FDCT concentrates most energy on the lower spatial frequencies, which provides the possibility

of data compression. The IDCT (Inverse Discrete Cosine Transform) is applied to 64 DCT coefficients and reconstructs a 64-point image signal.

• Quantization

The compression is achieved by the process of quantization on the FDCT coefficients. Quantization is used to discard information which is not visually significant in the image reconstruction. Each of the 64 DCT coefficients is quantized in conjunction with a 64-element quantization table, that is each of the 64 coefficients is divided by its corresponding threshold in the quantization table, followed by rounding to the nearest integer:

$$F^{Q}(u,v) = Integer\ Round\left(\frac{F(u,v)}{Q(u,v)}\right) \qquad (5.3)$$

The quantization tables recommended by the Committee are given in tables 5.1 and 5.2,

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Table 5.1 Luminance quantization table

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Table 5.2 Chrominance quantization table

- **Zigzag scan order**

The 8 × 8 block of the DCT coefficients is reorganized into a one-dimensional list in the order of zigzag sequence shown in table 5.3, so that the lower frequency coefficients are concentrated at lower indexes to facilitate later encoding. Because of the important role of the DC coefficient in the image reconstruction, it is coded in a different way. The DC coefficient from the preceding 8 × 8 block is used as a predictor for the DC value of the current block and either the Huffman coding or the arithmetic coding is employed to code the difference between the DC value of current block and the predictor.

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

Table 5.3 Zigzag order of an 8 × 8 matrix

- **Entropy coding**

Entropy coding can achieve additional lossless compression by encoding the quantized DCT coefficients more compactly based on their statistical characteristics. The JPEG proposal specifies two entropy coding methods, the Huffman coding [Huff52] and the arithmetic coding [Lang87][Witt87].

The entropy coding can be considered as a 2-step process. The first step converts the zigzag sequence of quantized coefficients into an intermediate sequence of symbols. The second step encodes the symbols in a data stream. The intermediate symbol sequence for AC coefficients is composed of two parts, symbol-1 and symbol-2:

symbol-1: (RUNLENGTH, SIZE)

symbol-2: (AMPLITUDE)

where the RUNLENGTH is the number of consecutive zero value AC coefficients in the zigzag sequence, SIZE is the number of bits used to encode AMPLITUDE, and AMPLITUDE is the value of a non-zero coefficient. At the end of each block, there is a terminating symbol [EOB] to signify the end of the block. RUNLENGTH uses 4 bits and represents a value from 1 to 15. If there are more than 15 consecutive zeroes, symbol-1 is represented as (15, 0) which is interpreted as the extension symbol with RUNLENGTH = 16. Therefore, for each 8×8 block of pixels, the zigzag sequence of 63 quantized coefficients is represented as a sequence of symbol-1 and symbol-2 pairs, and each pair can have repetitions of symbol-1 in the case of a long run-length or only one symbol-1 in the case of an EOB.

The numerical analysis of the 8×8 FDCT equation shows that, if the 8×8 block input pixel is N-bit integers, the non-fractional part of the output coefficients can grow by at most 3 bits. Therefore, in the 8×8 block FDCT with 8-bit pixels, the quantized AC coefficient amplitude is in the range $[-2^{10}, 2^{10} - 1]$. Then the symbol-2 AMPLITUDE code uses 1 to 10 bit length to represent signed integer in the range $[-2^{10}, 2^{10} - 1]$ and the SIZE

represents value from 1 to 10.

The intermediate symbol sequence for the DC coefficient is:

symbol-1: (SIZE)

symbol-2: (AMPLITUDE)

Because the DC coefficient is differentially encoded, it is in the range $[-2^{11}, 2^{11} - 1]$. The symbol-2 AMPLITUDE codes uses 1 to 11 bits and symbol-1 SIZE represents a value from 1 to 11.

For both DC and AC coefficients, each symbol-1 is encoded with a variable length code (VLC) from the Huffman table, which must be specified as an input to the encoder. Each symbol-2 is encoded with a "variable length" integer (VLI) code whose length in bits is given in table 5.4.

SIZE	AMPLITUDE
1	-1, 1
2	-3, -2, 2, 3
3	-7,...,-4, 4,..., 7
4	-15,..., -8, 8,..., 15
5	-31,..., -16, 16,..., 31
6	-63,..., -32, 32,..., 63
7	-127,..., -64, 64,..., 127
8	-255,..., -128, 128,..., 255
9	-511,..., -256, 256,..., 511
10	-1023,..., -512, 512,..., 1023

Table 5.4 The DCT coefficient arithmetic coding

Chapter 6

Experimental Hardware Environment

This chapter is about the hardware environment of the experimental system. First, it gives the whole structure of the experimental system (§ 6.1). After a description of the transputer (§ 6.2.1), the Inmos A121 Discrete Cosine Transform chip (§ 6.2.2) and the FIFO chip (§ 6.2.3), it presents the design and implementation of the DCT board (§ 6.2.4). It also introduces the MicroEye TC colour image capture and display board (§ 6.3).

6.1 System Structure

The system is a transputer-based parallel image processing system, which can be used to compress and decompress colour images by image compression techniques, such as the discrete cosine transform, vector quantization etc., or to carry out image processing, such as image segmentation, edge detection etc. Figure 6.1 provides an overview of the whole system. It consists of an IBM PC, a B004 transputer board, a discrete cosine transform board, a colour image capture and display board (MicroEye TC), a colour image display monitor and a video camera.

The B004 board is a PC plug in board, which mainly contains an Inmos transputer, such as T414 or T425 32-bit processor, and 2 Mbytes RAM. It provides four transputer links, where one link, normally link 0, is used in the interface with the PC, and the other three remaining links are left for

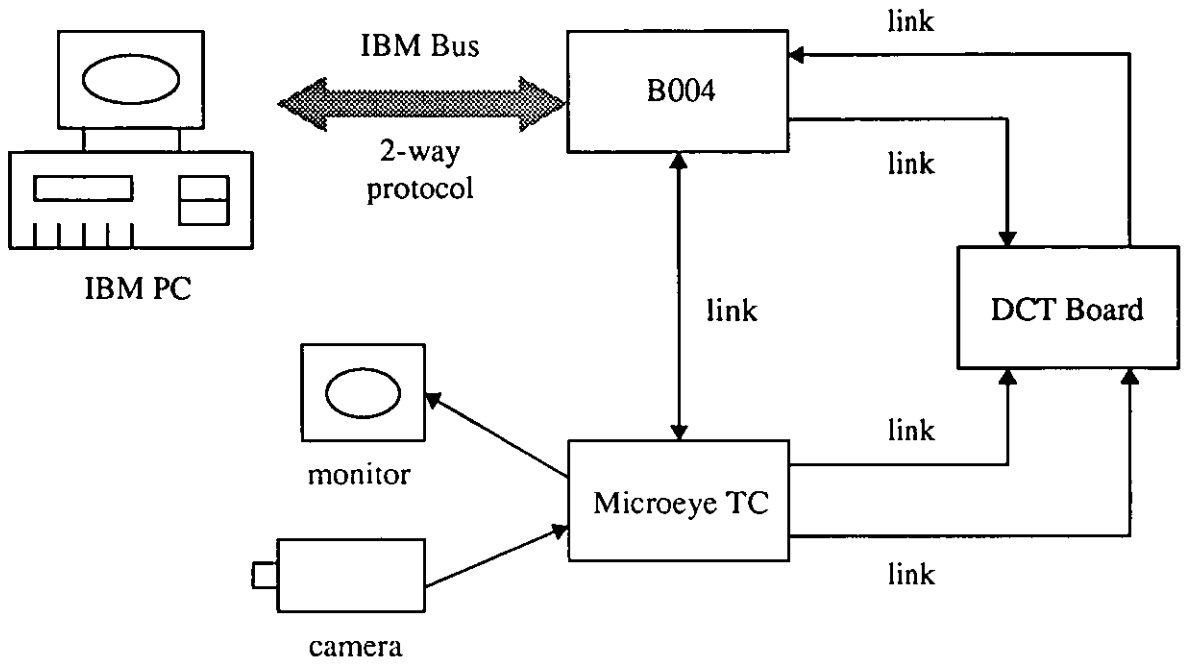


Figure 6.1 Experimental system structure

the expansion of the system by users. It runs the Tansputer Development System (TDS) [Inmo88], which provides an Occam application program development environment, and gives access to the PC resources. In the experimental system, the B004 board is combined with the PC as the host computer, while the DCT board and the MicroEye TC board are its transputer network nodes. The B004 board boots the DCT board and the MicroEye TC board and loads their corresponding codes before the whole system works. It performs most of the functions in the system. For example, it implements colour image vector quantization in the colour image VQ/DCT coding scheme and colour image edge detection in the colour image processing system.

The MicroEye TC board is a high speed colour video capture card and framestore using transputer technology, more details can be found in section 6.3. It provides an interface between the B004 and the image input and output equipments in the experimental system. That is, it grabs an image from the camera and sends the image to the B004 for processing, or sends the processed image from the B004 to the monitor for display. It can also be used as the B004 link extension to exchange messages between the B004 and the DCT board due to the limited number of links available on the

B004 board. It can, for example, transmit some side information, like the threshold for the DCT coefficients, from the B004 to the DCT board.

The DCT Board is specifically designed to implement the discrete cosine transform. It has high precision and fast speed owing to the use of discrete cosine transform chips. It provides four transputer links as interfaces to the other transputer-based systems. It receives blocks of image data from the B004 for the DCT coding and sends reconstructed image data to the B004 by transputer links.

The B004 board, the DCT board and the MicroEye TC board work in parallel and the message exchanges among them are made through fast transputer links. The link connection also makes the system architecture flexible in applications since different functions can be realized by only changing the link software configuration. For example, when it is used to perform the colour image vector quantization or image processing, the link software configuration is set in the way of figure 6.2, and when it is used to implement the colour image VQ/DCT coding, the link software configuration is set in the way of figure 6.3, while the hardware configuration need not be changed in both cases.

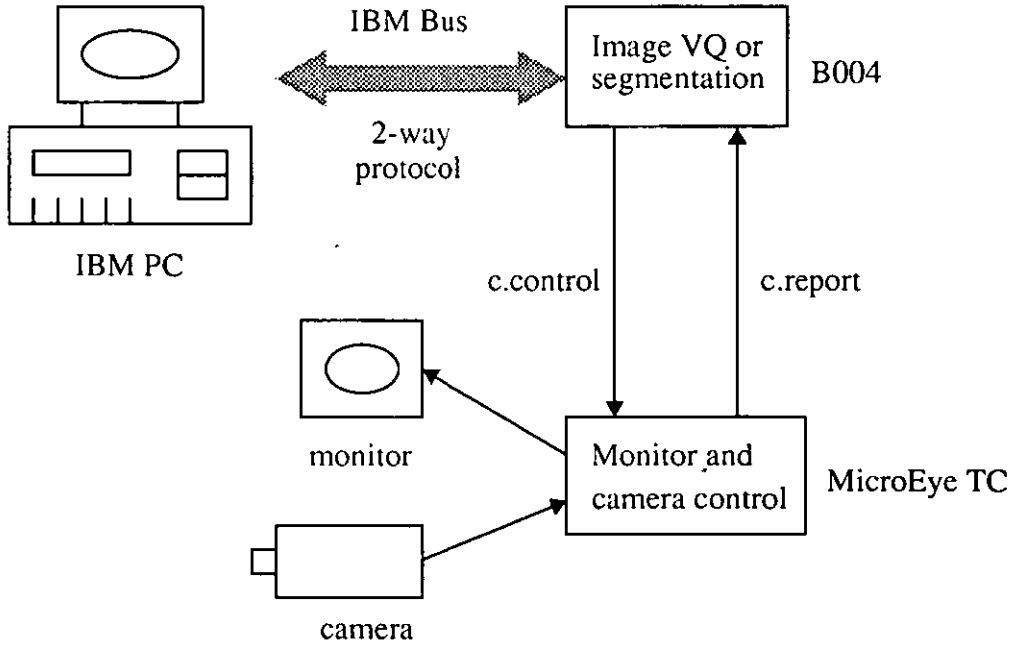


Figure 6.2 Configuration for image VQ or other processing

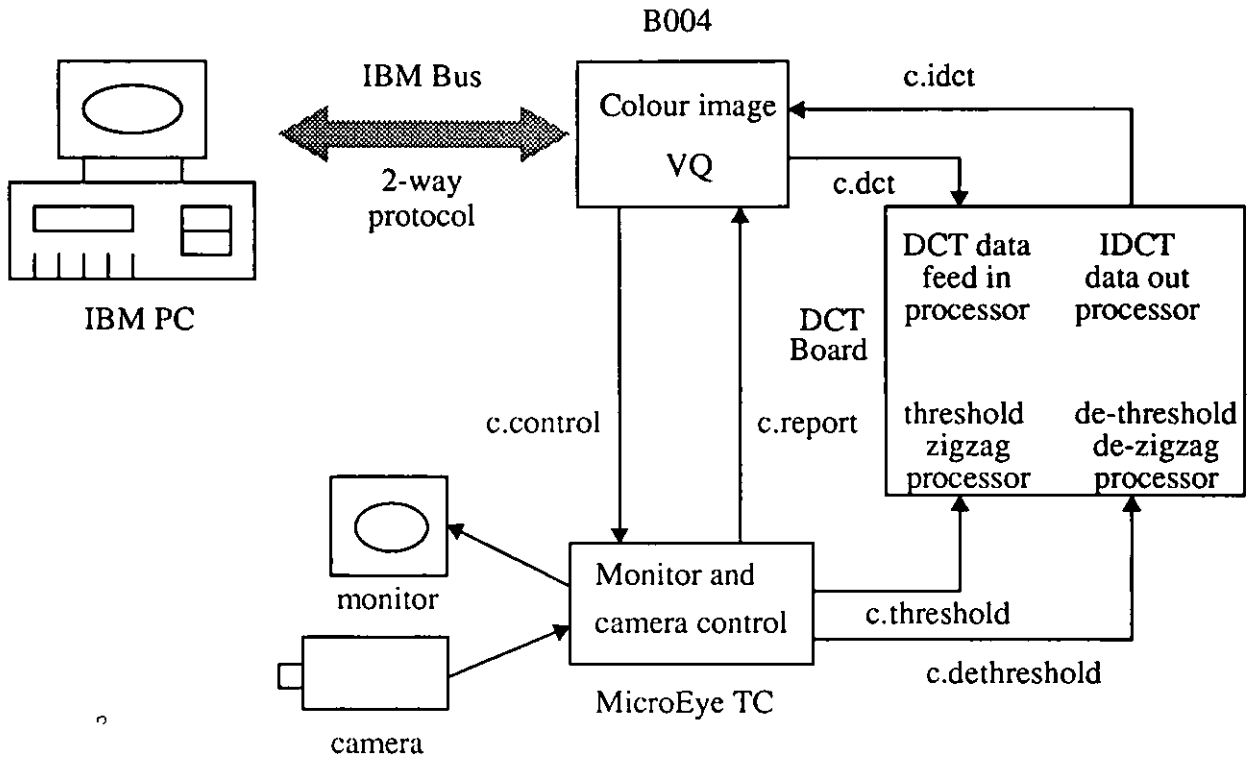


Figure 6.3 Configuration for colour image VQ/DCT coding

6.2 The DCT Board

The DCT board is designed for two reasons. First, though the Discrete Cosine Transform (DCT) coding on images can be accomplished in software, the speed on a 512×512 colour image is very slow, about 10 minutes. The speed can be improved by the fast DCT, but it is still not quick enough. In the experimental system, since the DCT processing is only used as a test method for colour codebook ordering, it is inefficient to consume most of the time on the DCT coding rather than on colour codebook ordering. Second, since the index image (see § 3.2) is very sensitive to any errors, it is required to keep the distortion caused by the computation precision in the DCT processing as small

as possible. The DCT board is then designed to implement the DCT with high speed and precision. In this section, first the main chips used in the DCT board are introduced. Then the design of the DCT board is discussed.

6.2.1 Transputer

The transputer [Inmos89[1]] is a high performance microprocessor with on-chip memory and four or two communication links for point-to-point direct connection with other transputers, see figure 6.4. The transputer can also access external memory space, 4 Gbytes for the T800, T425 and T414, and 64 Kbytes for the T222 and T212. The on-chip memory and the external memory are part of the same linear address space.

Much of the power of the transputer lies in its link structure, 2 links for the T400 and 4 links for the others. The independent bi-directional high speed Inmos serial links provide point-to-point synchronized communication with other processors with bit rate of 5, 10, or 20 Mbps (Million bits per second). Thus the transputer has the power of both data processing and data transferring, while conventional microprocessors just have the former power. Its link structure facilitates inter-process communications and supports parallel and concurrent applications. It is easy to construct a network of transputers for parallel and concurrent systems using point-to-point link communication without external complicated logic circuits. The point-to-point link communication also saves the time and the cost of system design. Moreover, the network of transputers is flexible and extensible for diverse processing requirements. Arbitrarily large systems can be constructed and more processing power can be added by only adding more transputers connected by links. Additionally, transputer links have DMA (Direct Memory Access) interface into memory to allow the data transfer to be carried out with the minimum processor intervention. The DMA communications are particular useful and efficient in transferring large blocks of data, such as image data.

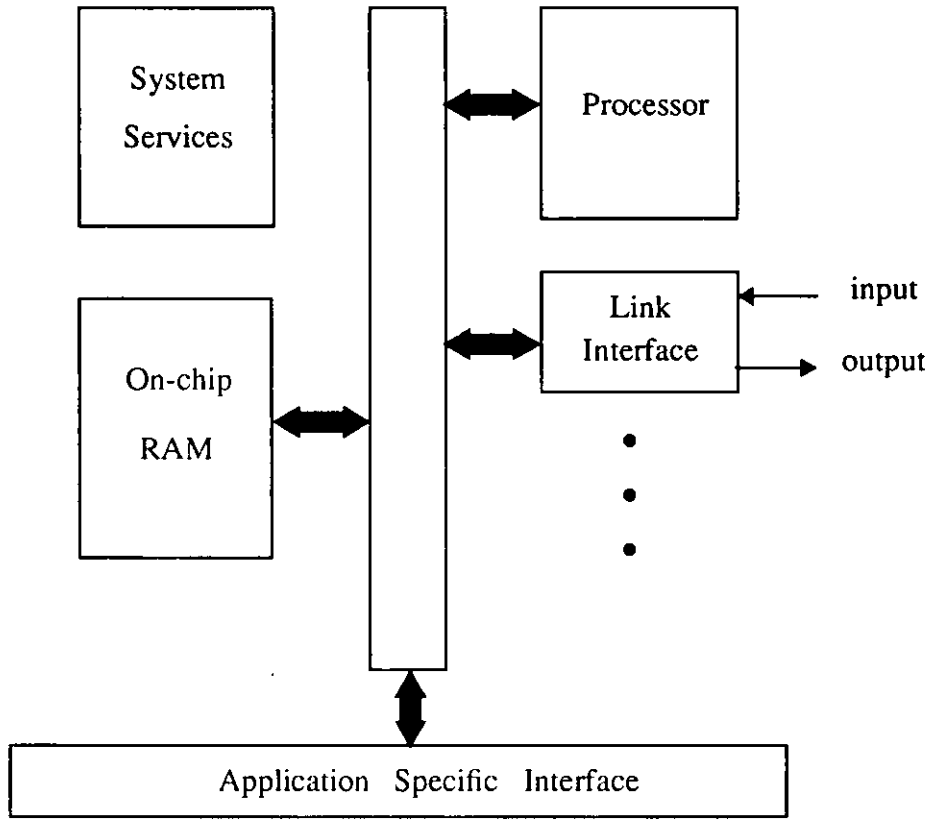


Figure 6.4 The transputer architecture

The other advantage of the transputer is its on-chip memory, 2 Kbytes or 4 Kbytes. The on-chip memory is fast static memory with high rates of data throughput. It has a higher sustained data rate than external memory. For the T222, for example, its internal memory sustained date rate is 40 Mbytes/sec, while the external memory sustained data rate is 20 Mbytes/sec. When memory requirement is less than the size of the on-chip memory, external memory can be saved, which simplifies the system design and reduces the cost of the system without the trials and complications of memory interface circuits.

To gain most benefits from the transputer, the whole system can be designed in language Occam [Burn88]. Occam is a concurrent high level language which is designed to support fully the transputer construct. It has the communication structure which uses the link features of the transputer

and its many instructions are based on the features of the transputer. On the other hand, all transputers contain special instructions and hardware to provide maximum performance and optimal implementations of the Occam model of concurrence and communications. All transputer instruction sets are designed to enable simple, direct and efficient compilation of Occam.

The transputer family includes the 16 bit Inmos T212 and Inmos T222, the 32 bit Inmos T414 and Inmos T425, and the Inmos T800, a 32 bit transputer with an integral high speed floating point processor. The T222 is used in the DCT board and its features can be found in appendix A.

6.2.2 The Inmos A121 DCT Chip

The Inmos A121 [Inmos89[2]] is a digital signal processing device which is designed to provide high speed computation of a 2-dimensional Discrete Cosine Transform (DCT) or Inverse Discrete Cosine Transform (IDCT) at video rates for image processing. The pixel rate of the A121 can reach up to 20MHz, that is, for a block of 8×8 data, the 2-dimensional DCT calculation time can be less than 3.2 μs. It can then be used in image data compression and decompression, e.g. video codecs.

The Inmos A121 computes on blocks of data which have fixed size of 64 samples and represent an 8×8 matrix, see figure 6.5. Data is sampled on the Din port every cycle and data is output every cycle on the Dout port. For the DCT, the input is a 9 bit signed integer in the range -256 to +255 and the output is a 12 bit signed integer in the range -2048 to +2047. For the IDCT, the input is a 12 bit signed integer in the range -2048 to +2047 and the output is a 9 bit signed integer in the range -256 to +255. The main computation is performed by two identical multiplication arrays, each of which performs an 8×8 matrix multiplication in 64 cycles. Therefore, the first sample of the block is output on the Dout port 128 cycles after the first sample of the block was sampled on the Din port. The intermediate 8×8 matrix result is rounded to 16 bits and stored in the transposition RAM between the two multiplication arrays. The transposition RAM also serves a function of transpos-

ing the data from column order into row order. This permits the two matrix computation elements to be identical although the first multiplication is the column computation and the second one is the row computations. The 2-dimensional DCT is then realized by two identical multiplications, i.e. $Y = (X \times C)^T \times C$, and the 2-dimensional IDCT is realized by two identical multiplications, i.e. $X = (Y \times C^T)^T \times C^T$, where X is the input matrix data, C the DCT coefficient matrix, C^T the IDCT coefficient matrix and $(\,)^T$ the transposition operation on a matrix. The DCT and the IDCT coefficients are 14 bits signed integers stored in 4 banks of fixed ROM.

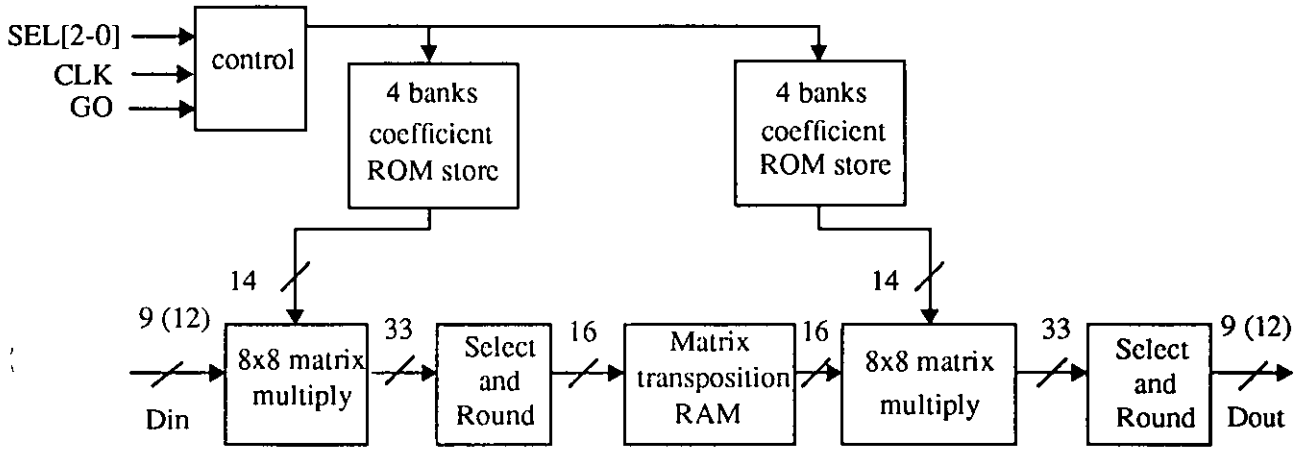


Figure 6.5 The diagram of the Inmos A121 chip

The Inmos A121 chip is a high speed processing chip, which is partly realized by pipeline processing technique. The device is fully pipelined with data sampled on the input port at the clock frequency, data processed in the chip, and the result output appearing 128 clock cycles later. As a result, on the input port, blocks of data can follow directly after one another so that the first data of a block is presented to the input exactly 64 cycles after the first data of the preceding block, meanwhile the preceding block is being processed and the further preceding block is output. Figure 6.6 gives an illustration of the pipeline processing. For example, while block C is sampled on the input port, block B is processed inside the chip and block A is output on the output port at the same time. In this way, the high throughout is achieved.

The main control signal of the Inmos A121 chip required to design is GO signal, see appendix B.

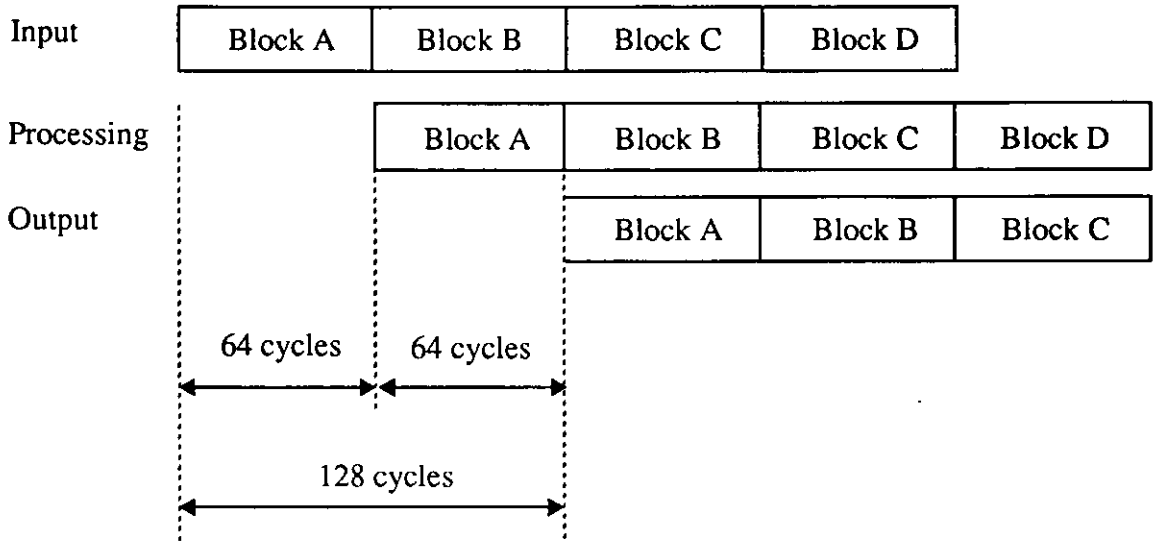


Figure 6.6 Data timing diagram

The GO signal is used to indicate the start of a block. When it is sampled high, the data on the Din port is the first sample of the block. Data is always assumed to be valid for the 64 cycles from the start of a major cycle, meanwhile the GO signal is ignored. Blocks of data may be processed at any time and any spacing between the major blocks, by toggling the GO signal as necessary.

6.2.3 The CY7C408A FIFO Chip

The CY7C408A chip [Cypr89] is a 64-word deep by 8-bit wide first-in first-out (FIFO) buffer memory. It is implemented by a dual port RAM cell, where the writing and reading operations are independent of each other so that the input and output operations are truly asynchronous. This allows the FIFO to be used as a buffer of two digital machines of widely different operation frequencies. Also, due to the dual port RAM architecture, data input and output is realized by increasing the write and read pointers instead of moving data. Since the time required to increment the read and write pointers is much less than the time that would be required for data to propagate through

the memory, high throughput rate is achieved. Its shift-in and shift-out rates range from 17MHz to 35 MHz. It is ideal for high speed communications and controllers.

The logic block diagram of the CY7C408A is shown in figure 6.7. It consists of a memory array of 64 words of 8 bits, a write pointer, a read pointer and the control logic necessary to generate the handshaking (SI/IR, SO/OR) signals as well as the Almost Full/Almost Empty(AFE), and the Half-Full (HF) flags. The memory accepts 8 parallel bits at its inputs pins (DI0-DI7) under the control of the Shift-In (SI) input when the Input-Ready (IR) control signal is High. The data is output in the same order as it was stored, on the DO0-DO7 output pins under the control of the shift-out (SO) input when the Output-Ready (OR) control signal is High. If the FIFO is full (IR LOW), pulses at the SI input are ignored; if the FIFO is empty (OR LOW), pulses at the SO input are ignored. The IR and OR signals are also used to connect the FIFO's in parallel to make a wider word, or in series to make a deeper buffer, or both.

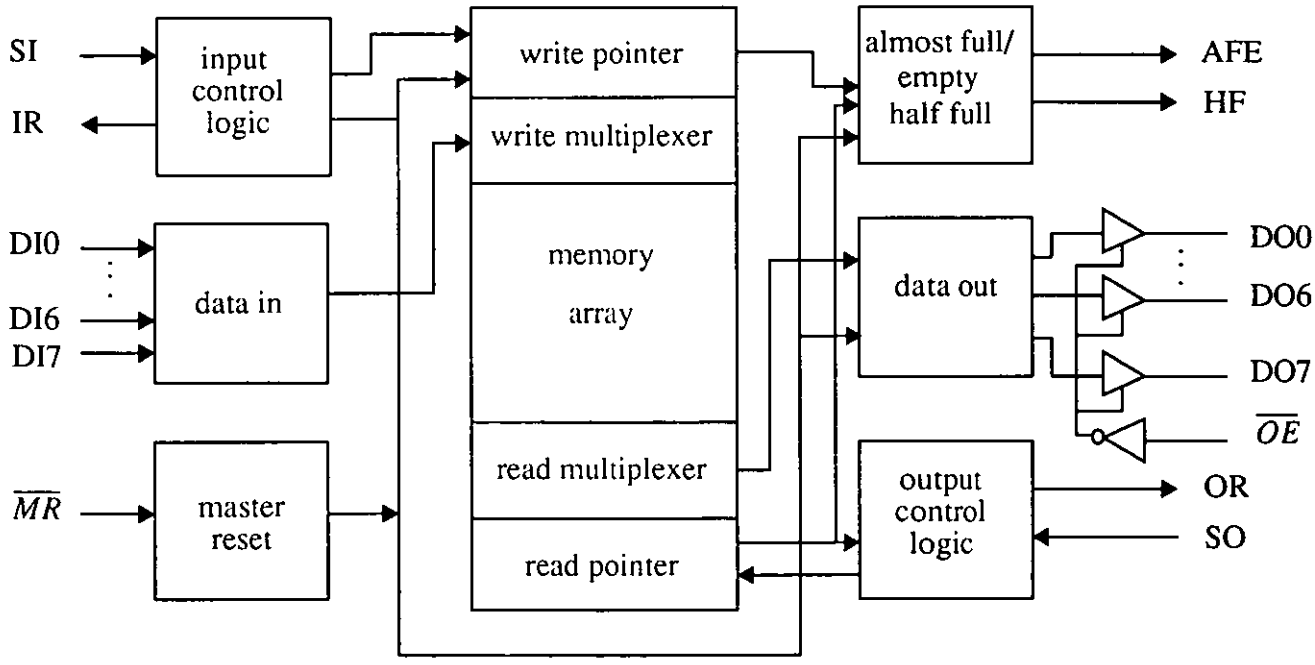


Figure 6.7 Logic block diagram

The main control signals which need to be designed are SI and SO. These two signals are used to

control the data in and out of the FIFO. A low to high transition on the SI pin will load the data on the input pins into the FIFO. Therefore, the low to high transition on the SI pin should happen after the input pins hold efficient data. A low to high transition on the SO pin will load the data in the FIFO out to the output pins. The low to high transition should not happen until the read device is ready to receive the data from the FIFO. The data in and out timing diagram can be seen in appendix C.

The CY7C408A is expandable in required word width and FIFO depth. Parallel expansion for wider words is implemented by logically ANDing the IR and OR outputs respectively of the individual FIFOs together, see figure 6.8. The AND operation insures that all of the FIFOs are either ready to accept more data or are ready to output data and thus it can compensate for variations in propagation delay times between devices. Serial expansion for deeper buffer memories is accomplished by connecting the data outputs of the FIFO closest to the data sources to the data inputs of the following FIFO, see figure 6.8. In addition to insure proper operation, the SO signal of the upstream FIFO must be connected to the IR output of the downstream FIFO and the SI signal of the downstream FIFO must be connected to the OR output of the upstream FIFO. In the serial expansion configuration, the IR and OR signals are used to pass data through the FIFOs.

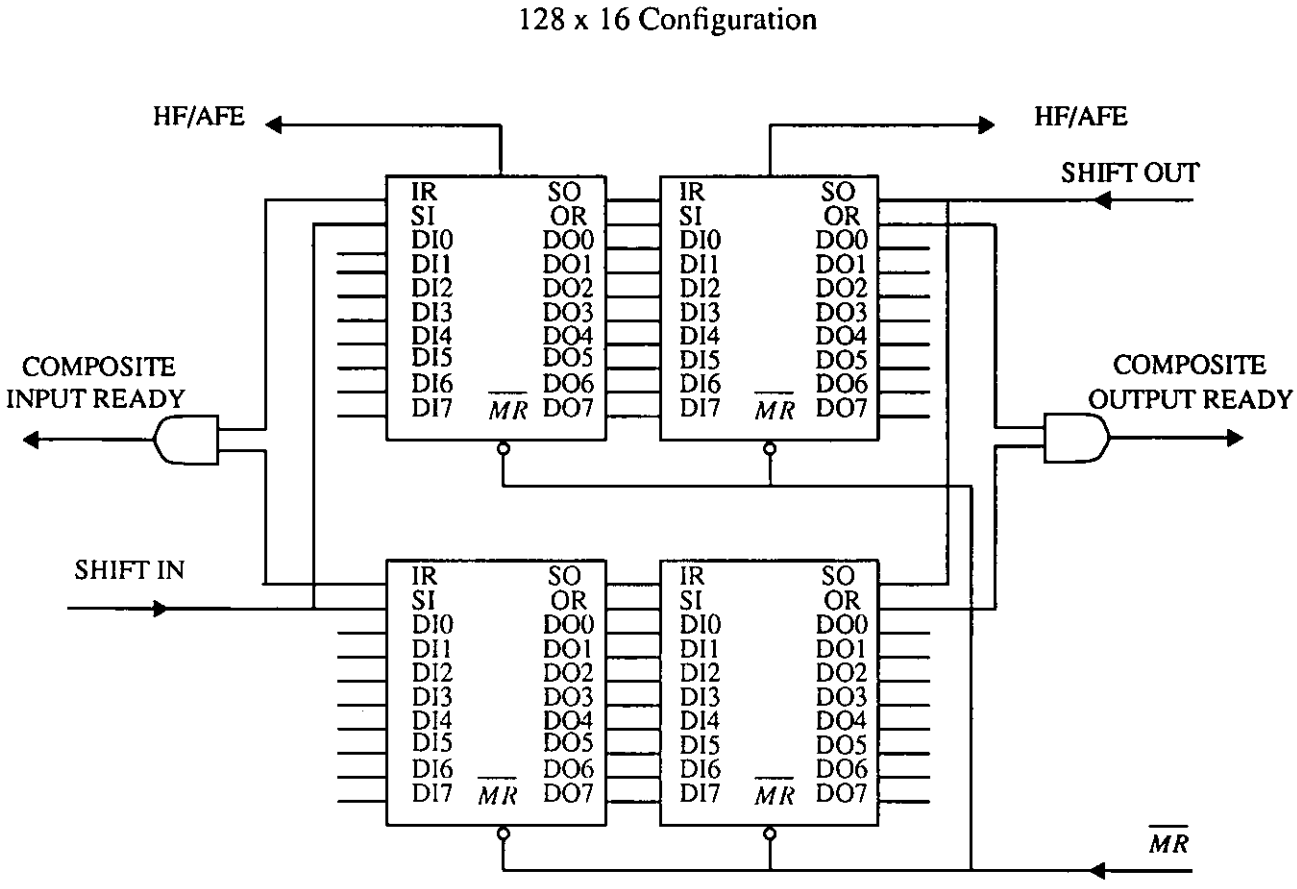


Figure 6.8 Depth and width expansion

6.2.4 The DCT Board Design

The DCT board is designed to perform the discrete cosine transform on image data with high precision and speed. It is a transputer-based board and can be used as a node in transputer networks communicating with other boards by transputer links. The board is mainly composed of two independent parts, namely the DCT part and the IDCT part, see figure 6.9. In the DCT part, it receives blocks of data, each of which has fixed size of 64 pixels and represents an 8×8 matrix, from an

image processing device through a transputer link and then applies the 2-d DCT to the blocks. After the DCT, it thresholds the DCT coefficients, sequences the matrixes in the order of zigzag, see §5.4, and then sends the processed blocks of data out by a link. While in the IDCT part, it receives sequences of 64 data items via a transputer link, de-zigzags them into an 8×8 matrix and then de-thresholds the matrix. The 2-d IDCT is then applied to the matrix. Thereafter, every element in the matrix is rounded to the range 0 to 255 and output to an image processing device by a link. The maximum throughput of the board can reach up to 20 Mpixels/sec, but the throughput here is confined by the link speed and is 256 Kpixels/sec.

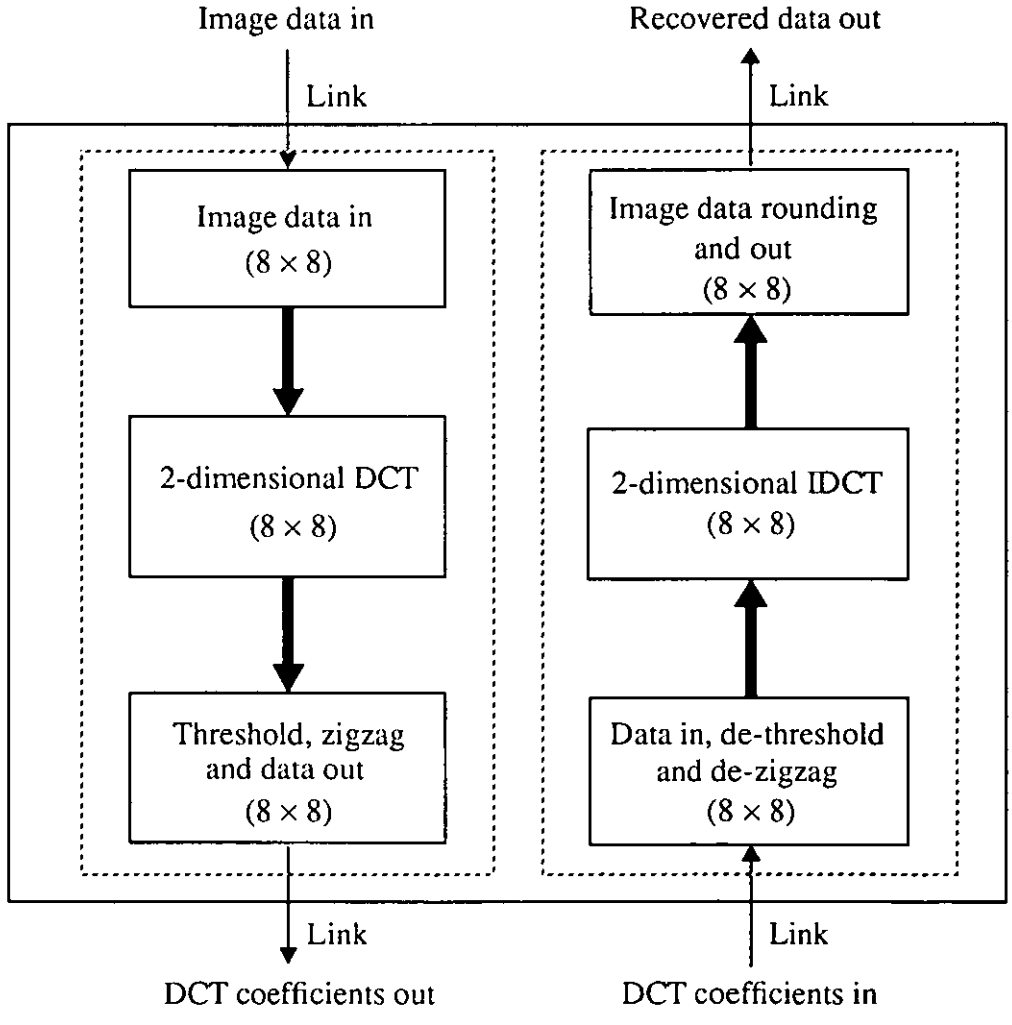


Figure 6.9 The function diagram of the DCT board

The Inmos A121 chip is chosen to realize the DCT owing to its high speed and precision. Two A121 chips are used here, one for the DCT and one for the IDCT.

The transputer is selected in the board mainly for two reasons. First, since the whole system works on the transputer-based system, it is feasible to use the transputer to provide transputer link interfaces for the board so that the message can be exchanged easily among the processing boards. Second, it has on-chip memory which can save a lot of complicated external memory interface circuits and simplify the board design. As the data for the DCT are 9 bit signed integers and the coefficients out of the DCT are 12 bit signed integers, the 16 bit T222 is chosen for economical considerations. Four T222's are used on the board, see figure 6.10, where T222 (1) controls blocks of data in through a link and blocks out them in the FIFO (1), T222 (2) blocks in the DCT coefficients from the FIFO (2), does the threshold and zigzag processing on the coefficients, and then control the data out through a link, transputer (3) controls blocks of data in through a link, does the de-zigzag and de-threshold processing to them, and blocks out them in the FIFO (3), and T222 (4) blocks in blocks of data from FIFO (4) and rounds the data in the range 0 to 255 and controls blocks of data out through a link. Each T222 provides one link as the board interface to communicate with other transputer-based systems.

The T222 and the A121 write/read data in different frequencies and they cannot be connected directly. The A121 reads/writes a block of 64 data each time, that is, when the GO signal is High, it takes in/outputs a sequence of 64 data spanning 64 clock cycles, where each data is sampled in one clock cycle. The data in and out have strict time requirement. The data input port is sampled 64 times on successive clock cycles, commencing when GO is sampled High. Data must be valid on the rising edge of CLK for each of the 64 cycles. The output port will be valid for periods spanning 64 clock cycles, and the data will be valid on the rising edge of the clock, exactly 128 cycles after the data was sampled on the input. On the other side, the T222 reads in/writes out one data each time in one external memory cycle. It can read/write a sequence of data in a burst of short time. But it is very hard to control the T222 reading/writing data to comply with the pace of the A121 read-

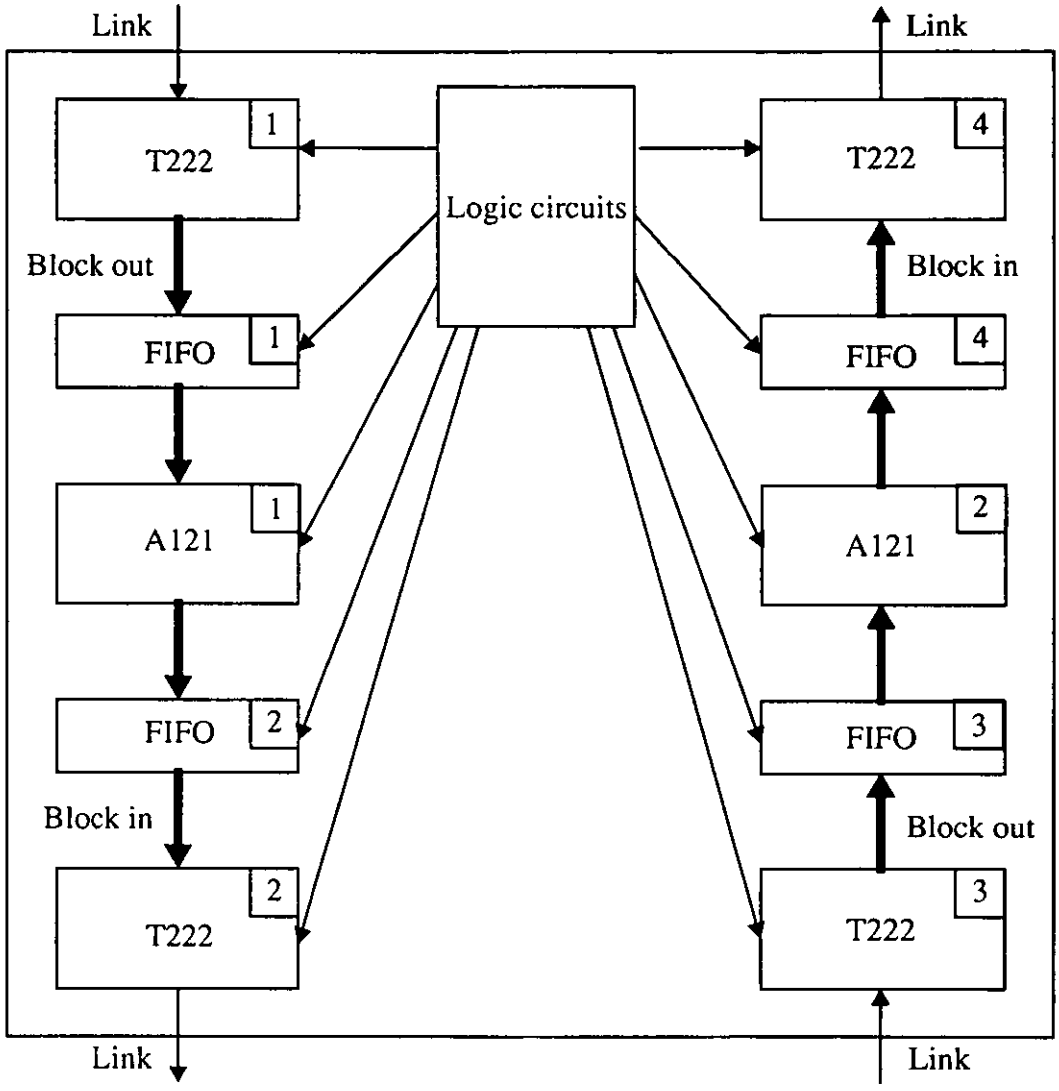


Figure 6.10 The DCT board diagram

ing/writing. To deal with the incompatibility of the operation frequencies of the T222 and the A121, the two port CY7C408A FIFO is used as a buffer between them where one port is used for the T222 and the other one for the A121. Since the T222 data bus is 16 bit wide and the A121 is 12 bit wide while the CY7C408A is 8 bit wide, FIFOs are depth expanded by two CY7C408A chips. Four FIFOs are used on the board, see figure 6.10.

The control signals required to design are as follows:

1. EVENT: When the T222 reads an EVENT signal, it writes/reads 64 data to/from the FIFO.
2. $\overline{MR}$: Before the T222 begins to write 64 data items in the FIFO, or after the T222 finishes reading 64 data items from the FIFO, a $\overline{MR}$ signal is used to reset the FIFO.
3. SI: 64 consecutive SI signals for the FIFO to input 64 data items from the T222/A121.
4. SO: 64 consecutive SO signals for the FIFO to output 64 data items to the A121/T222.
5. GO: When there is a GO signal except when the A121 is reading data, the A121 reads in 64 data items from the FIFO for the DCT/IDCT processing and after 128 cycles, the processed data are output and stored in the FIFO.

Since the DCT and the IDCT parts are almost symmetry, the DCT part is discussed in the remaining part of this section and the IDCT part can be concluded similar results. The timing diagram of the DCT part is shown in Figure 6.11. The clock cycles are divided by 128 and the first 64 cycles are for the transputers to write/read 64 data in/from FIFOs, and the second 64 cycles are for the A121 to read/write 64 data from/to FIFOs. Also in the second 64 cycles, the transputers receive/send data from/to the other image processing devices via a link and do the data processing, like thresholding, zigzagging etc. More detail, when T222 (1) reads an Event signal, it resets FIFO (1) and then blocks out 64 data in the FIFO (1). When T222 (2) reads the Event signal, it blocks in 64 data from the FIFO (2) and then resets the FIFO (2). At the $(2k+1) \times 64 + 1$ cycles, k=0, 1, 2,..., GO signal is High, and A121 (1) samples 64 pixels at its input port from FIFO (1) and output 64 data at its output port to FIFO (2). In every 128 cycles, the transputers read/write a block of 64 data in/from FIFOs in the first 64 cycles, while the A121 chip processes a block of 64 data in the second 64 cycles and the T222s also receive/read a block of 64 data from/to image processing devices via links in the second 64 cycles. The board works on in this way after it is started. The T222s and the A121s are pipelined to achieve high speed. For example, during the time from cycle 193 to cycle 256, A121 (1) input block2 from the FIFO (1) and at the same time outputs block1; during the time from cycle

257 to cycle 320, blocks is written in the FIFO (1), block2 is applied the DCT processing and block1 is read by T222(2) from FIFO(2).

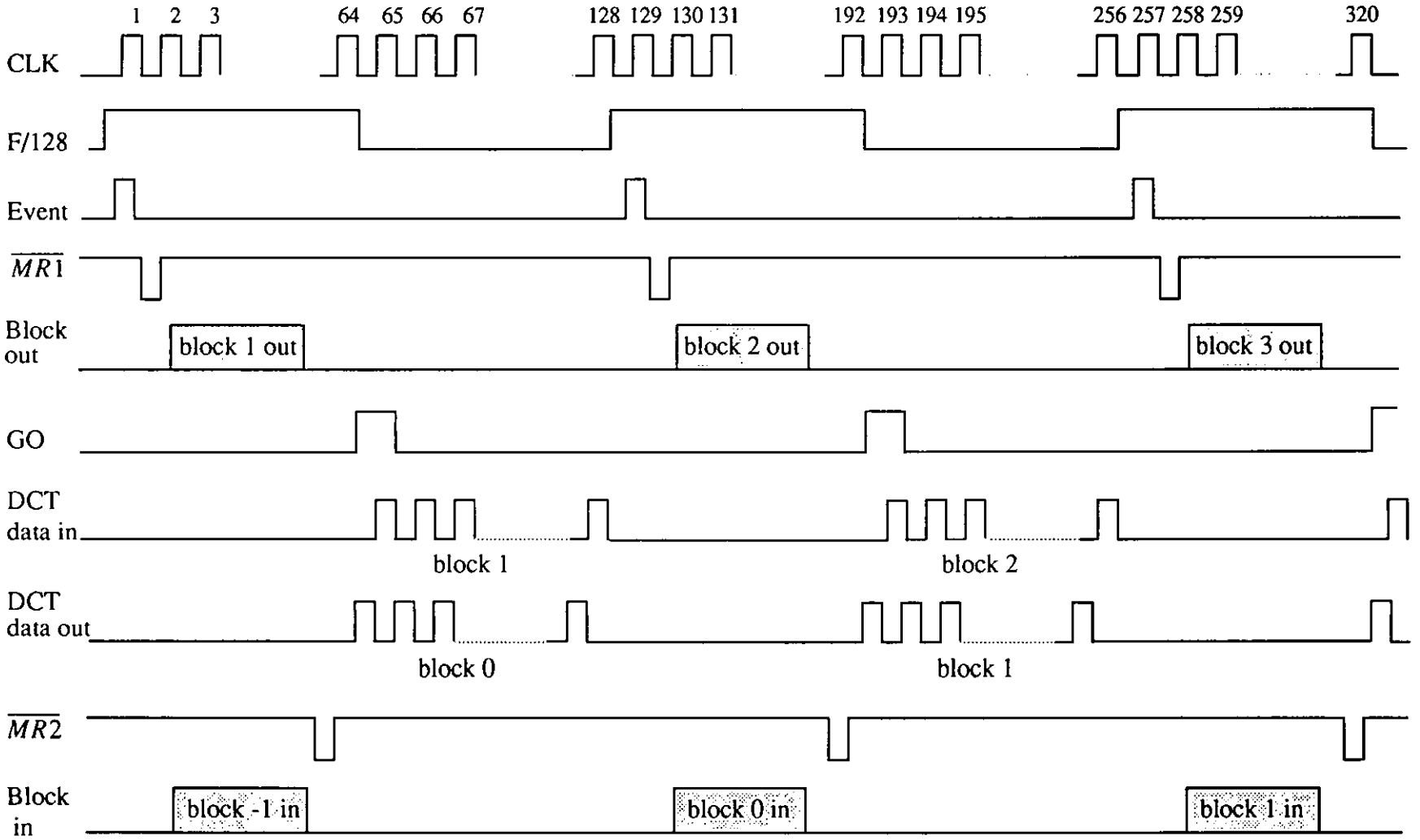


Figure 6.11 The timing diagram of the DCT part

6.3 The MicroEye TC Colour Image Capture and Framestore Board

MicroEye TC board is a high speed video capture card and framestore using transputer technology. The card can be installed in one 8-bit expansion slot inside a PC. The MicroEye TC board can digitize and grab live video signals from a camera, a recorder, an imaging system or other sources, and has a video output for replaying stored or processed images. It has a large memory capacity, a graphics overlay facility, and high speed data transfer and processing. It provides full 24 bit (16.7 million colours) frame grabbing and output. Images may be digitized at full 720×512 pixel resolution in 1/25 second (frame rate) or at 720×256 resolution in 1/50 second (field rate).

The structure of the MicroEye TC board is shown in figure 6.12. A T425 is used as a processing unit to control the board and conduct image processing. Under the control of the transputer, it digitizes RGB signals in one frame period or one field period, and holds the resulting data in a frame store video memory or be played through the on-board colour look-up table for output. Each pixel consists of 24 bits, 8 bits for each of the R, G, B colour components. A grabbed or transputer-processed image in the frame store can be played through the on-board 24-bit programmable colour look-up table for output as real or pseudo colour graphics to a video display. The memory on the MicroEye TC board is composed of three main regions, which are video memory, picture memory and program memory. The video memory is a dual-port memory which is used to hold the 720×512 pixel $\times$ 24 bit grabbed image data straight from the A/D converters. The picture memory is the 1024×512 pixel $\times$ 24 bit memory to which an image may be transferred to provide a second copy for reference and processing purposes. The frame store also contains 4 additional bits which can be used to generate overlay graphics under the control of the transputer. The video memory and picture memory together are referred to as image memory which has 3 Mbytes altogether. The transputer can access the frame store to process image data and the picture store can output from or load data into the frame store by a fast transfer routine under the control of the transputer. Thus data in the picture store may be generated or processed independently, without affecting the framestore. The 1 Mbytes program memory provides sufficient space for most applications.

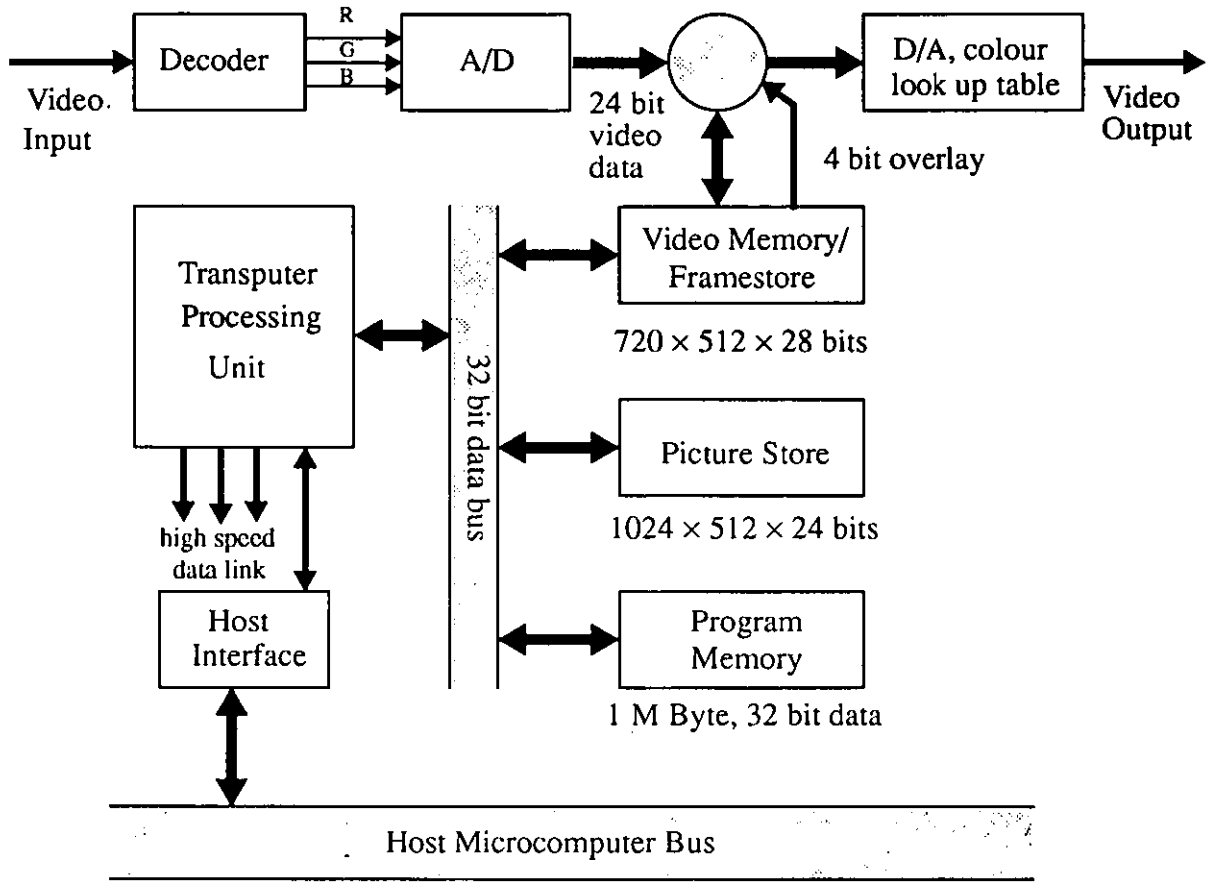


Figure 6.12 The MicroEye TC structure diagram

In the experimental system, the board is mainly responsible for image grabbing and displaying. It takes a video input, digitizes it in one frame period and stores it as a 720 × 512 pixel, 24 bit data block in its video memory under the control of the B004 board. The grabbed image is then sent to the B004 board via a transputer link or replayed through the on-board palette to provide a full colour video output. The images processed by the B004 are sent back to the MicroEye TC board to be displayed by a link. The 1 MByte program memory is sufficient for the Occam control programs, which control the grabbing, displaying images and data transferring from or to the B004 board or the DCT board. One of its transputer links is connected to the B004 board and two links to the DCT board and the other one is left for further expansion.

6.4 Summary

In the experimental system, most of the processing, such as colour codebook generation, colour codebook ordering, colour image vector quantization etc., are carried out on the B004 board. The experimental image can be obtained either from a DOS file or from a video camera under the control of the MicroEye TC board. When the B004 board need an image from the video camera, it sends a control signal through a transputer link to the MicroEye TC board and the MicroEye TC board captures an image and sends the image to the B004 board through a link. When the image need to be DCT processed, the B004 board sends the image data to the DCT board and after 128×2 cycles, while it is sending the image block data to the DCT board, it receives the recovered image block data from the DCT board at the same time. The DCT processing on a 512×512 image can be carried out by the DCT board with high speed, about 1 second, and also the DCT processing precision is high. When the processed image, or the final recovered image or some parts of the image need to be displayed, the B004 board sends a control signal and image data to the MicroEye TC board, then the MicroEye TC board displays the image on the monitor. The boards in the experimental system work in concurrent and the data communication between the boards is realized by transputer links.

Chapter 7

Experimental Results and Discussions

The previous chapters have discussed how it may be possible to transform a colour image into a meaningful index image with a well ordered colour codebook. It can be seen from previous discussions that the crucial step to achieve this transform is colour codebook ordering. A number of techniques for colour codebook ordering have been proposed and are discussed in chapter 3. In this chapter, experiments are described which test the efficiency of those techniques for colour codebook ordering. The tests are: computing the ordering function on the ordered colour codebook (§ 7.3.1), applying edge detection to the index image (§ 7.3.2) and compressing the index image using the DCT processing (§ 7.3.3).

7.1 Experimental Images

Four colour images are used as the test images to which the techniques discussed in the previous chapters are applied. The four colour images referred to as "girl1", "girl2", "pepper" and "plane" respectively are shown in figure 7.1. They are taken from a standard image database. The experimental colour images are all of resolution 512×512 pixels. Each pixel is represented by 24 bits where 8 bits are used for each colour component, e.g. Red, Green and Blue. These images are stored on the hard disk memory for processing. The details of the four colour images are described as follows:

(a) image "girl1"

(b) image "girl2"

(c) image "plane"

(d) image "pepper"

Figure 7.1 Experimental images

- **Image 1: "girl1"**

Image "girl1" shows a girl's head, see figure 7.1(a). In this image, the head covers more than half the area of the image. There are a few edge details, mainly in the area of hair, but a lot of smoothly varying regions, especially, in the face. It is not a colourful image and the primary colours used in the image are yellow, pink, red, brown, and green. These colours are quite close, especially light yellow, light brown, light pink and light green. The skin colour is actually a mixture of these colours. The histograms of the Red, Green and Blue components of the image are shown in figure 7.2, which illustrates the number of occurrences of each component likely levels. It can be noted that their level ranges are rather narrow, from about 90 to 255. This means that the colour dynamic range in the image is small and the image is not colourful. The Blue colour component has relatively small luminance value, and the Red and Green components have big luminance values. This means that the Red and Green components, especially the Red component, occupy more energy in the image, and they play more important roles in the colour appearance in the image.

- **Image 2: "girl2"**

Image "girl2" is about a girl's head and shoulder, see figure 7.1(b). Compared with image "girl1", it has more edge details and rather complex background. It also has a lot of smoothly varying regions in the areas of the face, the shoulder and the background. The primary colours used in the image are orange, reddish colours, yellow, purple, the colours between orange and red, and the colours between purple and red. The colours used are very similar. The histograms of the Red, Green and Blue components are shown in figure 7.3. Their levels cover a wider range than image "girl1". It can be seen that the Red colour component has big luminance levels and it plays the dominant role in the colour appearance of the image.

- **Image3: "plane"**

Image "plane" is a typical natural scene image as shown in figure 7.1(c). It has a few abrupt edge

details and the details are mainly in the area of mountains and the figures on the plane. It has large areas of snow and clouds which have no abrupt edges. The image does not contain many saturated colours and the main colours used in the image are blue and red. The histograms of the Red, Green and Blue components are shown in figure 7.4. It can be seen that the histograms of the Red, Green and Blue components are quite similar and their peaks are around levels 192 to 224. This means that there is a lot of correlation among the Red, Green and Blue components and there are a lot of achromatic colours in the image. In this image, the Blue components occupy the most energy and the basic colour of the image is blue.

- **Image 4: "pepper"**

Image "pepper" is a still object image, see figure 7.1(d). The image is crowded with different size and colour peppers, big, small, red, green etc. There are some smoothly varying regions, for example, in the area of the big green peppers. It is a quite colourful image compared with the other three images. The main colours used in the image are bright red, green, orange and yellow. The histograms of the Red, Green and Blue components are shown in figure 7.5. The histograms of the three colour components are quite different and the three colour components have a quite wide dynamical range and their peaks are in different places. This means that the colour image is quite colourful. Most of the energy is taken by the Red and Green components.

The four experimental colour images are of different types in colour distributions and image patterns. Image "girl1" has a lot of smoothly varying regions, especially in the face, and few edge details. Image "girl2" has more details and also has a lot of smoothly varying areas. These two images do not have many saturated colours and the colours used are quite close. These two images are about human's head or head and shoulder, which are the main contents in many applications, such as video-conference, video-phone etc. Image "pepper" is more colourful and has two strong contrasting colours, red and green, as well as some colours in the middle between red and green, such as yellow, brown etc. It has few details and some smoothly varying areas. Image "plane" is a typical natural scene image, which uses a few colours and a lot of achromatic colours. It has few smoothly

varying areas and edge details. Image "plane" has lower image quality requirement than the other three images. For example, the distortion in the sky is not so easy to perceive. Therefore, it is easier to process.

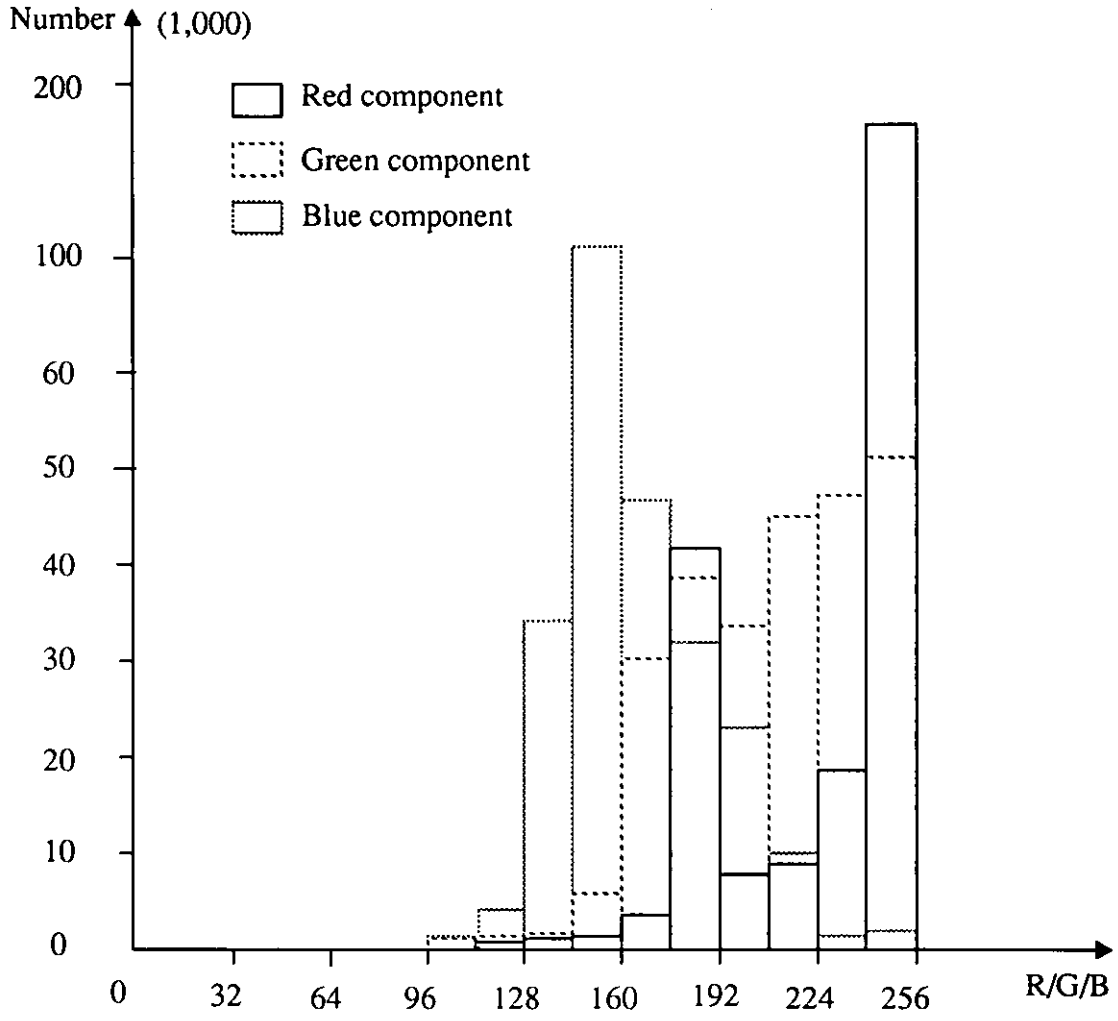


Figure 7.2 The histograms of the Red, Green and Blue components in image "girl1"

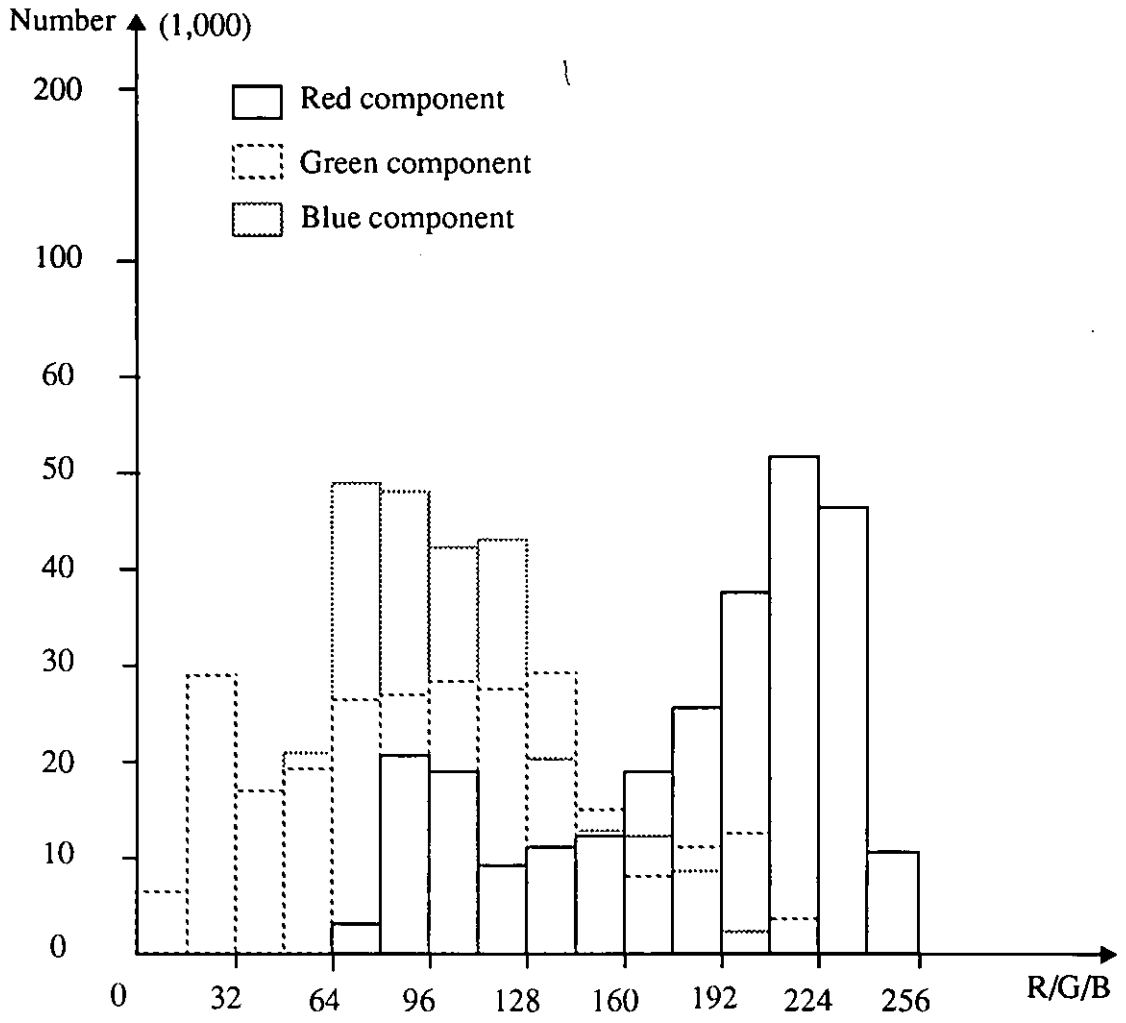


Figure 7.3 The histograms of the Red, Green and Blue components in image "girl2"

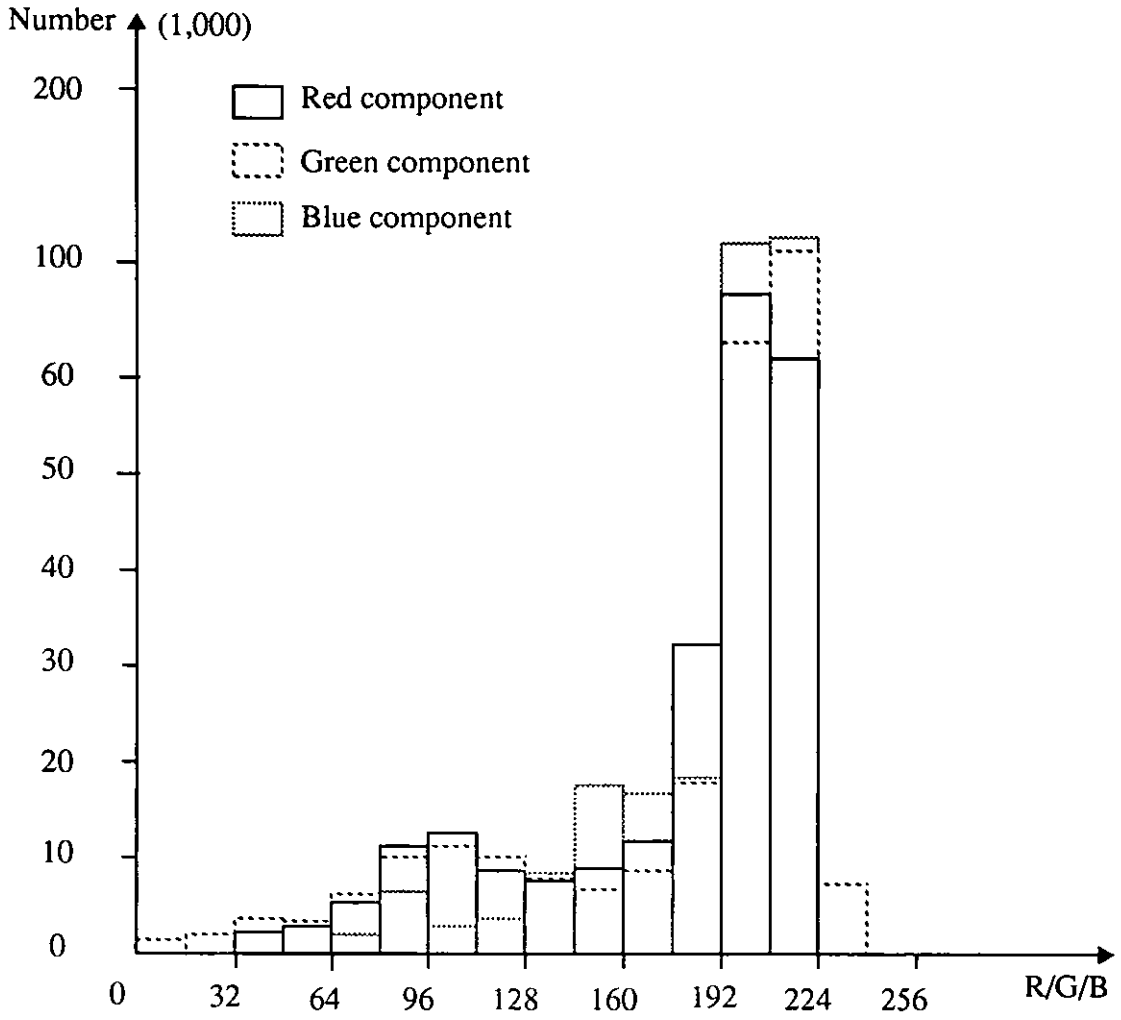


Figure 7.4 The histograms of the Red, Green and Blue components in image "plane"

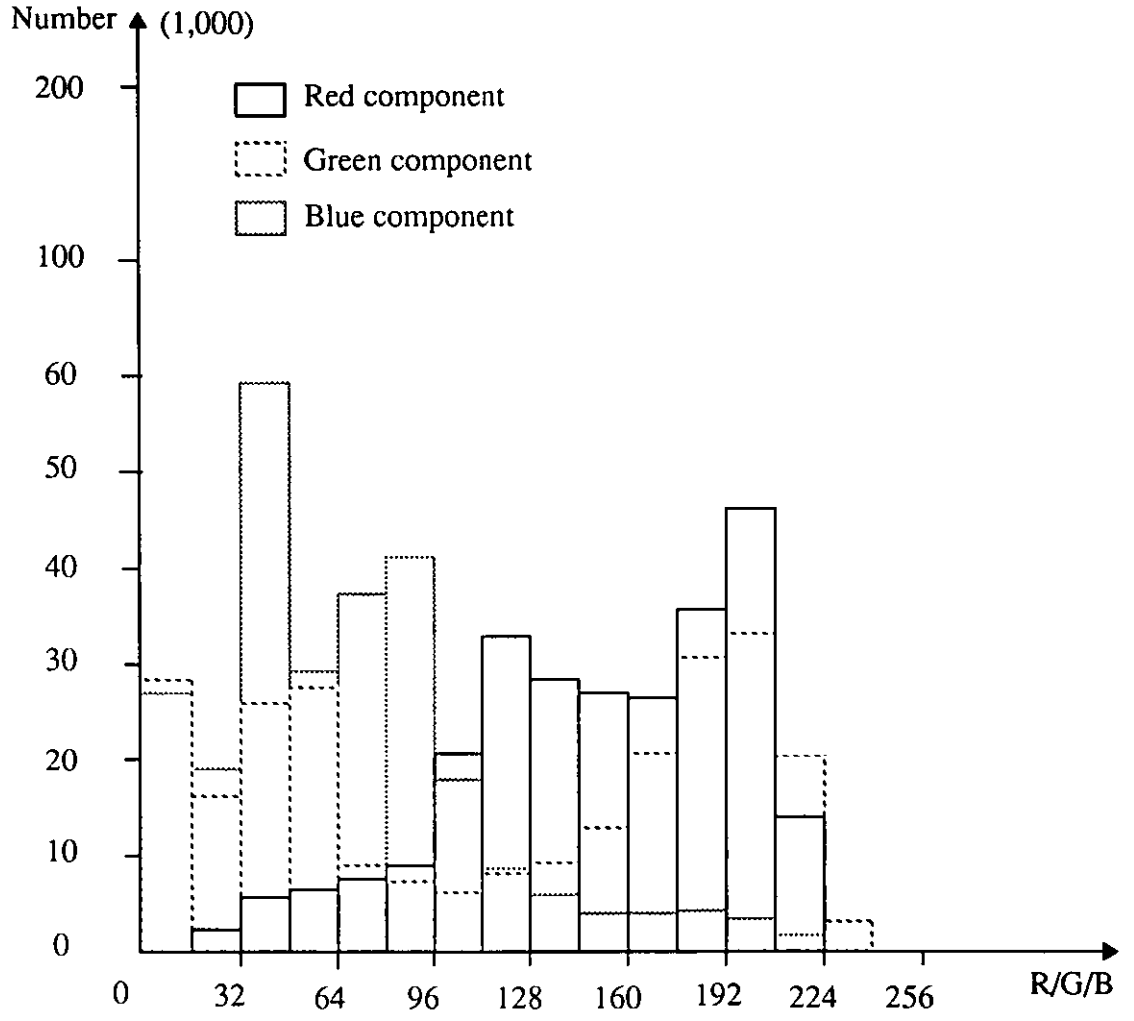


Figure 7.5 The histograms of the Red, Green and Blue components in image "pepper"

7.2 Colour Codebook

Besides the ordering algorithms, the quality of colour codebook ordering is also dependent on the features of the codebook, such as the codebook size, the codewords distribution in colour space etc. In this section, the colour codebooks of the four experimental images are discussed. The colour codebooks discussed in the following sections are all local codebooks, i.e., the training vectors are generated from the image to be coded as described in chapter 3.

7.2.1 Colour Codebook Size

The codebook size and the quality of the reconstructed colour image are normally conflicting. In colour codebook ordering, it is easier to order a small size codebook than a large one. The smaller the size is, the easier the ordering is. It is then desirable that the colour codebook which can generate the good quality of the reconstructed colour image has as small a size as possible. On the other hand, a codebook of small size will introduce more distortion in the reconstructed image. Therefore, it is required to trade off between the codebook size and the quality of the reconstructed colour image. Normally, 256 codewords are sufficient for most colour images and the reconstructed colour image is almost visually identical to the original full colour image. For some colour images, less codewords, typically 100-150, are able to vector quantize the colour images and can generate quite good quality reconstructed colour images. The optimum size of the colour codebook for a colour image depends on the colour image to be coded and the codebook generation algorithms. The algorithms used in the following experiments are the PNN algorithm and the LGB algorithm, where the PNN algorithm is used to generate an initial codebook and the LBG algorithm is used to refine the initial codebook. For codebooks of different sizes, the Signal to Noise Ratio (SNR) of the reconstructed colour images is computed. The SNR is computed as follows:

$$MSE = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left((r_{ij}-r'_{ij})^2+(g_{ij}-g'_{ij})^2+(b_{ij}-b'_{ij})^2\right) \qquad (7.1)$$

$$SNR = \left(\frac{\frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (r_{ij}^2 + g_{ij}^2 + b_{ij}^2)}{(MSE)^{1/2}} \right)^{1/2} \quad (dB) \qquad (7.2)$$

where the r_{ij}, g_{ij}, b_{ij} are the original pixel data, r'_{ij}, g'_{ij}, and b'_{ij} are the reconstructed pixel data, and N is 512 here.

The relationship between the codebook size and the SNR of the reconstructed colour image for the four test images is plotted in figure 7.6. It can be observed that the plots of image "plane" and "girl1" are rather flat while the plots of images "girl2" and "pepper" are steeper. This means that the quality of the reconstructed colour images "girl2" and "pepper" is more sensitive to the codebook size than that of images "girl1" and "plane". The reason for this is that since more colours are used in images "girl2" and "pepper", more codewords are required to represent the original full colour image. It can also be noticed that, for the four images, when their codebook sizes are smaller than certain numbers, the plots of the four colour images in those ranges are all very steep. In this case, the reconstructed colour image quality is very sensitive to the codebook size. For example, for image "plane", when the codeword number is smaller than 95, the plot is steep. This indicates that for any image, a certain number of codewords are required and when less codewords are used, more distortion will be caused in the reconstructed colour image. Combining the consideration of the visual quality of the reconstructed colour images, the minimum codeword numbers which can generate good visual quality reconstructed colour images are 80, 80, 90 and 80 for images "plane", "girl1", "girl2" and "pepper" respectively, see figures 7.16(a), 7.17(a), 7.18(a) and 7.19(a). The codebook size used in the later discussions for images "plane", "girl1", "girl2" and "pepper" are chosen as 80, 80, 90 and 80 respectively. It can also be noted that when the image is more colourful, more distortion will be introduced in the reconstructed image, namely lower SNR levels. For example, image "pepper" is the most colourful image among the four test images and its average SNR is the smallest, while image "plane" is not very colourful, its average SNR is the biggest.

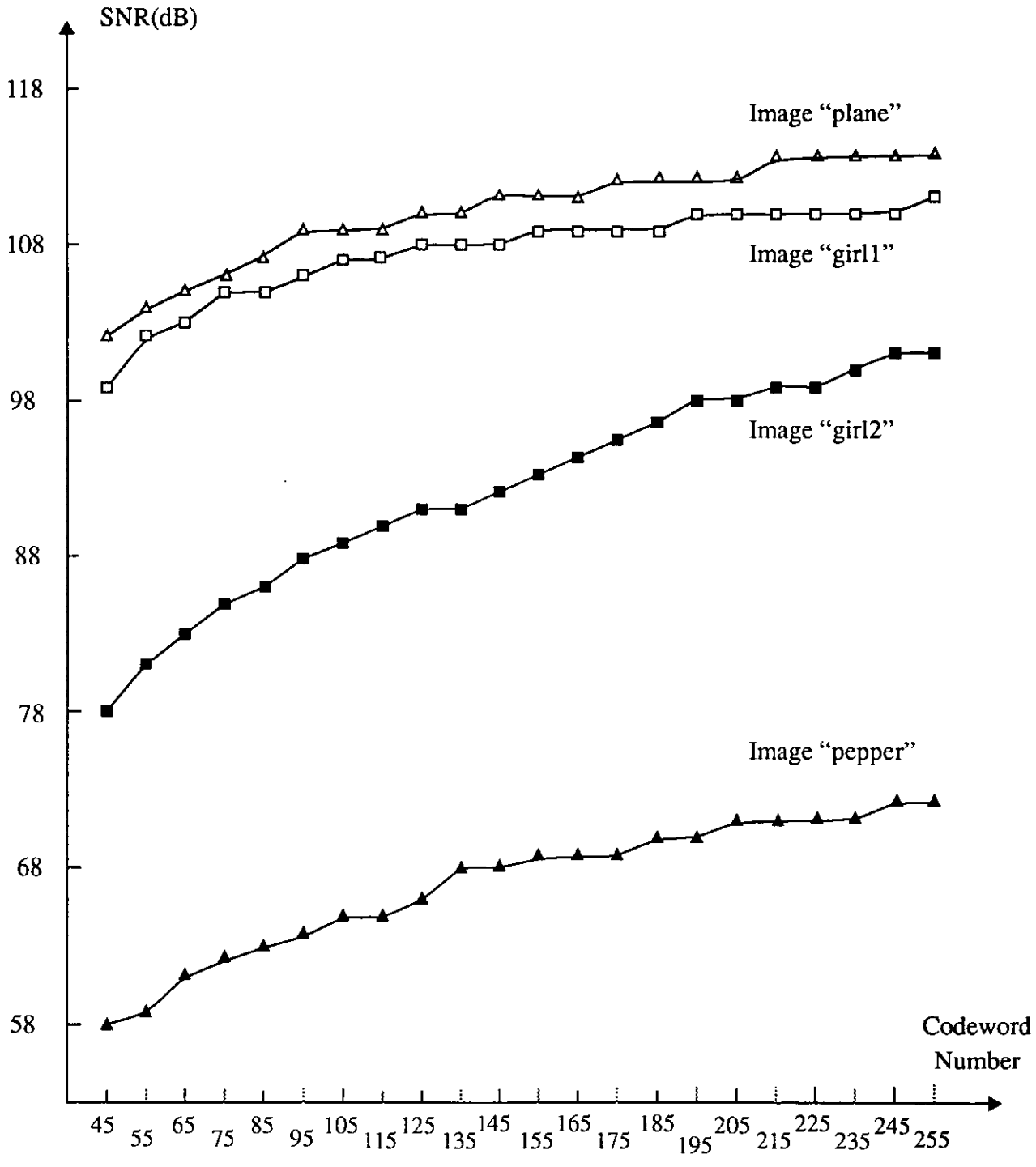
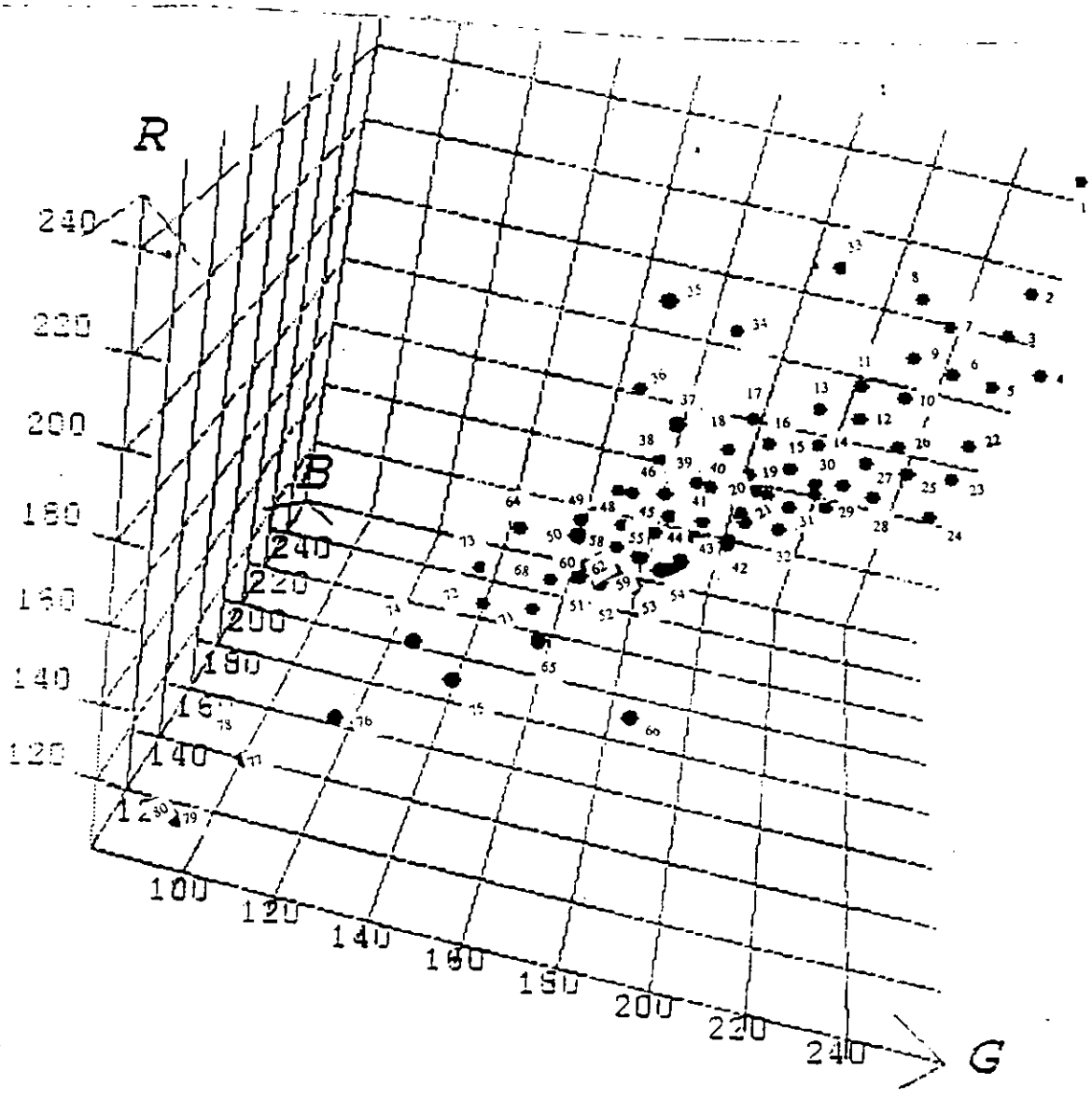


Figure 7.6 The SNR vs. the codebook size for four images

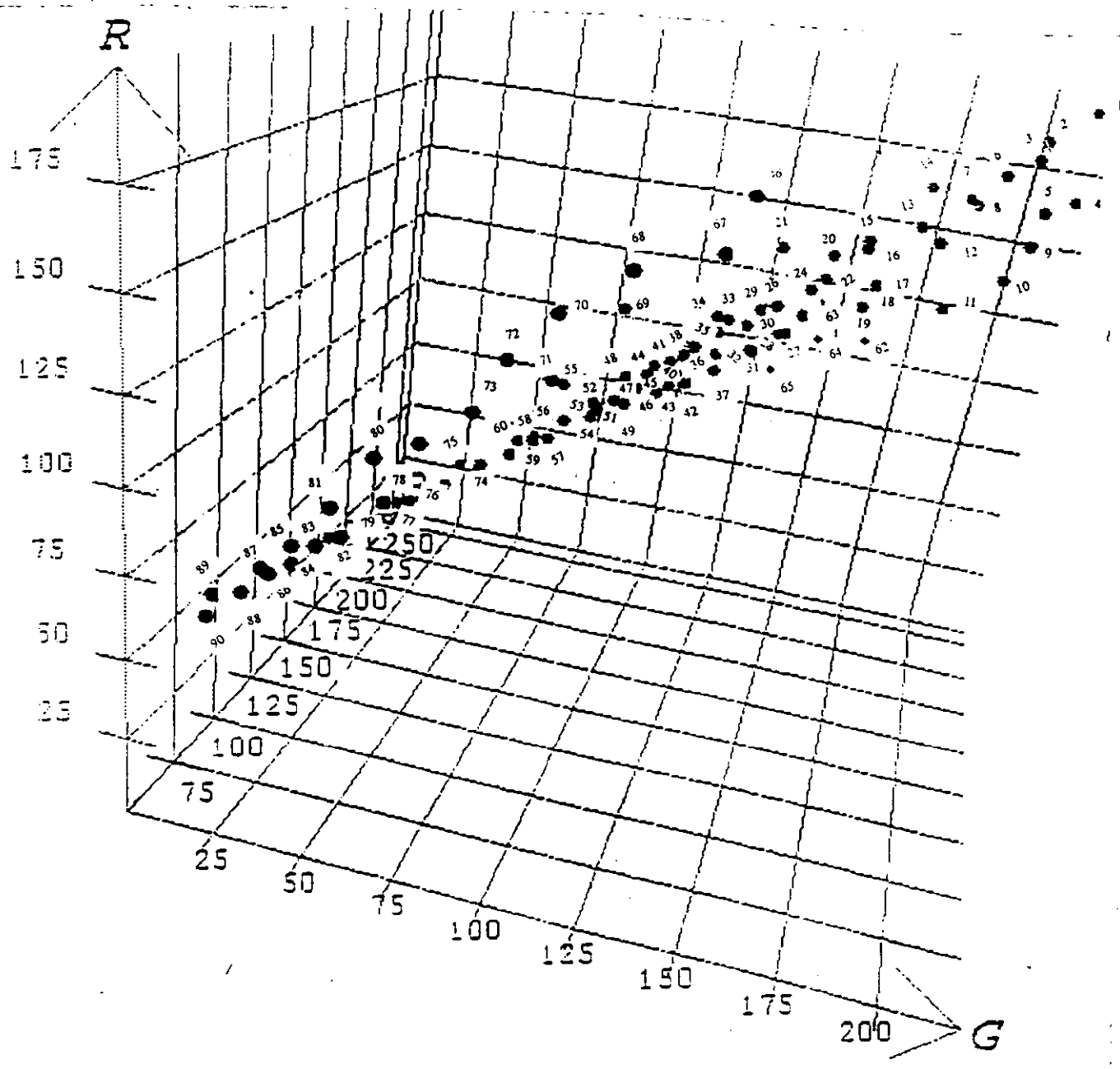
7.2.2 Colour Codebook Distribution in the RBG Colour Space

Colour codebook ordering also depends on the distribution of the codewords in colour space. If the codewords are uniformly distributed in colour space, it is impossible to order the codewords to satisfy the two ordering conditions. On the other hand, if the codewords are distributed on a line, the ordering is very easy and can completely satisfy the two ordering conditions. For most colour images, the codewords normally cover a quite small range of colour space and are not distributed over the whole colour space. Furthermore, the codewords are not distributed uniformly nor on a line but in the form of clusters. This can be seen in figures 7.7, 7.8, 7.9, and 7.10, which are the codebook distributions in the RGB colour space for the test images "girl1", "girl2", "plane" and "pepper" respectively.



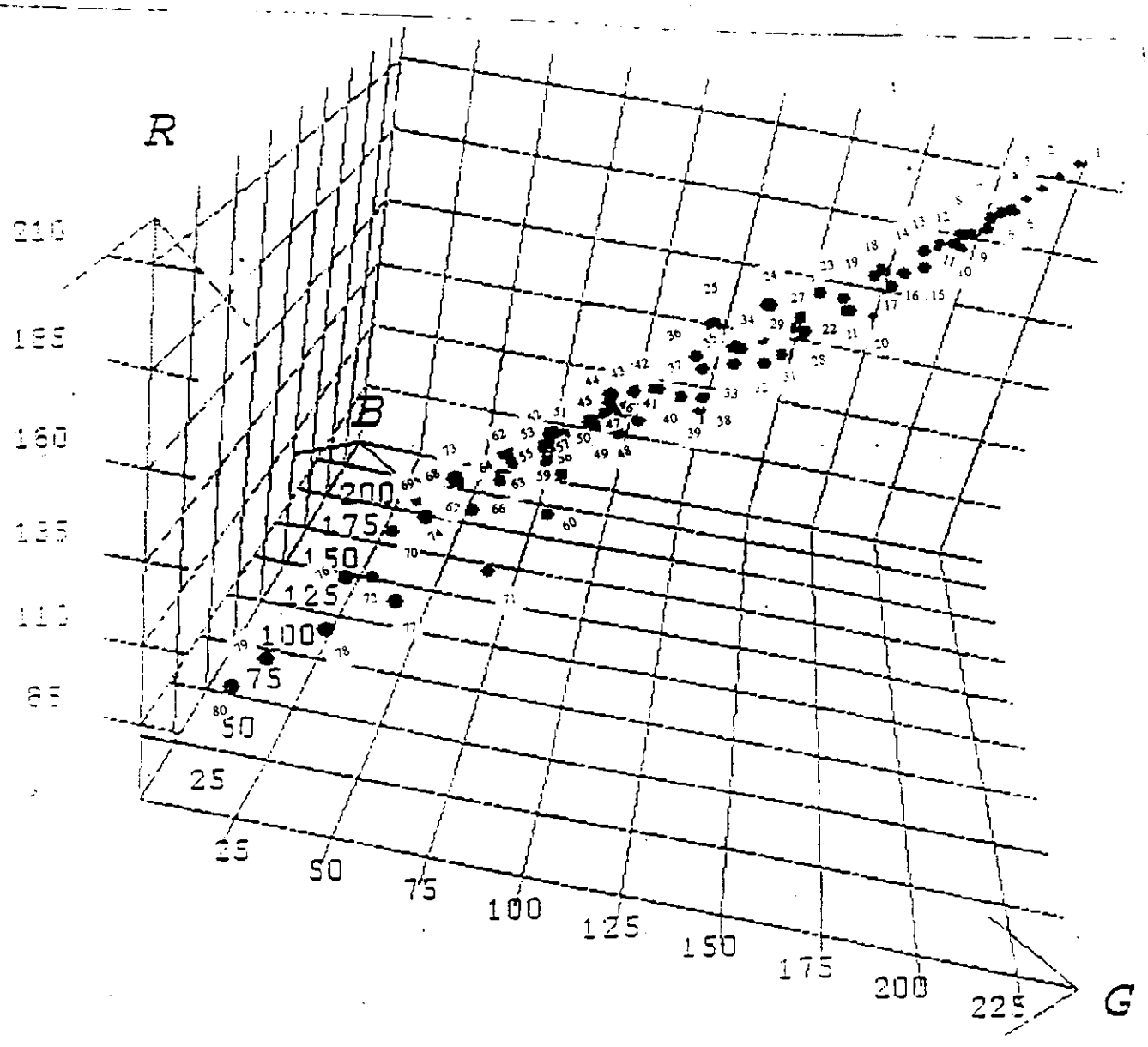
Note: the number beside a point is the index of the codeword in the ordered codebook

Figure 7.7 The codebook distribution in the RGB colour space for image "girl1"



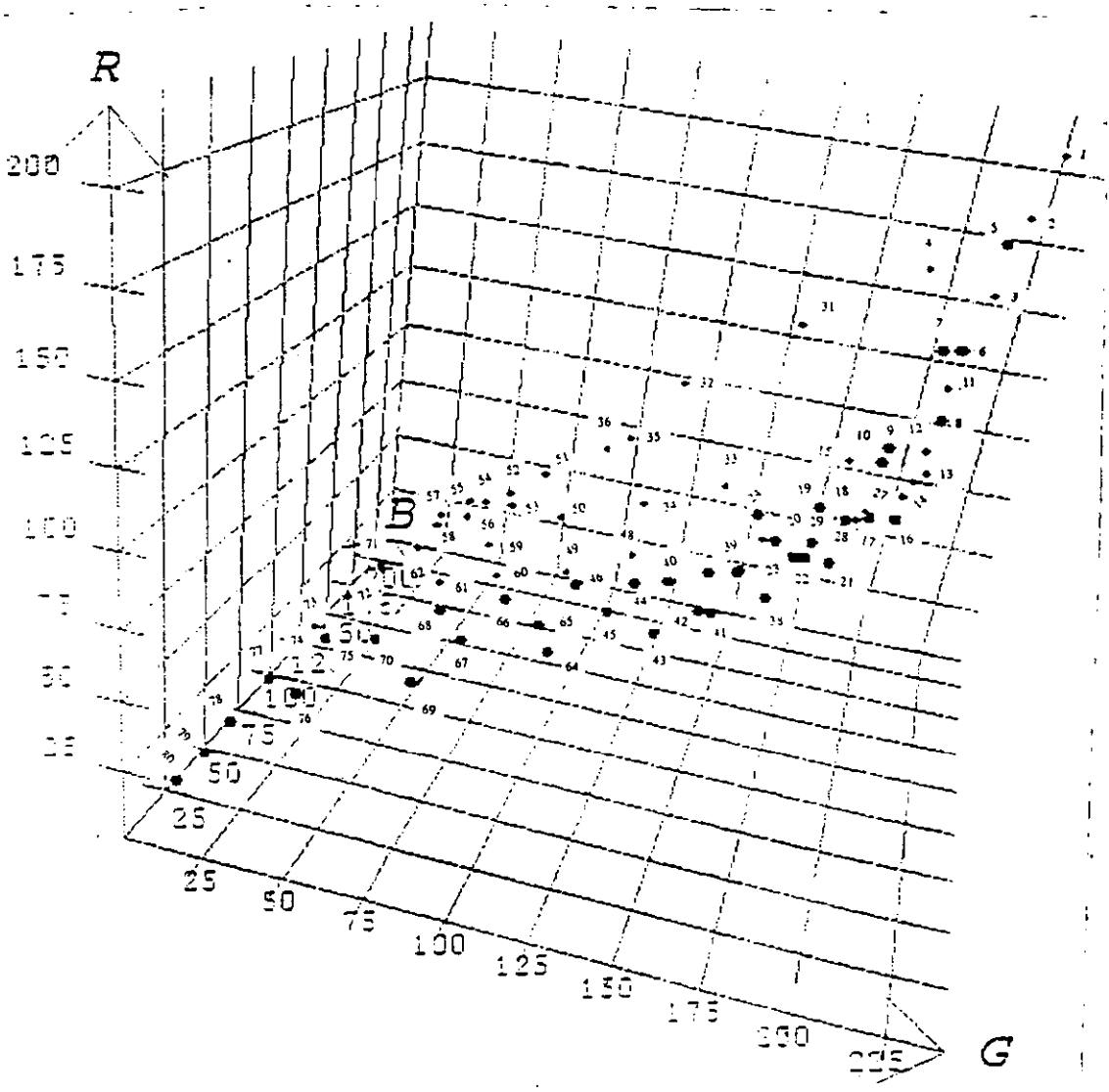
Note: the number beside a point is the index of the codeword in the ordered codebook

Figure 7.8 The codebook distribution in the RGB colour space for image "girl2"



Note: the number beside a point is the index of the codeword in the ordered codebook

Figure 7.9 The codebook distribution in the RGB colour space for image "plane"



Note: the number beside a point is the index of the codeword in the ordered codebook

Figure 7.10 The codebook distribution in the RGB colour space for image "pepper"

7.3 Comparison of Several Colour Codebook Ordering Strategies

A colour codebook can be ordered by the ordering techniques, which have been presented in chapter 3, to satisfy the two ordering conditions (see § 3.1). Four methods for colour codebook ordering are compared in this section. They are: 1) arranging the codewords in the decreasing order of the luminance value Y, which is referred to as the Y ordering method; 2) the centroid method (see § 3.2.1); 3) the PNN-based method (see § 3.2.2); and 4) the PNN-based method followed by the refinement of the ordering function (see § 3.3), which is referred to as the PNN-based/ordering function method. The randomly arranged codebook is used as a comparison reference. Finally the role of the strategy of the artificial codeword insertion (see § 3.4) played in colour codebook ordering is tested.

The efficiency of several colour codebook ordering techniques is compared by testing their corresponding ordered codebooks in three ways. First, they can be evaluated by computing the ordering function on the ordered codebooks. This is because the ordering function can be used to indicate how well the arrangement of the codewords satisfies the two ordering conditions (§ 3.1) and the smaller the ordering function value is, the better the codeword arrangement conforms with the two ordering conditions. Second, the ordered codebooks are tested by applying edge detection processing to the index images. This is to test whether the codewords which are far away in colour space are far away in the codebook so that the edges existing in the original full colour image can be retained in the index image. Finally, the ordered codebooks can be tested by applying image compression techniques, especially information lossy compression techniques such as the DCT, to the index images and then examining the reconstructed colour image quality with a certain compression ratio. This test can indicate how well the two ordering conditions are met by the ordered codebook. Information lossy compression techniques introduce some distortion in the reconstructed index image and the distortion eventually degrades the reconstructed colour image. However, if the codewords which are close in the codebook represent close colours, the degradation caused by the distortion in the recovered index image can be alleviated and the quality of the reconstructed colour image is good at a certain compression ratio. On the other hand, since the compression is achieved

by reducing the data redundancy in the image, high compression ratio can be obtained if much of the data correlation is retained in the index image. To retain the data correlation in the index image, it is desirable the close codewords which are close in colour space should be close in the codebook. Therefore, applying the image compression techniques to the index image and checking the reconstructed colour image quality with a certain compression ratio is an effective way to test whether the ordering techniques can order the codebook to satisfy the two ordering conditions.

7.3.1 Test By the Ordering Function

For the four experimental images, the ordering function is computed on five codebooks ordered in different ways. They are: randomly arranged codebook; the codebook ordered by the Y ordering method; the codebook ordered by the centroid method; the codebook ordered by the PNN-based method; and the codebook ordered by the PNN-based/ordering function method. The computed results are listed in table 7.1.

Ordering function \ Image / Codebook	Girl1 (codewords:80)	Girl2 (codewords:90)	Plane (codewords:80)	Pepper (codewords:80)
random	16147	23615	11776	16429
Y ordering	3304	3122	2368	2716
centroid method	3171	2807	1904	1984
PNN-based	2267	1942	1422	2162
PNN-based/ ordering function	1359	1374	1048	994

Table 7.1 Comparison of the codebooks by the ordering function

It can be seen that, for the four images, the randomly arranged codebooks have the biggest function values, while the codebooks ordered by the PNN-based/ordering function method have the smallest values. This means that the PNN-based/ordering function method can generate the best ordered codebook in the sense of satisfying the two ordering conditions by the arrangement of the codewords. Clearly, the codebook without any ordering does not satisfy the two ordering conditions. For the four images, the centroid method and the PNN-based method can achieve better ordering than the Y ordering method. For images "girl1", "girl2" and "plane", the PNN-based method yields better ordering than the centroid method, while for image "pepper", the centroid method is slightly better than the PNN-based method. To sum up, the PNN-based/ordering function method is the best in regard to producing the smallest function value, followed by the PNN-based method, the centroid method, and the Y ordering method, which is the least satisfactory.

It can also be noted from the table that the function values of the ordered codebooks for images "plane" and "pepper" are normally smaller than those of images "girl1" and "girl2". This indicates that the images which have a lot of saturated colours, like image "pepper", or have a lot of achromatic colours, like image "plane", have smaller ordering function values than the images which use a lot of close colours, like images "girl1" and "girl2". This is because, in the codebooks for images "girl1" and "girl2", there are a lot of junction codewords, see §3.1, and therefore it is much harder to order the codebooks.

7.3.2 Edge Detection on the Index Image

"Edge" in an image is defined as the boundary between two regions which have distinct features. The feature can be brightness for a monochrome image and colour for a colour image. Many techniques have been developed for edge detection [Davi75] [Robi77]. Here, a simple method is used and the processing steps are described as follows,

1. Compute the gradient at each pixel f(i, j) in the index image [Gonz87] [Youn86]:

Consider the sub-image shown in figure 7.11

f(i-1, j-1)	f(i-1, j)	f(i-1, j+1)
f(i, j-1)	f(i, j)	f(i, j+1)
f(i+1, j-1)	f(i+1, j)	f(i+1, j+1)

(a)

-1	-2	-1
0	0	0
1	2	1

(b)

-1	0	1
-2	0	2
-1	0	1

(c)

Figure 7.11 (a) 3×3 sub-image (b) Mask for G_i computation at center point f(i, j) (c) Mask for G_j computation at center point f(i, j). These two masks are referred to as the Sobel operator

Two partial derivatives for pixel f(i, j) are computed:

$$G_i = (f_{i+1j-1} + 2f_{i+1j} + f_{i+1j+1}) - (f_{i-1j-1} + 2f_{i-1j} + f_{i-1j+1}) \qquad (7.3)$$

$$G_j = (f_{i-1j+1} + 2f_{ij+1} + f_{i+1j+1}) - (f_{i-1j-1} + 2f_{ij-1} + f_{i+1j-1}) \qquad (7.4)$$

The gradient for pixel f(i, j) is then,

$$G(f(i,j)) = \sqrt{G_i^2 + G_j^2} \qquad (7.5)$$

Each pixel in the index image is replaced by its gradient G(f(*i, j*)) to form a new image

2. Thresholding each pixel in the newly formed image to generate a gradient image:

$$G(f(i,j)) = \begin{cases} L_{G_f} & G(f(i,j)) \geq T \\ L_B & otherwise \end{cases} \qquad (7.6)$$

where T is a non-negative threshold and is properly selected to emphasize significant edg-

es without destroying the characteristics of smooth backgrounds. L_G and L_B are two specified luminance levels and are 0 and 255 respectively in the following experiments.

Following the above steps, a binary gradient image is formed, in which the edges and backgrounds are 0 and 255 respectively and the location of edges can be seen clearly.

The above process is applied to the index image to test whether the edges existing in the original full colour image can be retained in the index image. In the test, the black and white image corresponding to the colour image is used as the comparison reference. The following results can be obtained after analyzing the binary gradient images, see figures 7.12, 7.13, 7.14, and 7.15.

First, the randomly arranged codebook does not retain edges in the index image. This can be seen in figures 7.12(b), 7.13(b), 7.14(b) and 7.15(b), where the binary gradient images are in mess and have no meaning.

Second, the ordered codebooks can retain most edges in the index images. In the black and white images, since an edge is a brightness discontinuity without any consideration of the colour discontinuity, some edges which are caused by colour rather than brightness are not retained. For example, consider image "pepper", the pixels in the lower left part of the big green pepper in the middle of the image have close luminance values but different chrominance values. The edges in this part cannot be retained in the black and white image, as illustrated in figure 7.15(a). However, the ordered codebooks cannot only retain the brightness discontinuity but also some colour discontinuity. Take image "pepper" as an example, see figure 7.15. It can be seen that almost all the edges which exist in figure 7.15(a) also exist in figures 7.15(c), 7.15(d), 7.15(e), and 7.15(f). Besides, some edges which do not exist in figure 7.15(a) exist in figures 7.15(c), 7.15(d), 7.15(e), and 7.15(f). The edges which do not exist in figure 7.15(a) are caused by colour discontinuity. This can also be seen in images "girl1", "girl2", and "plane", see figures 7.12, 7.13 and 7.14.

Thirdly, the codebooks ordered by the centroid method, the PNN-based method and the PNN-

based/ordering function method can retain more colour discontinuity than the codebook ordered by the Y ordering method. This can be easily understood because in the Y ordering method, as long as the codewords have close luminance values, no matter how different their chrominance values are, they are put close in the codebook. Therefore, some edges caused by colour discontinuity cannot be retained in the index image. Fourthly, the PNN-based/ordering function method works best in the sense of retaining more edges, especially colour discontinuity.

Finally, in the process of the above edge detection, the thresholds used in the black and white image and the index image by the PNN-based/ordering function method, see Eq.(7.16), are smaller than the thresholds used in the other index images. This is because, in the other index images, there are many wrong edges which do not exist in the original full colour images. The reason for the wrong edges is that some codewords which are close in colour space are placed far away in the codebook. High threshold is required to remove these wrong edges. But at the same time, high threshold also removes some meaningful edges. Therefore, improper ordering of the codebook results in many wrong edges in the index image. The PNN-based/ordering function method is the best in the sense that few wrong edges are introduced in the index image.

On the whole, the PNN-based/ordering function method can order the codebook best with respect to the amount of edge retained. The resulting index images retain more edges and have less wrong edges. Next is the PNN-based method or the centroid method. Finally comes the Y ordering method. The randomly arranged codebook cannot retain any edges in the index image and the index image is meaningless.

(a) black and white image, threshold = 60

(b) random codebook, threshold = 100

(c) Y ordering method, threshold = 100

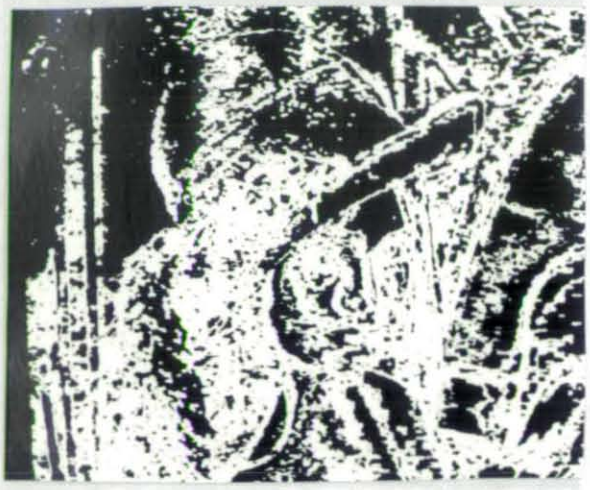
(d) centroid method, threshold = 100

(e) PNN-based method, threshold = 100

(f) PNN/ordering function method, threshold = 70

Figure 7.12 Edge detection by the Sobel operator for image "girl1"

(a) black and white image, threshold = 70



(b) random codebook, threshold = 80

(c) Y ordering method, threshold = 80

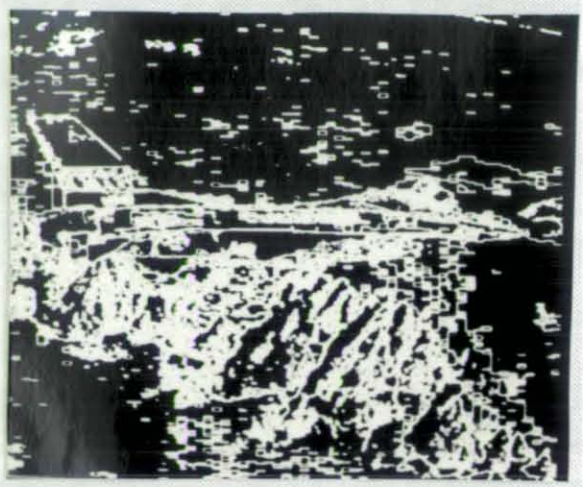
(d) centroid method, threshold = 80

(e) PNN-based method, threshold = 80

(f) PNN/ordering function method, threshold = 70

Figure 7.13 Edge detection by the Sobel operator for image "girl2"

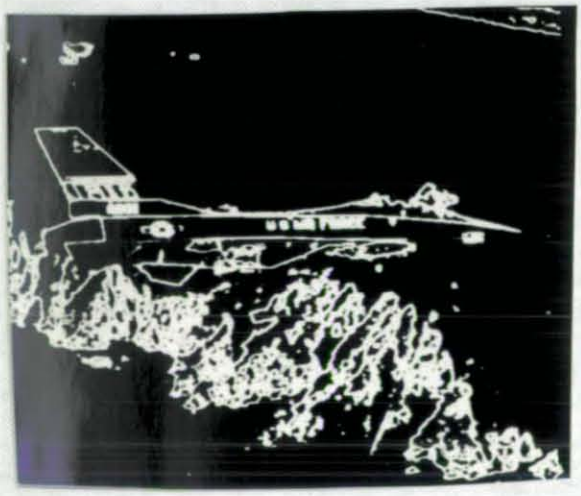
(a) black and white image, threshold = 70



(b) random codebook, threshold = 80

(c) Y ordering method, threshold = 80

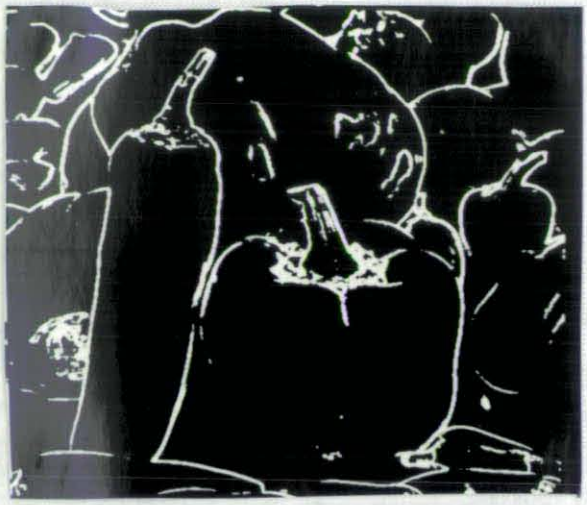
(d) centroid method, threshold = 80



(e) PNN-based method, threshold = 80

(f) PNN/ordering function method, threshold = 70

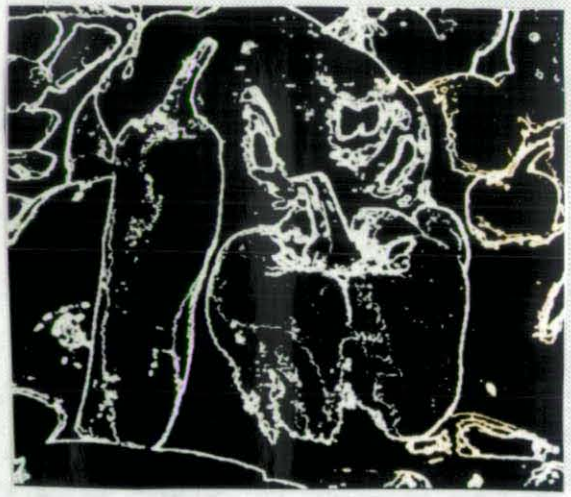
Figure 7.14 Edge detection by the Sobel operator for image "plane"



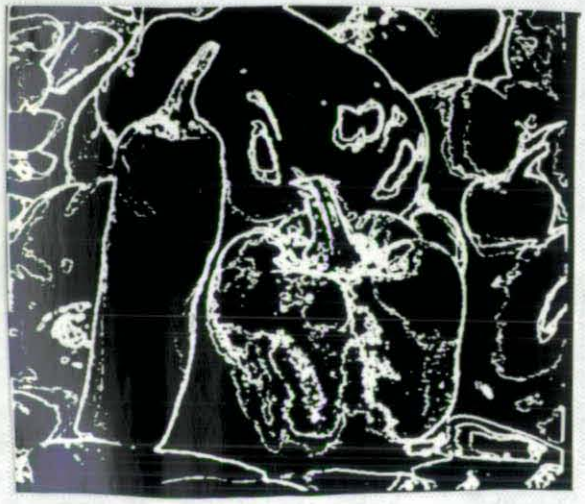
(a) black and white image, threshold = 70

(b) random codebook, threshold = 90

(c) Y ordering method, threshold = 90



(d) centroid method, threshold = 90



(e) PNN-based method, threshold = 90

(f) PNN/ordering function method, threshold = 70

Figure 7.15 Edge detection by the Sobel operator for image "pepper"

7.3.3 The DCT Processing on the Index Images

The codebooks, which are ordered by different ordering techniques, can also be tested by applying the discrete cosine transform and the threshold processing on their corresponding index images. The steps in the test are as follows:

1. Vector quantize the colour image by an ordered colour codebook
2. Apply the discrete cosine transform on the index image
3. Zigzag and threshold the coefficients and compute the required bits to encode the processed image
4. De-threshold and de-zigzag the coefficients
5. Apply the inverse discrete cosine transform to the coefficients
6. De-vector quantize the recovered index image
7. Compute the SNR on the reconstructed colour image
8. Subjectively evaluate the reconstructed colour image

The ordered colour codebooks are then compared in three aspects, namely, the required bit rate, the SNR of the reconstructed colour image and the visual quality of the reconstructed colour image. The bit rate is computed in the similar way as that in the JPEG still image coding standard (§ 5.4) with the difference that, the Symbol-1 for DC coefficient is encoded by constant 8 bits and for AC coefficients by constant 4 bits rather than using a Huffman coding table. Coding the coefficients in this way requires more bits but the coding is simpler. The bit rate can be lower if the Huffman coding is used to encode Symbol-1. The signal to noise ratio (SNR) is used as a rough indicator for the quality of the reconstructed colour image, see Eq.(7.2). The results of the four test images are shown in table 7.2, table 7.3, table 7.4, and table 7.5.

For image "girl1", the following observations can be made from table 7.2. First, when using the randomly arranged codebook to vector quantize colour image "girl1", the index image retains rather few correlation and more bits are required to represent a pixel. For example, at threshold 2, its

Threshold	Codebooks ordered by different ordering methods									
	Random		Y ordering method		Centroid method		PNN-based method		PNN-based and ordering function	
	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB
1	8.56	57	6.32	78	6.19	82	6.25	83	6.12	83
2	6.56	47	3.98	69	4.01	74	4.04	75	3.90	76
3	5.35	45	2.89	67	2.95	72	2.96	73	2.86	74
4	4.49	43	2.24	66	2.31	71	2.32	71	2.24	73
5	3.87	43	1.82	66	1.89	70	1.91	70	1.84	72

Table 7.2 Apply the DCT to the index images of image "girl1"

bit rate is 4.49 bits/pel, which is bigger than the bit rates of the other kinds of the ordered colour codebooks. Furthermore, its SNR of the reconstructed colour image is rather low and the visual quality of the reconstructed colour image is unacceptable even when the threshold is only 1. Therefore, the index image which is generated by the randomly arranged codebook has no structure. Second, though the codebook ordered by the Y ordering method has very close or even lower bit rate compared with the codebook ordered by the centroid method, or the PNN-based method, or the PNN-based/ordering function method, its SNR is much smaller than the other kinds of ordered codebooks and the visual quality of the reconstructed colour image is also worse. The reason for this is that, when the codebook is ordered by the Y ordering method, only the luminance of the colour is considered in the ordering without the chrominance of the colour. As a result, though the close codewords which have close luminance values are put close in the codebook, it is not necessary that the codewords which are close in the codebook represent similar colours, because the codewords having close or the same luminance level can represent quite different colours. Consequently, the distortion in the recovered index image after the DCT coding may result in more wrong

colour recoveries which degrades the reconstructed colour image.

Third, for image "girl1", the codebooks ordered by the centroid method or the PNN-based method have quite close results both in the bit rate and the SNR. Comparatively, for image "girl1", the codebook ordered by the PNN-based method has slightly lower SNR value and better visual quality of the reconstructed colour image. Finally, when the codebook is ordered by the PNN-based/ordering function method, the bit rate and the SNR are more or less improved. For example, its SNR at threshold 4 is the same as that of the PNN-based method at threshold 3 and the visual quality of the reconstructed colour image is better. On the whole, for image "girl1", the PNN-based/ordering function method is the best regarding to the high SNR, low bit rat and good visual quality of the reconstructed colour image, followed by the PNN-based method or the centroid method and the Y ordering method. For the other three test images, similar results are also obtained.

Threshold	Codebooks ordered by different ordering methods									
	Random		Y ordering method		Centroid method		PNN-based method		PNN-based/ ordering function	
	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB
1	9.63	32	6.44	49	6.63	57	6.28	58	5.59	58
2	7.68	27	3.84	43	4.20	49	3.82	50	3.23	51
3	6.41	26	2.67	42	3.02	47	2.69	48	2.23	49
4	5.48	26	2.01	42	2.33	46	2.06	47	1.69	48
5	4.77	25	1.60	42	1.88	46	1.66	47	1.37	47

Table 7.3 Apply the DCT to the index images of image "girl2"

For image "girl2", the following points can be observed from table 7.3. First, the randomly arranged codebook cannot retain any structure in the index image. It requires high bit rate and has

rather low SNR and poor visual quality of the recovered colour image. Second, the Y ordering method has low bit rate, but low SNR and poor visual quality of the reconstructed colour image compared with the other kinds of ordered codebooks. Third, the PNN-based method is better than the centroid method and it has lower bit rate, higher SNR and better visual quality of the reconstructed colour image. Finally, the ordering function can improve the codebook ordering. The codebook ordered by the PNN-based/ordering function method can retain more structure in the index image than that by the PNN-based method. It has lower bit rate, higher SNR and better visual quality of the reconstructed colour image. Therefore, for image "girl2", the PNN-based/ordering function method is the best, followed by the PNN-based method, the centroid method, the Y ordering method and the randomly arranged codebook.

Threshold	Codebooks ordered by different ordering methods									
	Random		Y ordering method		Centroid method		PNN-based method		PNN-based/ ordering function	
	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB
1	5.94	48	4.34	66	4.37	82	4.57	83	4.39	88
2	4.44	40	2.72	59	2.77	74	2.95	73	2.75	80
3	3.61	36	2.04	56	2.07	71	2.21	70	2.04	76
4	3.04	33	1.65	55	1.67	69	1.78	68	1.64	74
5	2.64	32	1.40	55	1.42	68	1.51	67	1.38	73

Table 7.4 Apply the DCT to the index images of image "plane"

For image "plane", the following results can be taken from table 7.4. First, the randomly arranged codebook cannot retain structure in the index image. It has high bit rate and rather low SNR and poor visual quality of the reconstructed colour image. Second, the Y ordering method has very low bit rate, and sometimes the bit rate is even lower than the centroid method and the PNN-based method, but it has low SNR and poor visual quality of the reconstructed colour image. For exam-

ple, its SNR of the reconstructed colour image at threshold 1 is lower than that of the centroid method and the PNN-based method at threshold 5. And the visual quality of the reconstructed colour image at threshold 2 is worse than that of the centroid method at threshold 5. Third, for image "plane", the centroid method is better than the PNN-based method regarding to the low bit rate, high SNR and good visual quality of the reconstructed colour image. Finally, the ordering function can improve the ordering. The codebook ordered by the PNN-based/ordering function method can retain more structure in the index image than that by the PNN-based method. It has lower bit rate, higher SNR and better visual quality of the reconstructed colour image. On the whole, for image "plane", the PNN-based/ordering function method is the best, followed by the centroid method, the PNN-based method, the Y ordering method and the randomly arranged codebook.

Threshold	Codebooks ordered by different ordering methods									
	Random		Y ordering method		Centroid method		PNN-based method		PNN-based/ ordering function	
	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB
1	9.69	25	6.37	35	6.30	43	6.39	44	5.63	47
2	7.80	21	3.71	31	3.72	37	3.89	38	3.13	42
3	6.56	20	2.50	30	2.56	35	2.74	36	2.12	40
4	5.65	20	1.84	30	1.93	34	2.08	35	1.59	39
5	4.94	20	1.45	30	1.54	34	1.67	34	1.28	38

Table 7.5 Apply the DCT to the index images of image "pepper"

For image "pepper", the following results can be yielded from table 7.5. First, the randomly arranged codebook cannot retain any structure in the index image. Second, the Y ordering method has low bit rate, but low SNR and poor visual quality of the reconstructed colour image compared with the other kinds of ordered codebooks. Third, the centroid method has lower bit rate but lower

SNR and poorer visual quality of the reconstructed colour image than the PNN-based method. Finally, the ordering function can improve the codebook ordering. The codebook ordered by the PNN-based/ordering function method can retain more structure in the index image than that by the PNN-based method. It has lower bit rate, higher SNR and better visual quality of the reconstructed colour image. On the whole, for image "pepper", the PNN-based/ordering function method is the best, followed by the PNN-based method, the centroid method, the Y ordering method and the randomly arranged codebook.

For the four experimental colour images, the PNN-based/ordering function method is the best ordering technique among the techniques tested in the sense of low bit rate, high SNR and good visual quality of the reconstructed colour image. However, the visual quality of the reconstructed colour image is un-acceptable when threshold is much larger than 2. When the threshold is 2, the bit rates for the four test images "girl1", "girl2", "plane" and "pepper" are 3.95, 3.23, 2.75 and 3.13 respectively and the compression ratios are 1:6.0, 1:7.4; 1:8.7 and 1:7.7. These compression ratios are not high enough for practical applications. The reasons for the low compression are that, though the ordering techniques can order the colour codebook quite well, there are still some rough places, where adjacent codewords in the codebook have big distance in colour space, such places as the join of two clusters, and there are codewords which are close in colour space are far away in the codebook, such as the junction codewords. The strategy of the artificial codeword insertion, see § 3.4, has been proposed to improve codebook ordering. Artificial codeword insertion has been applied to the codebooks ordered by the PNN-based/ordering function method. The results are listed in table 7.7, while the results of the PNN-based/ordering function method alone are listed in table 7.6 for comparison.

It can be seen that artificial codeword insertion improves the codebook ordering quite remarkably. First, for the four images, the SNR of the reconstructed colour images is increased substantially. For example, consider image "plane", before the insertion the SNR at thresholds 1, 2, 3, 4, 5 is 88, 80, 76, 74 and 73 respectively, as shown in table 7.6, while, after artificial codeword insertion, the SNR is 93, 86, 82, 79 and 78 respectively, as shown in table 7.7. Second, the visual quality of the

Threshold	Images							
	Girl1		Girl2		Plane		Pepper	
	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB
1	6.12	83	5.59	58	4.39	88	5.63	47
2	3.90	76	3.23	51	2.75	80	3.13	42
3	2.86	74	2.23	49	2.04	76	2.12	40
4	2.24	73	1.69	48	1.64	74	1.59	39
5	1.84	72	1.37	47	1.38	73	1.28	38

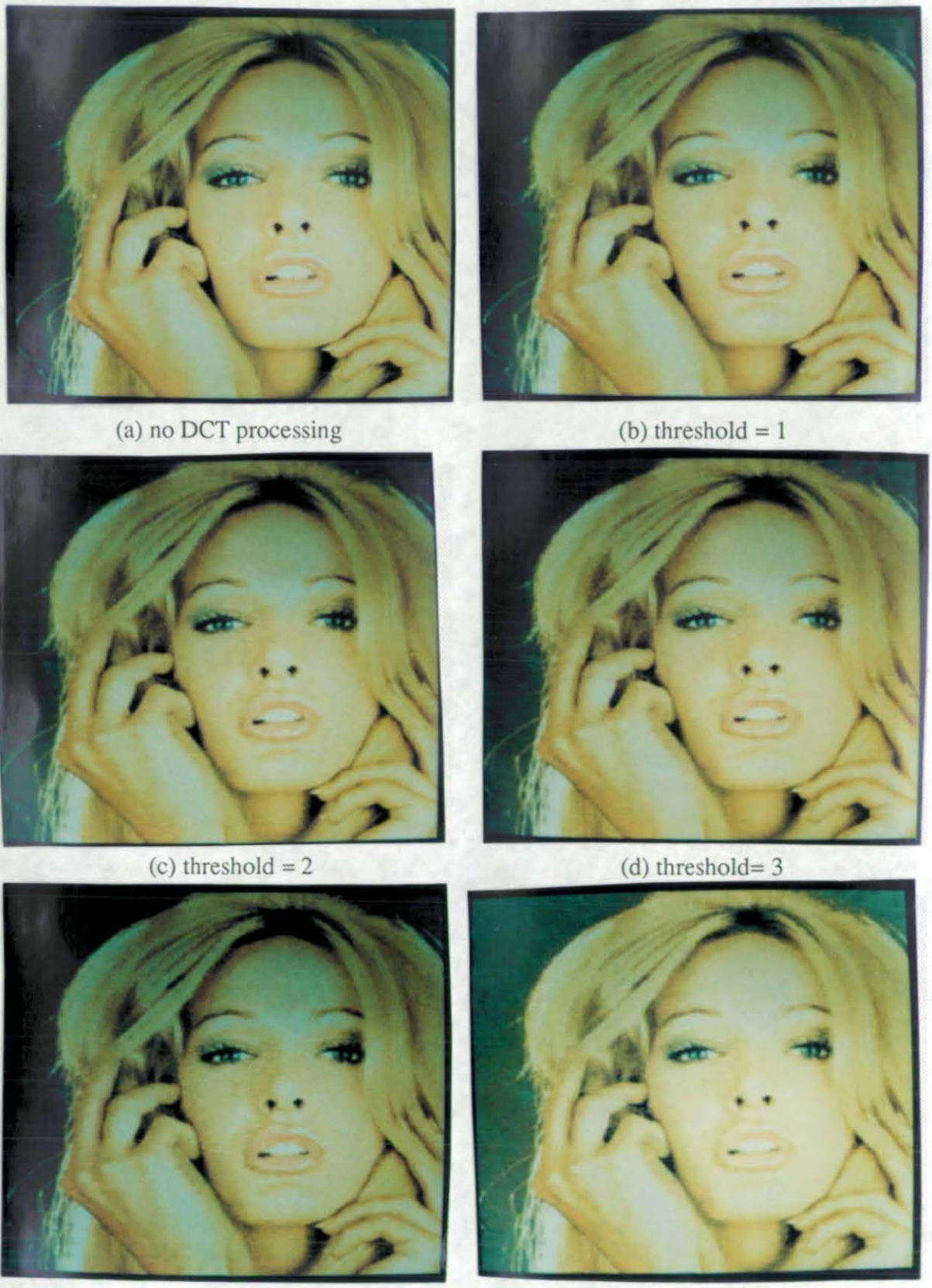
Table 7.6 The experimental results before the artificial codeword insertion

Threshold	Images							
	Girl1		Girl2		Plane		Pepper	
	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB	Rate bits/pel	SNR dB
1	6.45	90	5.89	67	5.21	93	6.20	51
2	4.33	84	3.61	60	3.54	86	3.81	46
3	3.27	80	2.61	56	2.70	82	2.74	43
4	2.63	77	2.05	54	2.21	79	2.14	42
5	2.19	75	1.70	53	1.88	78	1.76	41

Table 7.7 The experimental results after the artificial codeword insertion

reconstructed colour image is also improved and at the same threshold, the reconstructed colour image after insertion looks better than that before the insertion. For example, for image "girl1", the visual quality at threshold 3 before the insertion is unacceptable, while after the insertion, the visual quality of the reconstructed colour image at threshold 3, even 4 is quite good, see figure 7.16. Finally, the bit rate is decreased when the visual quality of the reconstructed colour image is good or acceptable. For images "girl1", "girl2" and "pepper", when the threshold is up to 4, the visual quality of their reconstructed colour images is quite good. In this case, the bit rates are 2.63, 2.05, 2.21 and 2.14 respectively and their compression ratios are then 1:9.1, 1:11.7, 1:10.9, and 1:11.2 respectively. For image "plane", when the threshold is up to 5, the visual quality of the reconstructed colour image is still good, and the bit rate is 1.88, and the compression ratio is 1:12.8.

The reconstructed colour images at thresholds 1, 2, 3, 4, 5 of the images 'girl1', 'girl2', and 'plane' are shown in figures 7.16, 7.17, 7.18. (It is unfortunate to point out that because of photography problem, image "pepper" only has two reconstructed colour images at thresholds 1, and 2.) Normally, when the threshold is up to 4, the reconstructed colour images look good. For image "plane", the quality of the reconstructed colour image is still good at threshold 5. But it can be noted that when threshold is big, say 5, the degradation in the reconstructed colour image is quite obvious to be perceived. It has been explained in chapter 3 that the value of a pixel in the index image is the index of the closest mapped codeword in the codebook. The value of a distorted pixel will cause wrong colour recovery. Though the codebook is ordered, there are still some places where the arrangements of the codewords cannot satisfy the two ordering conditions well. Therefore, when the distortion is large, more distortion will be introduced in the recovered index image and the distortion will result in wrong colour recovery. The other problem is that some significant edges are blurred even though the threshold is very small. Some smooth edges are zigzagged. For example in figure 7.18(b), the edge along the tail of the plane is zigzagged. Because after the DCT and the threshold process, the recovered image blocks are actually averaged to some extent. Therefore, in the blocks where there are sharp edges, after the averaging more distortion is then introduced. Since the pixel in the index image is the index of the mapped closest codeword in the codebook, it is easy to have blocky effect along the edges.



(a) no DCT processing (b) threshold = 1

(c) threshold = 2 (d) threshold= 3

(e) threshold = 4 (f) threshold = 5

Figure 7.16 Reconstructed colour images of image "girl1"

(a) no DCT processing (b) threshold = 1

(c) threshold = 2 (d) threshold= 3

(e) threshold = 4 (f) threshold = 5

Figure 7.17 Reconstructed colour images of image "girl2"

(a) no DCT processing

(b) threshold = 1

(c) threshold = 2

(d) threshold= 3

(e) threshold = 4

(f) threshold = 5

Figure 7.18 Reconstructed colour images of image "plane"

(a) no DCT processing

(b) threshold = 1

(c) threshold = 2

Figure 7.18 Reconstructed colour images of image "pepper"

7.3.4 Summary

The quality of the ordered codebook depends both on the codebook and the ordering techniques. The characteristics of a codebook are determined by the image to be coded. For the images which have a lot of saturated colours, such as image "pepper", or have a lot of achromatic colours, such as image "plane", their codebooks are easier to order. They have low ordering function values, low bit rates and good visual quality of the reconstructed colour images after the DCT processing. They can retain more edges in the index image. For the images which have a lot of close colours, such as images "girl1" and "girl2", their codebooks are more difficult to order. They have higher ordering function values, higher bit rates etc.

Among the four ordering techniques, the PNN-based/ordering function method is the best regarding low bit rate, high SNR and better visual quality of the reconstructed colour image after the DCT processing, low ordering function value and more edges retained and less wrong edges in the index image. Next come the centroid method and the PNN-based method. They give quite similar results, such as the bit rates, SNR, ordering function etc. In most cases, the PNN-based method works better than the centroid method. Finally comes the Y ordering method. It can retain a lot of significant edges and some correlation in the index image. But it is not suitable for the image processing which introduces distortion into the index image, such as information lossy compression techniques. The randomly arranged codebook cannot retain any structure in the index image. Artificial codeword insertion has been shown to improve the codebook ordering. It is especially useful in improving the reconstructed colour image quality when there is distortion in the index image.

The conclusion can be reached from the above tests that if the codebook is ordered by a proper ordering method, such as the PNN-based/ordering function method together with the artificial codeword insertion, then using the ordered codebook to vector quantize the colour image gives satisfactory results. The practical application of this process is that further image processing, such as edge detection, image compression techniques etc., can be applied to the index image. Therefore, colour image processing is transformed to index image processing which works in the same

way as black and white image processing. This not only simplifies colour image processing but also saves the memory space and the system expense required in the processing. Consequently, the three band colour image is transformed into a single band index image and the colour image vector representation is transformed into a scalar representation.

Chapter 8

Conclusions

In this thesis, the possibility of representing colour images by scalar variables is proposed. In this kind of colour image representation, the colour codebook ordering is the most crucial step. Several strategies for the colour codebook ordering have been designed and tested. The examples are the Y ordering method, the centroid method, the PNN-based method, and the PNN-based/ordering function method. The experimental results show that the PNN-based/ordering function method together with artificial codeword insertion is the best way of colour codebook ordering among those ordering methods so far discussed. The colour codebook which is ordered by the PNN-based/ordering function method and artificial codeword insertion can retain a lot of structure, such as edges and correlation, in the index image so that edge detection and image compression techniques can be applied to the index image. This indicates that further image processing can be applied to the index image if the colour codebook is well ordered. In this sense, a three-band colour image is transformed into a single band index image with a well ordered colour codebook. Conventionally, the colour image processing is realized by applying image processing techniques to the three bands separately. But now, colour image processing can alternatively be carried out on the index image in the same way as on a monochrome image. In this way, it not only simplifies colour image processing but also reduces the processing complexity and memory requirement up to one third. Additionally, the image processing techniques which have been developed for monochrome images can be applied to index images straightforwardly.

The research on the colour image representation by scalar variables is at its early stage of research

and further research is worthwhile. In the future, it would be interesting to carry out further research associated with the ordering function in colour codebook ordering. It can be seen that the ordering function plays an important role in the process of colour codebook ordering. It can quantitatively evaluate how well the ordering conforms with the two ordering conditions. What is more, it can be used in refining the ordering. The ordering function proposed in this thesis works quite well. This can be seen from the results of the experiments discussed in chapter 7. Further research can be carried out in the following aspects.

First, a better colour space rather than the RGB colour space should be used. In the colour codebook ordering, it is required that the codewords which represent close colours should be put close in the codebook and the codewords which are close in the codebook should represent close colours. Therefore, it is desirable that the close ordinal number in colour space is computed in the colour space in which the colour tristimulous difference is more consistent with perceived colour difference. Second, a better colour distance measurement, rather than the Euclidean distance, should be used in the computation of the close ordinal number in colour space. In the computation of the close ordinal numbers in colour space for a reference codeword, the codewords are first ordered according to their distance in colour space from the reference codeword. It is assumed here that the distance between two codewords in colour space can describe the difference of the colours they represent and the smaller the distance is, the closer the colours they represent. But it is obvious that the Euclidean distance between two codewords does not necessarily correlate with the perceived colour difference. This is because the Euclidean distance does not consider the aspect of the human visual system.

Finally, the arrangement of the junction codewords should be taken into account in the ordering function. A junction codeword is close to several codewords with almost the same small distance. According to the two ordering conditions, it is required that the junction codeword should be put close to all its close codewords in the codebook. But in a linear codebook, this cannot be realized. In the techniques proposed in this thesis, the junction codeword is either taken out to form a separate cluster or put close to any of its close codeword. Better arrangement of the junction codeword

would be that the junction codeword is put close to that codeword to which it is most likely to be close to in the index image. Then, in the calculating of the function f(i), weight should be used to reflect the importance of a codeword in the image to a reference codeword. Therefore, if this arrangement of the junction codewords is reflected in the ordering function, the ordering refinement, by reducing the ordering function, can achieve better colour codebook ordering.

Appendix A

The Inmos Transputer T222

The Inmos T222 is a 16 bit version with on-chip RAM for high speed processing, an external memory interface and four standard Inmos communication links. The T222 has 4 Kbytes of fast on-chip memory and can access 64 Kbytes of external memory The memory is byte addressed with words aligned on two-byte boundaries. Its links allow networks of transputers to be constructed by direct point-to-point connections with no external logic. The links support the standard operating speed of 10 Mbits/sec, but also at 5 or 20 Mbits/sec. Each Inmos T222 link can transfer data bi-directionally at up to 2.05 Mbytes/sec. Its instruction set achieves efficient implementation of high level languages and provides direct support for the Occam model of concurrence when using either a single transputer or a network. Procedure calls, process switching and typical interrupt latency are submicrosecond and this feature is very important as image processing involves moving lots of data, i.e. many interrupts. The T222 also supports block in and block out operations in Occam, which provides an efficient and easy way to control and communicate with fast external memory, such as FIFOs. The main features of the T222 are summarized as follows:

. 16 bit architecture
. 50 ns internal cycle time
. 20 MIPS (peak) instruction rate
. 4 Kbytes on-chip static RAM
. 40 Mbytes/sec sustained data rate to internal memory
. 64 Kbytes directly addressable external memory
. 20 Mbytes/sec sustained data rate to external memory
. 950 ns response to interrupts

. Four INMOS serial links 5/10/20 Mbytes/sec per link

. Internal timers of 1 μs and 64 μs

. Boot from ROM or communication links

. Single 5 MHz clock input

. Single +5V ± 5% power supply

Appendix B

The CY7C408A FIFO Chip

The A121 chip has 44 pins, see figure B.1. The signal of each pin is explained as follows:

. CLK: The clock input signal CLK controls the timing of the input and output on the three dedicated interface, and controls the progress of data through the multipliers and transposition RAM.

. GO: The GO signal is used to indicate the start of a block. When it is sampled high, the data on the Din port is the first sample of the block. Data is always assumed to be valid for the 64 cycles from the start of a major cycle, meanwhile the GO signal is ignored. Blocks of data may be processed at any time and any spacing between the major blocks, by toggling the GO signal as necessary.

. Din[11-0]: The data input ports Din[11-0] are sampled on every clock cycle and for a further 63 cycles, from when GO is sampled high. Data must be valid on the rising edge of CLK for each of the 64 cycles.

. Dout[11-0]: The data output ports Dout[11-0] will be valid for periods spanning 64 clock cycles. The data will be valid on the rising edge of the clock, exactly 128 cycles after the data was sampled on the input.

. SEL[1-0]: The mode select input port is sampled on the rising edge of CLK, when GO is active, at the start of a block of data. This fixes the selected mode for the entire data block. There are 4 possible modes of operation, and these 4 modes select 1 of the four coefficient ROMs.

. Mode 0, SEL[1-0]=00, bank 0. This selects the DCT function

. Mode 1, SEL[1-0]=01, bank 1. This selects the IDCT function

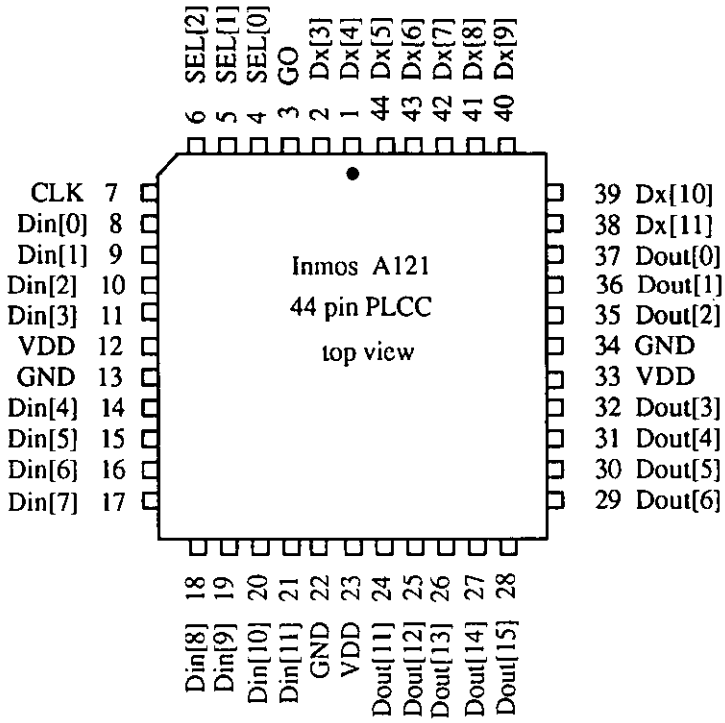


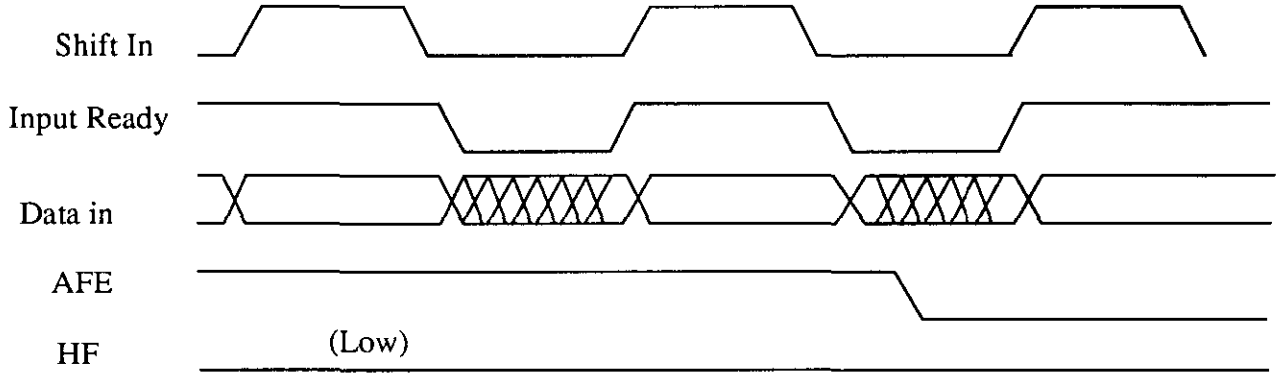
Figure B.1 The Inmos A121 44 pin PLCC top view

Appendix C

The Data In and Out Timing Diagram of the FIFO

The data in and out diagram of the CY7C408A FIFO chip:

Data In Timing Diagram



Data Out Diagram

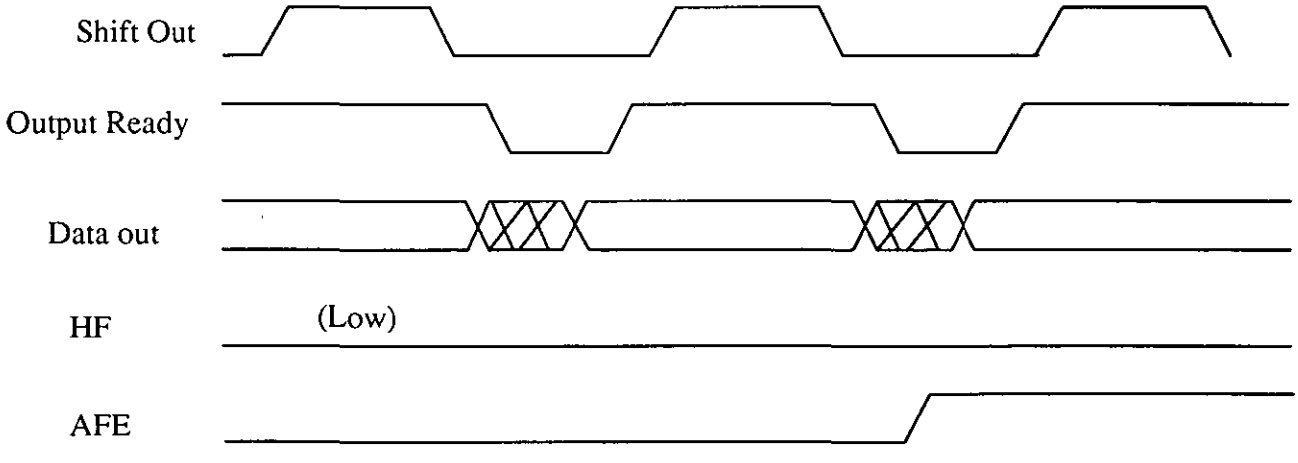


Figure C.1 Data in and data out timing diagram

Appendix D

The Program in the EPLD610 Chip

The configuration of EPLD610 chip is shown in figure D.1:

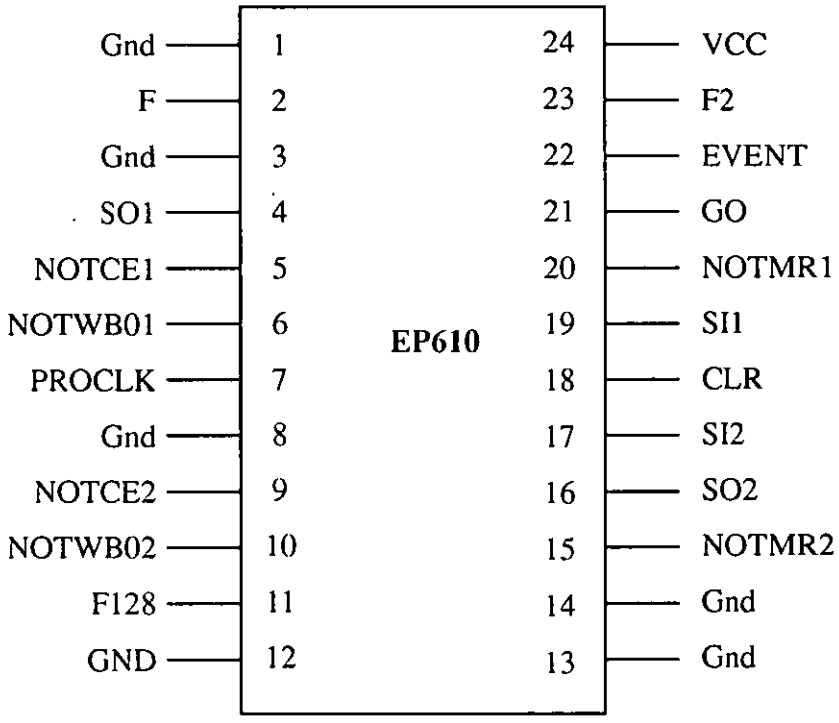


Figure D.1 The configuration of the EP610

The control program in the EP610 is as follows:

```
E610
controlling the DCT board
NETMAP Version 3.0, Baseline 15, 8/3/1985
```

Part: EP610

INPUTS: F128@11, F@2, F2@23, NOTCE1@5, NOTWB01@6, PROCLK@7, NOTCE2@9, NOTWB02@10

OUTPUTS: SO1@4, GO@21, NOTMR1@20, EVENT@22, SI2@17, SO2@16, NOTMR2@15, SI1@19, CLR@18

NETWORK:

SO1 = CONF (so1, VCC)

GO = RONF (VCC, f128clk, f2, GND, VCC)

NOTMR1 = CONF (notmr1, VCC)

EVENT = RONF (VCC, notf128, f2, GND, VCC)

SI2 = CONF (si2, VCC)

SO2 = CONF (so2, VCC)

NOTMR2 = CONF (notmr2, VCC)

SI1 = RONF (VCC, proclk, clear, GDN, VCC)

CLR, clear = COIF (clr, VCC)

clr = OR (notce1, notwb01)

notce1 = INT (NOTCE1)

notwb01 = INP (NOTWB01)

proclk = INP (PROCLK)

notmr2 = OR (notce2, notwb02)

notce2 = INP (NOTCE2)

notwb02 = INP (NOTWB02)

so2 = AND (ce2, notwb02)

ce2 = NOT (notce2)

si2 = AND (notf, f128)

notf = NOT(f)

```
f128clk = CLKB (f128)
f128 = INP (F128)
f = INP (F)
notf128 = NOT (f128)
f2 = INP (F2)
notmr1 = OR (notce1, wb01)
wb01 = NOT (notwb01)
wb01 = NOT (notwb01)
so1 = AND (f, f128)
END$
```

Bibliography

[Ahme74] N. Ahmed, T. Natarajan and K. Rao: "Discrete Cosine Transform", IEEE Trans. on Communications, January 1974.

[Bala91] R. Balasubramanian, J. Allebach: "A New Approach to Palette Selection for Colour Images", SPIE vol. 1453, Human Vision Visual Processing, and Digital Display, pp. 58-69, 1991.

[Brau87] G. Braudaway: "A Procedure for Optimum Choice of a Small Number of Colours from a Large Colour Palette for Colour Imaging", Electronic Imaging 87, San Francisco, CA, 1987.

[Burn88] A. Burns: "Programming in Occam2", Addison-Wesley Publishing Company, 1988.

[Chen77] W. Chen, C. Smith, and S. Fralick: "A Fast Computational Algorithm for the Discrete Cosine Transform", IEEE Trans. on Com., vol. Com-25, no. 9, pp. 1004-1009, September 1977.

[Chen77] W. Chen, C. Smith: "Adaptive Coding of Monochrome and Colour Images", IEEE Trans. on Com, vol. COM-25, no. 11, pp. 1285-1292, November 1977.

[Chen84] W. Chen, W. Pratt: "Scene Adaptive Coder", IEEE Trans. on Com., vol. COM-32, no. 3, pp. 225-232, March 1984.

[Chen91] S. Chen, W. Hsieh: "Fast Algorithm for VQ Codebook Design", IEE Proceedings-I, vol. 138, no. 5, pp. 357-362, October 1991.

[Clar85] R. Clarke: "Transform Coding of Image", Academic Press, 1985.

[Cypr89] Cypress Semiconductor: "CMOS BiMos Data Book", pp. 5.34-5.47, 1989.

[Davi75] L. Davis: "A Survey of Edge Detection Techniques", Computer Graphics and Image Processing, pp. 248-270, 1975.

[Eina87] G. Einarsson, G. Roth: "Data Compression of Digital Colour Pictures", Computer & Graphics, vol. 11, no. 4, pp. 409-426, 1987.

[Equi89] W. Equitz: "A New Vector Quantization Clustering Algorithm", IEEE Trans. on ASSP, vol. 37, no. 10, pp. 1568-1575, October 1989.

[Este90] M. Ester: "Image Quality and Viewer Perception", 1990 ISAST, pp. 51-63.

[Gers82] A. Gersho and B. Ramamurthi: "Image Coding Using Vector Quantization", 1982 IEEE.

[Gers85] R. Gershon: "Aspects of Perception and Computation in Colour Vision", Computer Vision, Graphics, and Image processing, pp. 244-277, 1985.

[Gold91] N. Goldberg: "Colour Image Quantization for High Resolution Graphics Display", Image and Visual Computing, vol. 9, no. 5, pp. 303-312, October 1991.

[Gonz87] R. Gonzales, P. Wintz: "Digital Image Processing", Addison-Wesley Publishing Company, second edition, 1987.

[Gran79] D. Granrath, B. Hunt: "A Two-Channel Model of Image Processing in the Human Retina", Proc. SPIE, vol. 199, pp. 126-133, August, 1979.

[Gray84] R. Gray: "Vector Quantization", IEEE ASSP Magazine, pp. 4-29, April 1984.

[Habi74] A. Habibi: "Hybrid Coding of Pictorial Data", IEEE Trans on Com., vol. COM-22, no. 5, May 1974.

[Heck82] P. Heckbert: "Colour Image Quantization for Frame Buffer Display", Computer Graphics, vol. 16, num. 3, pp. 297-307, July 1982.

[Helm90] R. Helms: "Introduction to Image Technology", IBM systems Journal, vol. 29, no. 3, pp. 313-332, 1990.

[Huan89] A. Huang,: "Image Compression: The Emerging Standard for Colour Images", IEEE Computing Futures, 1989.

[Huff52] D.Huffman: "A Method for the Construction of Minimum Redundancy Codes", Proceedings of the I. R. E., vol. 40(10), pp.1098-1101, September 1952.

[Inmo88] Inmos Limited: "Transputer Development System", Prentice Hall, 1988.

[Inmo89[1]] Inmos Limited: "The Transputer Data Book", Redwood Burn Ltd., 1989.

[Inmo89[2]] Inmos Limited: "Digital Signal Processing Data Book", Bath Press Ltd., pp. 77-96, 1989.

[Jain81] A. Jain: "Image Data Compression: A Review", Proceedings of the IEEE, vol. 69, no. 3, pp. 349-389, March 1981.

[Kama82] F. Kamangar, K. Rao: "Fast Algorithms for the 2-D Discrete Cosine Transform", IEEE Trans. on Com. vol. COM-31, no. 9, September 1982.

[Kunt85] M. Kunt, A. Ironomopoulos, M. Kocher: "Second-Generation Image Coding Techniques", Proceedings of the IEEE, pp. 549-591, April 1985.

[Kunt87] M. Kunt, M. Benard, R. Leodardi: "Recent Results in High-Compression Image Coding", IEEE Trans. on Circuits and Systems, vol. CAS-34, no. 11, pp.1306-1336, November, 1987.

[Lang84] G. Langdon: "An Introduction to Arithmetic Coding", IBM J. Res. Develop., vol. 28, no. 2, pp. 135-149, March 1984.

[Lava90] F. Lavagetto, S. Zappatore: "Comparative Evaluation of Different Techniques in Image Vector Quantization", SPIE vol. 1349, Applications of Digital Image Processing XIII (1990), pp. 192-202.

[Limb77] J. Limb, C. Rubinstein, J. Thompson: "Digital Coding of Colour Video Signals-A Review", IEEE Trans. on Communication, vol. COM-25, no. 11, November 1977.

[Limb79] J. Limb: "Distortion Criteria of the Human Viewer", IEEE Trans. on System, Man and Cybernetic, vol. SMC-9(12), pp. 778-793, December 1979.

[Lind80] Y. Linde, A. Buzo, R. Gray: "An Algorithm for Vector Quantization Design", IEEE Trans. on Com., vol. COM-28, no. 1, pp. 84-95, January 1980.

[Maen89] J. Maeng, D. Hein: "A Low-Rate Video Coding Based on DCT/VQ", Visual Communication and Image Processing IV (1989), pp. 267-273.

[Mare86] J. Marescq, C. Labit, "Vector Quantization in Transformed Image Coding", ICASSP 86, pp. 145-148, 1986.

[Macd90] L. Macdonald: "Using Colour Efficiently in Displays for Computer-Human Interface", Displays: Technology and Applications, vol. 11, pp. 129-141, July 1990.

[Makh87] J. Makhoul: "A Fast Cosine Transform in One and Two Dimensions", IEEE Trans. on ASSP, vol. ASSP-28, no. 1, February 1987.

[Mitc91] J. Mitchell, "Evolving JPEG Colour Data Compression Standards", Standards for Electronic Imaging Systems, 91.

[Mitr89] S. Mitra, I. Zarrinnaal, Y. Wang: "Digital Processing of Colour Images", SPIE vol. 1077, Human Vision, Visual Processing and Digital Display, pp. 132-135, 1989.

[Miya85] M. Miyahara, K. Kotanl: "Block Distortion in Orthogonal Transform Coding-Analysis, Minimization and Distortion Measure". IEEE Trans. on Com, no. 1, January 1985.

[Murc84] G. Murch, Tektronic: "Physiological Principles for the Effective Use of Colour", IEEE CG&A, pp. 49-54, November 1984.

[Nasr88] N. Nasrabadi, R. King: "Image Coding Using Vector Quantization: A Review", IEEE Trans. on Com., vol. 36, no. 8, pp. 957-971, August 1988.

[Netr88] A. Netravali, B. Haskell: "Digital Pictures: Representation and Compression", Plenum Press, 1988.

[Nibl86] W. Niblack: "An Introduction to Digital Image Processing", Prentice-Hall International Ltd., 1986.

[Prat71] W. Pratt: "Spatial Transform Coding of Colour Images", IEEE Trans. on Communication Technology, vol. COM-19, December 1971.

[Prat81] W. Pratt, J. Kana, H. Andrews: "Hadmard Transform Image Coding", The Radio and Electronic Engineering, vol. 51, no. 11, 1981.

[Pres91] K. Preston: "Who Needs 24-bit Colour?", PHOTONICS Spectra, pp. 119-121, April 1991.

[Rama86] B. Ramamurthi, A. Gersho: "Classified Vector Quantization of Images", IEEE Trans. on Com., vol. COM-34, no. 11, pp. 1105-1115, November 1986.

[Robi77] G. Robinson: "Colour Edge Detection", IEEE Trans. on Systems, Man, and Cybernetic, vol. SMC-7, no. 11, pp. 525-530, November 1977.

[Shan76] K. Shanmugam: "Comments on Discrete Cosine Transform", IEEE Trans. on Computers, vol. C-23, March 1975.

[Sakr77] D. Sakrison: "On the Role of the Observer and a Distortion Measure in Image Transmission", IEEE Trans. on Com., vol. COM-25, no.11, November 1977.

[Truc81] J. Truckenbrod: "Effective Use of Colour in Computer Graphics", Computer Graphics, vol. 15, no. 3, pp. 83-90, August 1981.

[Tsen78] B. Tseng, W. Miller: "On Computing the Discrete Cosine Transform", IEEE Trans. on Computers, vol. C-27, October. 1978.

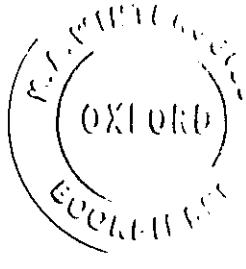
[Wall91] G. Wallace: "The JPEG Still Picture Compression Standard", Com. of the ACM, vol. 34, no.4, pp. 31-44, April 1991.

[Wang91] H. Wang: "A VQ/DCT Coding Scheme for Colour Images", ICC'91, pp. 236-240, June 1991.

[Wint72] P. Wintz: "Transform Picture Coding", Proceedings of the IEEE, vol. 60, no. 7, July 1972.

[Witt87] I. Witten, R. Neal, and J. Cleary: "Arithmetic Coding for Data Compression", Communications of the ACM, vol. 30(6), pp. 520-540, June 1987.

[Youn86] T. Young, K. Fu: "Handbook by Pattern Recognition and Image Processing", Academic Press, pp. 201-204, 1986.


OXFORD