1995

A video-on-demand system over a TCP/IP computer network using vector transform coding

Keren Hu
Lehigh University

AUTHOR:

Hu, Keren

TITLE:

A Video-On-Demand System Over a TCP/IP Computer Network Using Vector Transform Coding

DATE: May 28, 1995

A VIDEO-ON-DEMAND SYSTEM OVER A TCP/IP COMPUTER NETWORK USING VECTOR TRANSFORM CODING

by
Keren Hu

A Thesis
Presented to the Graduate Committee
of Lehigh University
in Candidacy for the Degree of
Master of Science
in
Electrical Engineering

Lehigh University
1995

This thesis is accepted in partial fulfillment of the requirements for the degree of Master of Science.

4/28/95
(Date)

Thesis Advisor

Chairperson of Department

Acknowledgments

I am deeply indebted to Dr. Weiping Li for his guidance and patience during the course of this study. Also, I'd like to thank my husband Shipeng Li for his never-ending support in my work and life. My sincere appreciation goes to the DSP group members: Qinhong Cao, Fan Lin, Scott Segan, Asaf Sofu, John Wus, for they have provided the wonderful academic atmosphere for me to work in.

I would like to thank the staff at Lehigh University, especially Linda and Maggie in the EECS graduate office, for all of their help.

I would also like to thank my parents and my sister for their unfailing love and support which helps me to keep improving myself.

Contents

Acknowledgments iv

Abstract 1

1 Introduction 2

2 Background 5

2.1 An Image Model 5

2.1.1 Introduction 5

2.1.2 Major Color Spaces 7

2.1.3 Color Quantization and Dithering 9

2.2 Image and Video Compression Overview 12

2.3 Transform Coding 15

2.4 Scalar Quantization 17

2.5 Vector Quantization 20

2.5.1 Vector Quantization Problem 20

2.5.2 Distortion Measures 22

2.5.3 Codebook Design 23

2.5.4 Encoding 25

2.5.5 Vector Transform Coding 26

2.6 Entropy Coding 27

2.7 Video Compression Standards 29

2.7.1 JPEG 29

2.7.2 H.261 30
2.7.3 MPEG-1 31
2.7.4 MPEG-2 32
2.8 About the Network Platform 34
2.8.1 What is TCP/IP 34
2.8.2 World Wide Web (WWW) 36

3 System Architecture **39**
3.1 Overview 39
3.2 Server 39
3.2.1 Video Encoding 41
3.2.2 Processing Module 42
3.2.3 Network Interface Module (NIM) 42
3.3 Client 43
3.3.1 Network Interface Module 44
3.3.2 Decoding Module 44
3.3.3 Synthesizing Module 44
3.3.4 Display Module 45
3.3.5 User Interface 46

4 Implementation of the VOD System **47**
4.1 Encoding of Video 47
4.2 Transmitting and Receiving Frames 49
4.3 Decoder 49
4.4 Synthesizer 50
4.5 Display 51
4.6 User Interface and Internet Interface 55

5 Results and Discussion **60**
5.1 Encoding Results 61
5.2 Decoding Results 62

6 Conclusions and Future Work 67

Bibliography 70

A Biography 73

List of Tables

2.1 Comparison of Several Lossless Compression Algorithms 14
2.2 A JPEG Quantization Table 19

5.1 Play Speed Comparison 63
5.2 Decomposition of the VOD Playback Time 65

List of Figures

2.1 Halftoning and Dithering 10
2.2 4x4 ordered dithering matrix generated from a 2x2 ordered dithering matrix 12
2.3 Diagram of Data Compression 13
2.4 Diagram of JPEG 19
2.5 A VTC Scheme 28
2.6 TCP/IP Architecture and Addressing 35

3.1 the VOD System Architecture 40
3.2 Structure of The Server 40
3.3 A Frame In Transform Domain with $L = 3$ 41
3.4 Structure of The Client 43

4.1 VV Video Clipfile Format 48
4.2 Data Structure for Bitstream Decoding 50
4.3 Subbands Organization in the Bitstream 51
4.4 Synthesis Procedures 52
4.5 Color Conversion and Dithering in the Display Module 54
4.6 How to play the video 55
4.7 Implementation Scheme of One-Button Video-On-Demand Using CGIs 57

5.1 Comparison of the VOD System and A MPEG-1 Encoder: Complexity vs. Bit Rate 61

5.2 Comparison between the VTC and MPEG-1: Average Peak SNR vs. Bit Rate 63
5.3 Comparison between the VTC and MPEG-1: Playback Time vs. Bit Rate 65
5.4 Comparison between the VTC and MPEG-1: Playback Speed vs. Bit Rate 66

Abstract

A novel video-on-demand (VOD) system over a TCP/IP computer network, using advanced Vector Transform Coding (VTC) video compression technology was developed through the master thesis study. It demonstrates the applicability of using a new video compression technology called VTC in applications of information-on-demand, such as video-on-demand, video conferencing etc. The testing results of the prototype VOD system shows that the compression performance exceeds that of the MPEG standard. Furthermore, a new VOD framework independent of compression technology was developed, which to some extent proves that there is no need for a dedicated video compression standard for computer based applications.

The performance of this prototype VOD system was tested and analyzed. The results indicate that VTC technology is suitable for VOD system because of better compression performance at low bit rate and faster play back.

Chapter 1

Introduction

An information-on-demand industry is emerging which requires the rapid search, sort, storage, transfer and display of data and images. New technology will provide customized news, weather, sports and entertainment programming as well as educational, financial, medical and employment information. The delivery of large amount of timely information over vast distances requires a storage density and flexibility that only digital systems can provide. These requirements are driving improvements in automated digital processes and in new compression software appropriate for image, text, sound and motion.

According to a report in Business Week, Feb. 14, 1994, "The biggest obstacle to the vaunted multimedia revolution is digital obesity. That's the bloat that occurs when pictures, sound and video are converted from their natural analog form into computer language for manipulation or transmission. ... Compression, a rapidly developing branch of mathematics, is putting digital on a diet. ... Its popularity is rooted in economics: compression lowers the cost of storage and transmission by packing data into a smaller space. Many new electronic products and services simply couldn't exist without it."

Data compression, especially image and video compression therefore became one of the hottest areas both in academy and industry today. To enable the interoperability among different sections of industries, different compression standards have been established. For example, the JPEG for still image compression, H.261 for

video conferencing, the MPEG for motion picture compression. Many new algorithms and techniques, such as subband coding, wavelet, fractal compression ([1], [2]) and model-based compression [3], are continually being developed and proposed to exceed the existing standards. Vector Transform Coding algorithm [4],[5] is one of these newly developed algorithms.

However, since the standardization process is very slow, and thus newly developed better compression techniques normally could not be applied in practice promptly due to the vast different stream formats and techniques used, which hinder the exchange of information. In order to accelerate the application of these new and better compression techniques in real world, it is an urgent need to find a generic framework which can accommodate different image and video compression formats to make them readily interchangeable.

While evaluating the VTC technology for VOD applications, we developed a generic framework independent of compression technology in the prototyped VOD system. The key of this generic VOD framework requires software only video playback. As we will see later, the highly asymmetric feature of VTC technology allows very simple operations in the video playback. However, unlike other compression technologies, the simplification on video playback does not sacrifice video quality much. This comes from the inherent advantage of VTC technology. VTC preserves high intra-vector-correlation and reduces inter-vector-correlation of video data.

The finished prototype one-button-VOD system runs on any networks that support TCP/IP protocols with X window. Using the widely used World Wide Web browser which supports the HyperText Transport Protocol (HTTP) as the user interface, a user can virtually click one button at his/her home-page and be able to watch a video clip no matter what machine and what compression technology he/she is using.

From the testing results, we proved the following two points: first, the VTC technology is applicable to such applications as video-on-demand with higher compression quality and faster playback than MPEG standard; second, it is not always necessary to have computer-based video compression standards.

The rest of the thesis is organized as the following: Chapter 2 will introduce

the necessary background for the design and implementation of our VOD system. Elaboration of the VTC technology which is the core of the system is also in this chapter. Chapter 3 then will present the architecture of the system. Chapter 4 will discuss the key implementation features of the system. Chapter 5 will give the performance analysis of the compression technology using the data collected from the test runs of the developed system. Chapter 6 summarizes the thesis, and covers some very important features of VOD systems but are not implemented in the system.

Chapter 2

Background

2.1 An Image Model

2.1.1 Introduction

The pictures we see on TV or in the theater, which are captured by cameras, are not interpretable by computers. They have to be digitized into digital forms. Thus a black-and-white image to us (human eyes) becomes simply a two-dimensional array of integers to the computer, with the values of the integers proportional to the brightness. Array elements are called pixels. All the image and video compression and processing we talk about in this thesis refer to this digital image model.

A video or an image sequence consists of a set of frames. Each frame is what we call a *still image* which corresponds to the scene captured at one single moment. A natural question then is how the color is represented in a video.

First, let's see how the color is perceived by the eye. Visible light is an electromagnetic wave in the 400nm - 700 nm range. Most light we see is not one single wavelength, it's a combination of many wavelengths instead. The eye is basically just a camera. Each neuron is either a *rod* or a *cone*. The rods and cones serve the same function as a CCD array in a camera. Rods are sensitive to the light but not colors. Cones are sensitive to different wavelength. Since cones come in 3 flavors: red, green and blue, each responds differently to various frequencies of light, the

color signal to the brain comes from the response of the 3 cones to the spectra being observed. Perceptually, we need three attributes to describe a color. Consequently, to represent a color signal, we need three integers to represent it in computer.

If we imagine that each of the three attributes used to describe a color are axes in a three dimensional space, then this defines a color space. According to Grassman's laws, for a color space the following properties (A,B,C,etc. are colors) hold:

1. If $A = B$ *and* $C = D$ *then* $A + C = B + D$
2. $If A = B + C$, *then* $B = A - C$
3. $If A = B$ *and* $B = C$, *then* $A = C$

The immediate question is that given a basis for the color space (three independent lights) how can we select coefficients so that a color is reproduced? There is not always an answer for such a question. Some colors cannot be exactly reproduced with a combination of the given three primaries. The primaries simply don't span the whole color space. In 1931, the Commission Internationale de l'Eclairage (CIE) defined a set of primaries that span the space with only positive coefficients, but the primaries are no longer pure colors. The colors that we can perceive can be represented by the CIE system. Other color spaces are subsets of this perceptual space. For instance, RGB color space, as used by television displays, can be visualized as a cube with red, green and blue axes. This cube lies within our perceptual space, since the RGB space is smaller and represents less colors than we can see.

So, a color space is a mathematical representation of our perceptions. For example, RGB is a 3-dimensional space with red, green and blue axes. It's useful to think so because computers are fond of numbers and equations. There are other color spaces for different purposes. In the next subsection, some of the common color spaces will be discussed a little further, more references can be found in [6], [7].

Now back to the question that how a color video is represented, once an appropriate color space is chosen, each pixel in a frame of a color video is simply represented by the three primaries of that specific color space. Still using RGB

space as an example, a frame is then represented by an array of (R,G,B) triples, with each (R,G,B) triple corresponding to each pixel in a frame.

2.1.2 Major Color Spaces

According to Bourgin [7], the major color spaces can be summarized as:

RGB (Red Green Blue)

This is an additive color system based on trichromatic theory, commonly used by CRT displays where proportions of excitation of red, green and blue emitting phosphors produce colors when visually fused. It is easy to implement, but non-linear, device dependent, un-intuitive, and commonly used in television cameras and computer graphics etc. .

CMY(K) (Cyan Magenta Yellow (Black))

CMY is a subtractive color space. It is mainly used in printing and photography. Printers often include the fourth component, black ink, to improve the color gamut (by increasing the density range), improving blacks, saving money and speeding up drying (less ink to dry). It is fairly easy to implement, but difficult to transfer properly from RGB, device dependent, also non-linear and un-intuitive.

HSL (Hue Saturation and Lightness)

This represents a wealth of similar color spaces, alternatives include HSI (intensity,hue), HCI (chrome/colorfulness), HVC, TSD (hue, saturation and darkness) etc. All these color spaces are linear transforms from RGB, they are thus, device dependent, non-linear but very intuitive. In addition the separation of the luminance component has advantages in image processing and other applications.

YIQ, YUV, YCbCr, YCC (Luminance - Chrominance)

These are the television transmission color spaces (YIQ and YUV analogue NTSC and PAL and YCbCr digital). They separate luminance from chrominance (lightness from color) and are useful in compression and image processing applications.As mentioned before, human eyes have different sensibility to lightness and colors. YIQ and YUV are, if used according to their relative specifications, linear. They are all device dependent and un-intuitive. Kodaks PhotoCD system uses a type

of YCC color space, PhotoYCC, which is a device calibrated color space.Empirical evidence suggests that distances in color spaces such as YUV, or YIQ correspond to perceptual color differences more closely than do distances in RGB space. These color spaces may give better results when color reducing an image.

SML (Short Medium Long)

A perceptual color space based on the response functions of the cones in the retina of the eye. It is good for psychometric research.

CIE

The HVS based color specification system. There are two CIE based color spaces, CIELuv and CIELab. They are near linear (as close as any color space is expected to sensibly get), device independent, but not very intuitive to use. CIELuv has an associated chromaticity diagram, a two dimensional chart which makes additive color mixing very easy to visualize, hence CIELuv is widely used in additive color applications, like television. CIELab has no associated two dimensional chromaticity diagram and no correlate of saturation so only Lhc can be used.

Since there is such a wide variety of color spaces, we need to know how to convert between them.

Many of the conversions are based on linear matrix transforms. The one commonly used in image and video compression is the conversion between RGB and YUV given as follows:

$$\begin{bmatrix} \mathbf{Y} \\ \mathbf{U} \\ \mathbf{V} \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \times \begin{bmatrix} \mathbf{R} \\ \mathbf{G} \\ \mathbf{B} \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{R} \\ \mathbf{G} \\ \mathbf{B} \end{bmatrix} = \begin{bmatrix} 1.000 & 0.000 & 1.140 \\ 1.000 & -0.396 & -0.581 \\ 1.000 & 2.029 & 0.000 \end{bmatrix} \times \begin{bmatrix} \mathbf{Y} \\ \mathbf{U} \\ \mathbf{V} \end{bmatrix}$$

These two matrices are the inversion of each other. More about conversions between other color spaces and more references can be found in [7].

2.1.3 Color Quantization and Dithering

Usually, when digitized, 8 bit or more depending on the application is used for each attribute per pixel of an image. For instance, 24 bit is used for each pixel in a full-color image where 8 bits are used for each color per pixel. Display devices are different. Typical display hardwares support 8 or fewer bits per pixel, so it can only display 256 or fewer distinct colors at a time. To display a full-color image, the computer must choose an appropriate set of representative colors and map the image into these colors. This process is called *color quantization* or *color reduction.* Color quantization is one of the most frequently used operations in computer graphics and image processing. Even though 24 bit graphics hardware is becoming more common, color quantization still maintains its practical value. It lessens space requirements for storage of image data and reduces transmission bandwidth requirements in multimedia applications.

There are a number of color quantization algorithms available, such as *Median Cut, Variance-based method, Octree, Kohonen Neural Network quantization and Local K-means* ([8], [9],[10],[11],[12]). The goal is to minimize the perceived difference between the original and the quantized images. Among them, *Median Cut* and *Octree* are the two most often used. Basically, the *Median Cut* tries to cut the color space (cube) of original image into two smaller cubes according to the histogram median color, it then recursively cuts these smaller cubes using the same technique until the total number of small cubes reach the desired number of colors. Then the cubes are sorted according to the sizes (determined by the number of colors in each small cube), then the available colors are allocated according to the order of the cube sizes, the largest cube first, the smallest cube last. In contrast to Median Cut, which divides the space to reach the desired number of colors, the *Octree* algorithm decreases the number of divisions of the original color space by merging until the total number of divisions is below the required. Briefly, the basic algorithm operates in three phases: **Classification, Reduction** and **Assignment. Classification** builds a color description tree for the image, ideally, each color in the original image will correspond to a leave of the tree. But in practice, a fully-formed

Figure 2.1: Halftoning and Dithering

color description tree consumes too much space, so the algorithm instead initializes data structures for nodes only as they are needed. **Reduction** collapses the tree based on certain criteria until the number it represents, at most, is the number of colors desired in the output image. **Assignment** defines the output image's color map and sets each pixel's color by reclassification in the reduced tree. *Octree* is a very fast algorithm that produces good quality palettes.

Color dithering is a software imaging process for arranging adjacent pixels of varying shades in order to achieve a visual effect. The process often enhances a computer's ability to display an image, particularly useful when the color or resolution of the original image must reproduced on computers with different display capabilities. However, the history of dithering technique can be traced way back in the printing business where more than two gray levels can be generated by using only black ink on white paper [6]. Nowadays, since dithering is used together with color reduction most of the time, people tend to use the term dithering and color reduction interchangeably. But the difference is clearly there, for we can perform color reduction without using dithering, except the display result won't be as good as using dithering technique.

The basic strategy of dithering is to trade intensity resolution for spatial resolution by averaging the intensities of several neighboring pixels. This averaging is usually done automatically by human eyes. A simple example will make this more clear (refer to Figure 2.1). By using 2×2 black and white pixels, 5 gray levels can be achieved, or more precisely can be perceived by human eyes.

The same idea extends to color dithering. It is similar to the process used by magazines to print full-color photographs with a base of only four colors. To replicate a color that is not in the palette of 256, alternating pixels in the frame buffer are set to either one of the two or more nearest available colors. This creates a faint and very fine checkerboard pattern that, when viewed from a distance, appears to

be the desired color. The success of this technique depends on the fact that the pixels are imperceptible at normal viewing distances. For color mapped devices, two most often used dithering algorithms are: error diffusion (sometimes called Floyd-Steinberg) and ordered dither.

In error diffusion dithering, each image pixel is mapped through color reduction to the closest pixel in a color map, whose entries are the colors supported by the display. The differences between the original triple and that in the color map is called *error*. This error is distributed to neighboring pixels. For example, half the error might be added to the pixel below and half to the pixel to the right of the current pixel. The next pixel (modified) is then processed. Processing is often done in a serpentine scan order: odd number rows are processed left to right and even number rows are processed right to left. Images which suffer from severe contouring when reducing colors can be improved with this option.

Ordered dithering is a form of threshold dithering. In black and white threshold dithering, any pixel below a certain threshold of luminance is mapped to black, and all other values are mapped to white. Ordered dithering uses the pixel's (x,y) coordinate in the image to determine the threshold value. An $N \times N$ dithering matrix D determines the threshold: $D[x_{modN}, y_{modN}]$ is the threshold at position (x,y). The matrix is chosen so that over any $N \times N$ region of the image with the same pixel value, the mean of dithered pixels in the region is equal to the original pixel value. When this scheme is extended to color dithering, dithering matrix is again used. The values in the matrix is not threshold values, but more like "error values". By adding these "error values" to each color attribute of a image before color quantization, a "smear effect" is applied to the image so as to reduce the visual artifact caused by color quantization. A typical 4×4 dithering matrix is used in the display module of our VOD system. It can be generated by a 2×2 dithering matrix, which is shown in Fig. 2.2.

The difference between error diffusion and ordered dithering is that in the error diffusion algorithm, the error values are calculated differences between the original image and color quantized image, while in ordered dithering, the values and the pattern in which the threshold values are applied to the original image are pure

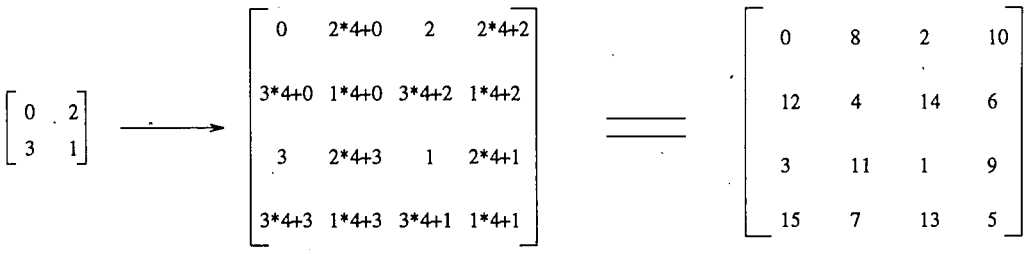

Figure 2.2: 4x4 ordered dithering matrix generated from a 2x2 ordered dithering matrix

empirical. Depending on the applications, sometimes error diffusion is better than the ordered dithering. Some times it is the other way around.

Even to date, color quantization together with color dithering are still black art. They are indispensable in computer graphics and image processing. And they are in fact one kind of lossy image compression. If not used correctly, or the color quantization is not done right, the error brought in by these procedure are more serious than the errors by the real lossy image compression algorithms.

2.2 Image and Video Compression Overview

Even though the cost of bandwidth of communication and storage of data are decreasing rapidly, there still exists strong compelling requirement for data compression. This is because the information is exploding at a higher rate and the demand for the transmission and storing of information by people is increasing at an unprecedent rate. Now with more and more PC going into ordinary people's families, the insatiable desire to work at home, to navigate the Internet, to get digital bits into the house have put forward a challenging task for data compression engineers to come up with new data compression techniques. For example, people can now get digital newspaper down on their PC everyday, it's very prompt and environmentally friendly (100% recycle rate), the only problem is that there are not many pictures come with it as in real news papers. The reason is simply that to transmit pictures would require a lot more bandwidth (or time), and it would sap up most of the PC memories. Since video is the most intuitive and powerful form to represent

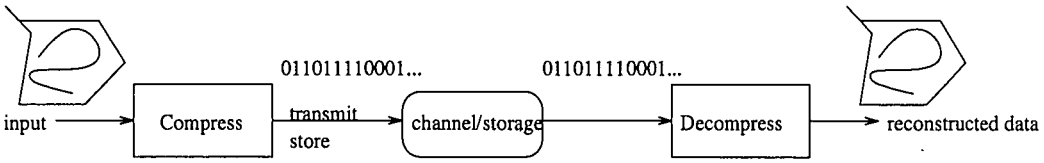

Figure 2.3: Diagram of Data Compression

information, more and more video clips are used in interactive multimedia information system to enhance education and entertainment, etc. With such explosion of video information, we have to find a better way to represent massive computer data such as image and video. For the transmission part, it could be partially solved by getting an ISDN or maybe Fiber optical line (cost is another issue), but if a person wants to store those realistic pictures he or she has to purchase another hard disk. Furthermore, what if he or she wants to watch a movie on the PC ?

With good data compression (see figure 2.3), this may soon become possible. For instance, even at a compression ratio of 3:1, we can get 3 pictures as opposed to just one single picture with no extra cost, i.e., you don't need to change your phone line, or buy extra hard disks. Imagine what high performance compression techniques can give us.

There are two types of data compression: lossless data compression and lossy data compression. Lossless data compression is the type of data compression where the reconstructed data is exactly the same as the original. Nothing is lost during the compression and decompression procedure. But the compression ratio can not be achieved very high, 3:1 is in general considered very good. Table2.1 gives a comparison among some lossless compression programs.[13].

On the other hand, lossy data compression exploring the characteristics of human perception in addition to the inherent data redundancy can obtain a much higher compression ratio at cost of data loss which is usually imperceptible to human. So, after a lossy data compression, the reconstructed data is no longer the same as the original data. However the difference is being made so small that in fact people usually can hardly notice the difference. Considering the large gain in compression ratio (in the range of 20:1 to hundreds to 1), the small loss is considered very worthwhile. Hence lossy compression techniques are usually more preferable in practice, especially in image and video compression where the tremendous data

compression program	compression ratio			
	Lena	football	F-18	flowers
lossless JPEG	1.45	1.54	2.29	1.26
optimal lossless JPEG	1.49	1.67	2.71	1.33
compress (LZW)	0.86	1.24	2.21	0.87
gzip (Lempel-Ziv)	1.08	1.36	3.10	1.05
gzip -9 (optimal Lempel-Ziv)	1.08	1.36	3.13	1.05
pack (Huffman coding)	1.02	1.12	1.19	1.00

Table 2.1: Comparison of Several Lossless Compression Algorithms

volume really needs some help.

The reason that natural image and video can be highly compressed with little quality loss is that the pixel values in an image or video frame are highly correlated. For images, the neighboring pixels are highly correlated. For videos, beside the intra-frame correlation inside each frame, the pixels in the consecutive frames are also correlated. These correlations are the redundancy which can be removed from the original images to achieve compression according to Shannon information theory. Image and video compression is to reduce these correlation (redundancy) as much as possible so that these images and videos can be represented by using as less bits as possible (quantization).

Quantization in data compression is where loss comes from. In a nut shell, the goal of quantization is to throw out bits. A more strict definition of quantization can be given as: the conversion of a continuous-amplitude signal into one of a set of discrete amplitudes, thus resulting in a discrete-amplitude signal that is different from the continuous-amplitude signal by the quantization error or noise. In image and video compression, the objective is to use lowest possible number of bits to represent the original images for a given distortion.

When each of a set of parameters (or a sequence of signal values) is quantized separately, the process is known as scalar quantization. When the set of parameters is quantized jointly as a single vector, the process is known as vector quantization [14]. We shall often abbreviate vector quantization in this paper as VQ. In the rest

of this chapter, we'll discuss briefly commonly used transform coding and two forms of quantization. For quantization, we'll discuss vector quantization as a more advantageous lossy data compression technique, its performance as opposed to scalar quantization. To fully exploit the advantage of VQ, a new coding scheme – Vector Transform Coding(VTC), which is used in our experimental VOD system, is then discussed. In the end, some current video compression standards are briefly described.

2.3 Transform Coding

The goal of transform coding is that by performing a suitable linear transformation on the input vector, X, we can obtain a new vector, Y, with the same number of components, often called transform coefficients or simply, coefficients, with the feature that these coefficients are much less correlated than the original samples. In addition, the information may be much more "compact" in the sense of being concentrated in only a few of the transform coefficients. Usually, the transform does not reduce the data nor does it lose any information about the original data, because an inverse transform can reproduce the original data if only transform is performed on the original data. In other words, all transform does is to prepare the data for optimized data quantization [15].

The most popular transform in terms of systems actually implemented and in established and proposed standards is the *discrete-cosine transform* (DCT). The DCT is an orthonormal transform having some of the features of a transformation to the frequency domain. It is equivalent to a discrete Fourier transform (DFT) of a symmetricized extension of the input set of samples and the fast algorithms are available for efficiently computing the DCT. The DCT has an advantage over the DFT in that it is a purely real transform if the input vector is real. A two-dimensional DCT can be given as:

$$X(k_1,k_2) = u(k_1,k_2)\sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x(n_1,n_2)\, cos(\frac{\pi(2n_1+1)k_1}{2N})\, cos(\frac{\pi(2n_2+1)k_2}{2N})$$

$$x(n_1,n_2) = \sum_{k_1=0}^{N-1}\sum_{k_2=0}^{N-1} u(k_1,k_2)(k_1,k_2) cos(\frac{\pi(2n_1+1)k_1}{2N}) cos(\frac{\pi(2n_2+1)k_2}{2N})$$

$$u(k_1, k_2) = \begin{cases} 1/N & if k_1 = k_2 \\ \frac{\sqrt{2}}{N} & if k_1 = 0 \text{ and } k_2 \neq 0 \text{ or } k_2 = 0 \text{ and } k_1 \neq 0 \\ 2/N & if k_1 \neq 0 \text{ and } k_2 \neq 0 \end{cases} \quad (2.1)$$

Karhunen-Loeve transform (KLT) is the best possible transform for minimizing the overall distortion D_{tc} for a given bit allocation under the assumptions that the input variables are Gaussian, and 1st order Markov model is used, that the optimal mean squared error quantization of the transform coefficients is performed, and that the optimal allocation result holds. However, it has the following disadvantages:

1) Actual image may not be 1st order Markov Gaussian;

2) Even if so, the correlation coefficient is different from application to application, thus different transform is required;

3) It is difficult to find fast algorithm.

On the other hand, it has been frequently reported that the DCT achieves performance very close to that of the optimal KLT and yet has the advantage that it is a fixed, signal-independent transform matrix.

Transform coding belongs to a larger family of coding techniques where the signal is decomposed or analyzed into component that in some sense offer a more fundamental or more primitive representation of the signal. Such coding schemes are called *analysis-synthesis coding systems.* The components most commonly represent some form of spectral decomposition of the signal as with the DCT. An important and widely used type of analysis-synthesis coding system is sub-band coding. In Subband coding, a bank of filters operates on the input signal to generate a set of narrow band cases each representing a different sub-band of the input spectrum. The narrow bandwidth of each sub-band signal allows subsampling to be performed, reducing the bit-rate needed to code each sub-band. Interpolation is then used to synthesize the reproduction of the original signal.

2.4 Scalar Quantization

An N-point scalar (one-dimensional) quantizer Q is a mapping $Q : R- > C$, where R is the real line, and $C = \{y_1, y_2, ..., y_N\}$ is the output set or *codebook* with size $|C| = N$.

For a quantizer with a finite output set, it is assumed that the indexing of output values is chosen so that $y_1 < y_2 < ... < y_N$.

Every quantizer can be viewed as the combined effect of two successive operations (mappings), an encoder E and a decoder D.

The purpose of quantization is to provide a limited-precision description of a previously unknown input value.

The most often used distortion measures between two numbers of a quantizer are:

$$d(x, x') = |x - x'|^2$$

or

$$d(x, x') = |x - x'|$$

where x is the input sample, and x'=Q(x) is the reproduced output value of the quantizer (or decoder).

They both are the special cases of:

$$d(x, x') = |x - x'|^m$$

where m is a positive integer.

The statistical average of the distortion is usually a more informative and meaningful performance measure.

$$D = E[d(X, Q(X))] = \int_{-\infty}^{\infty} d(x, Q(x)) f_X(x) dx.$$

where $f_X(x)$ is the probability distribution function of X. This is also referred to as *average distortion.*

Given a signal X and a scalar quantizer Q, the inevitable error $e = Q(X) - X$ that arises in the quantization of an analog signal is often regarded as "noise" introduced

by the quantizer. Specifically, *granular* noise is the component of the quantization error that is due to the granular character of the quantizer for an input that lies within the bounded cells character of the quantizer. The *overload* noise is that quantization error that is introduced when the input lies in an overload region of the partition , that is in any unbounded cell[16].

The performance of a quantizer is often specified in terms of a signal-to-noise ratio (SNR) or signal-to-quantization-noise ratio (sometimes denoted SQNR or SQR) defined by the following equation:

$$SNR = 10log_{10}\frac{E(X^2)}{D}$$

The goal of optimal quantizer design is to find the output points y_i and partition cells R_i that minimize

$$D = \sum_{i=1}^{N} \int_{R_i} (x - y_i)^2 f_X(x) dx$$

where $f_X(x)$ is the pdf of the random variable X, given N is fixed.

For a given decoder, i.e., y_i is fixed, the *optimal encoder* rule is *nearest neighbor.*

For a given encoder, i.e., R_i is fixed, the *optimal decoder* rule is the *centroid condition.*

As an example for scalar quantization, let's take a look at how the scalar quantization is done in JPEG. Figure 2.4 shows a diagram of the basic modules of a lossy JPEG.

The quantization module uses a quantization table something like Table 2.2 for a 8x8 block. For each 8x8 block out of DCT module, an element in the block is divided by the factor in the corresponding position of the quantization table, for example, element (0,0) would be divided by factor $q_{0,0} = 16$, and element (1,5) by factor $q_{1,5} = 19$. The result is then round up and stored in the same position. It can be easily seen that the dynamic range of the quantized block is much smaller than that of the block before quantization, thus it requires less number of bits to represent this quantized block than directly represent the original one. Therefore a certain amount of compression has been achieved. At the decoding end, this quantized block is reconstructed by multiplying the corresponding factors in the

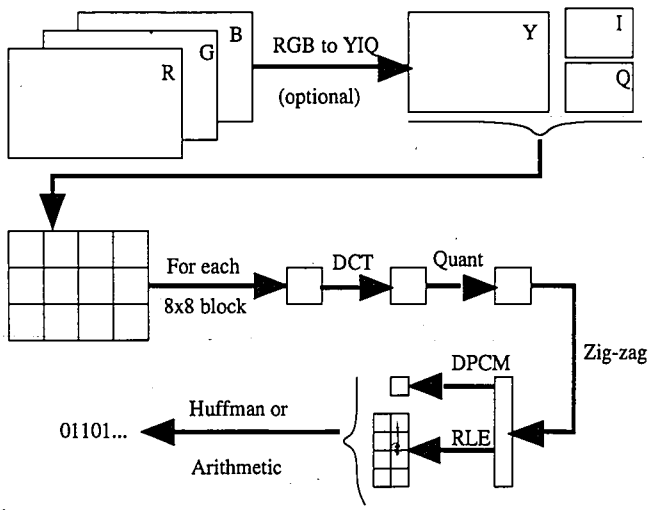

Figure 2.4: Diagram of JPEG

same quantization table. It is very clear that the compression comes at the cost of data fidelity. Suppose using the quantization table given in table 2.2, element (0,0) in block A is 85, after divided by 16 and then round up, it is stored as 5, at the decoding end, it is reconstructed as $5 \times 16 = 80 (\neq 85)$. Only those elements that are multiples of their quantization factors can be exactly reconstructed. By adjusting the values of the quantization factors, different compression ratio and compression quality can be achieved.

Compared to using a single quantization factor for the whole 8x8 block, the

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

Table 2.2: A JPEG Quantization Table

quantization table can usually achieve better compression performance. Because it takes advantage of human visual characteristics. Human eyes have different sensibility to different frequencies. Eye is more sensitive to low frequencies (upper left corner in the 8x8 block), less sensitive to high frequencies (lower right corner in the 8x8 block), so, by using smaller factors at the upper left corner, and larger factors at the lower right corner, we keep more low frequency transitions than high frequency ones, eyes can hardly tell anyway. Thus we can achieve higher compression ratio without losing perceptual quality (Mean Square Error might disagree, it must be replaced by a better distortion criterion).

Scalar quantization (SQ) has been used in other image compression standards. For example, SQ with feedback is used in H.261, MPEG-1 and MPEG-2. Except that JPEG uses separate quantization table for each color component while in MPEG Luminance and chrominance share quantization tables. More about these video compression standards are discussed in the following section 2.7.

2.5 Vector Quantization

2.5.1 Vector Quantization Problem

Vector quantization (VQ) is a generalization of scalar quantization to the quantization of a vector, an ordered set of real numbers. While scalar quantization is used primarily for analog-to-digital conversion, VQ is used with sophisticated digital signal processing, where in most cases the input signal already has some form of digital representation and the desired output is a compressed version of the original signal. VQ is usually, but not exclusively, used for the purpose of data compression.

A vector can be used to describe almost any type of *pattern*. Vector quantization can be viewed as a form of pattern recognition where an input pattern is "approximated" by one of a predetermined set of standard patterns, or in other words, the input pattern is matched with one of a stored set of templates or codewords. VQ can also be viewed as a front end to a variety of complicated signal processing tasks, including classification and linear transforming. In such applications VQ can be

viewed as the subsequent computations, sometimes permitting complicated digital signal processing to be replaced by simple table lookups.

Three main results about VQ have been proved as:

- VQ is *always* better than scalar quantization, which is supported by Shannon's fundamental rate-distortion theory.
- Higher-correlation between vector components results in higher VQ performance.
- VQ is the best compression technique if complexity is not considered. Before we look into theoretical properties of vector quantization, we need to form a vector quantization problem in general first.

Assume that $\mathbf{x} = [x_1 x_2 \cdots x_N]^T$ is an N-dimensional vector whose components $\{x_k, 1 \leq k \leq N\}$ are real-valued, continuous-amplitude random variables. If the vector $\mathbf{x}$ is mapped onto another real-valued, discrete-amplitude, N-dimensional vector $\mathbf{y}$, we say that $\mathbf{x}$ is quantized as $\mathbf{y}$, and $\mathbf{y}$ is the quantized value of $\mathbf{x}$. $\mathbf{y}$ is also called reconstruction vector corresponding to $\mathbf{x}$. Typically, $\mathbf{y}$ takes on one of a finite set of values $\mathbf{Y} = \{\mathbf{y}_i, 1 \leq k \leq L\}$, where $\mathbf{y}_i = [y_1 y_2 \cdots y_N]^T$. The set $\mathbf{Y}$ is referred to as the *codebook*, L is the size of the codebook, and $\{\mathbf{y}_i\}$ are the set of *code vectors.*

When $\mathbf{x}$ is quantized as $\mathbf{y}$, a quantization error results, and a *distortion measure* $d(\mathbf{x}, \mathbf{y})$ can be defined between $\mathbf{x}$ and $\mathbf{y}$.

There are two major tasks in the design of a vector quantizer: the selection of the appropriate distortion measure and the design of the codebook (also known as the training of a codebook).

Either for purposes of transmission or for those of storage, each vector $\mathbf{y}_i$ is encoded into a codeword of binary digits (bits) c_i, of length B_i bits, in general, the different codewords will have different length, the average number of bits per parameter or per dimension is then given by

$$R = \frac{B}{N} bits/dimension$$

where

$$B = \frac{1}{M}\sum_{i=1}^{M} B_i$$

with M being the number of codewords transmitted or stored.

In designing a data compression system, one attempts to design quantizer such that the distortion in the output is minimized for a given bit rate.

The next two sections will discuss the two most important issues in the design of a vector quantizer separately.

2.5.2 Distortion Measures

To be useful, a distortion measure must be tractable and subjectively representative. The first requirement is because it can be computed in reasonable time so that people can analyze it and get to know whether the distortion is too big or not. The second criterion had been ignored until recently. As studies [14] show that in the case of audiovisual signal quantization, the most often used distortion measure does not always give a consistent indication of distortion with the perception of human being. For example, people have found that a few decibels of decrease in the distortion is quite perceivable by a person in one situation but not in another. As the search for quantitative perceptual measure distortion continues, the following traditional distortion measures are still used by people to evaluate the performance of a coding system.

1) **Mean-Square Error**: This is the counter part of MSE in scalar quantization. And is by far the most often used distortion measure. This is simply because it is easy to calculate and tractable, and to a large extent match with the perception of human beings. It can be easily obtained by expanding the definition for scalar quantization, as follows:

$$d_2(\mathbf{x},\mathbf{y}) = \frac{1}{N}(\mathbf{x}-\mathbf{y})^{\mathrm{T}}(\mathbf{x}-\mathbf{y}) = \frac{1}{N}\sum_{k=1}^{N}(x_k - y_k)^2$$

This is also the distortion measure we used in our VOD system.

2) **Weighted Mean-Square Error**: In the MSE d_2 we assumed that the distortions contributed by quantizing the different parameters $\{x_k\}$ were weighted equally.

In general, unequal weights can be introduced to render certain contributions to the distortion more important than others. A general weighted MSE is then defined by:

$$d_w(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})^{\mathbf{T}} \mathbf{W} (\mathbf{x} - \mathbf{y})$$

where $\mathbf{W}$ is a positive-definite weighting matrix.

3)Perceptually Motivated Distortion Measures: For very small distortions, and therefore high bit rates, most reasonable distortion measures, including those mentioned above , all exhibit similar behavior, by linearity arguments. Furthermore, they would all be expected to correlate well with subjective judgments of human visual or audio quality. However, as bit rate decreases and distortion increases, simple distortion measures have not always correlated well with perceptual judgments. Since VQ is expected to be especially useful at low bit rates, it becomes more important to develop and use distortion measures that are correlated better with human audio-visual behavior. A number of perceptually based subjective judgments, have been proposed and used. And efforts have been continued to devise fully quantitative perceptual distortion measures, which depends a lot on the fully understanding of the human perception behavior.

The principal goal in design of vector quantizers is to find a codebook, specifying the decoder, and a partition or encoding rule, specifying the encoder, that will maximize an overall measure of performance considering the entire sequence of vectors to be encoded over the lifetime of the quantizer.

2.5.3 Codebook Design

To design the L-level codebook, we partition N-dimensional space into L cells, which are $C_i, 1 \leq i \leq L$. Associate with each cell C_i a vector $\mathbf{y}_i$. A quantizer is said to be an optimal(minimum-distortion) quantizer if the distortion in either of the above distortion equations is minimized over all L-level quantizers.

There are two necessary conditions for optimality. The first condition is that the optimal quantizer is realized by using a minimum-distortion or nearest neighbor

selection rule.

$$q(\mathbf{x}) = \mathbf{y}_i, \quad iff \ d(\mathbf{x}, \mathbf{y}_i) \leq \ d(\mathbf{x}, \mathbf{y}_j), \ j \neq i, 1 \leq j \leq L. \tag{2.2}$$

That is, the quantizer chooses the code vector that results in the minimum distortion with respect to $\mathbf{x}$.

The second necessary condition for optimality is that each code vector $\mathbf{y}_i$ is chosen to minimize the average distortion in cell C_i. That is, $\mathbf{y}_i$ is the vector $\mathbf{y}$ which minimizes

$$D_i = \epsilon[d(\mathbf{x}, \mathbf{y})|\mathbf{x} \in C_i] = \int_{\mathbf{x} \in C_i} d(\mathbf{x}, \mathbf{y}) p(\mathbf{x}) d\mathbf{x}$$

Like in scalar quantization, such a vector is called the *centroid* of the cell C_i.

Computing the centroid for a particular region will depend on the definition of the distortion measure.

In practice, we are given a set of training vectors. $\mathbf{x}(n), 1 \leq n \leq M$. A subset M_i of those vectors will be in cell C_i. The average distortion D_i is minimized by

$$\mathbf{y}_i = \frac{1}{M_i} \sum_{\mathbf{x} \in C_i} \mathbf{x}(n)$$

Thus one method for codebook design is an iterative clustering algorithm known as Lloyd algorithm.It is one of the important quantizer design algorithms. It basically contains the following steps:

Step 1. *Initialization:* Set m = 0. Choose by an adequate method a set of initial code vectors $\mathbf{y}_i(0), 1 \leq i \leq L$.

Step 2. *Classification:* Classify the set of training vectors $\mathbf{x}(n), 1 \leq n \leq M$ into the clusters C_i by the nearest neighbor rule:

$$\mathbf{x} \in C_i(m), \quad iff \ d[\mathbf{x}, \mathbf{y}_i(m)] \leq d[\mathbf{x}, \mathbf{y}_j(m)], \ for \ all \ j \neq i.$$

Step 3. *Code Vector Updating:* $m \leftarrow m + 1$. Update the code vector of every cluster by computing the centroid of the training vectors in each cluster

$$\mathbf{y}_i(m) = centroid(C_i(m)), \ 1 \leq i \leq L$$

Step 4. *Termination Test:* If the decrease in the overall distortion $D(m)$ at iteration m relative to $D(m-1)$ is below a certain threshold, stop; otherwise go to step 2.

The above algorithm can be shown to converge to a local optimum. Furthermore, any such solution is, in general, not unique. Global optimality may be achieved approximately by initializing the code vectors to different values and repeating the above algorithm for several sets of initializations and then choosing the codebook that results in the minimum overall distortion.

2.5.4 Encoding

Having designed a codebook as described above, one can then use it to quantize each input vector $\mathbf{x}(n)$. The quantization is performed as in equation 2.2 by computing the distortion between $\mathbf{x}(n)$ and each of the code vectors, then choosing the code vector with the minimum distortion as the quantized value of $\mathbf{x}(n)$. This type of quantization is known as a *full search*, since all code vectors are tested for quantizing each input vector. For an L-level quantizer, the number of distortion computations needed to quantize a single input vector is L. While a distortion computation can be arbitrarily complex, we can assume here that each distortion computation requires a total of N multiply-adds. Therefore, the computational cost for quantizing each input vector is

$$\zeta = NL$$

If we encode each code vector into $B = RN = \log_2 L$ bits for transmission, then

$$\zeta = N2^{RN}$$

So to speak, the costs are exponential in the number of bits per vector. Computational and storage costs double for each increase of 1 bit in the rate. Thus computation cost grows exponentially with the number of dimensions and the number of bits per dimension. As tests shown that the complexity of a codebook with $L = 2^{16}$ is beyond the computational capability of most contemporary computers. To compensate this problem, a number of fast-search algorithms have been proposed,

which are designed to reduce the computations in the encoding at some reduction in performance. Such as binary search, whose cost is only linear with the number of bits.

2.5.5 Vector Transform Coding

For a long time, many scalar decorrelation techniques like the above discussed DCT have been developed and used to optimize scalar quantization. As we discussed before, it is well known that vector quantization is always better than scalar quantization. For this reason, people tried to replace scalar quantization with vector quantization after the scalar transform has been applied to the original data, hoping that better compression performance would be achieved. However, the results didn't agree with this theory. Recently, an interesting question has been proposed that if vector transform is used prior to vector quantization the total performance of compression should be better. Some very encouraging results have been achieved through this new VT and VQ joint optimization.[5],[15],[17],[18].

The reason that performing VQ after scalar transformation is not effective can be explained as following: after scalar transform, like DCT, the transform domain coefficients are highly decorrelated, so the vectors formed by these coefficients have very little intra-vector correlation. As we discussed earlier, VQ is more efficient when the vectors have more intra-vector correlation, thus, VQ is normally not efficient after scalar transform.

In order to effectively exploit both the decorrelation property of transform and the VQ advantage over highly intra-correlated vectors. We must find a way to preserve the correlation within vectors while decorrelate the correlation between different vectors. Vector Transform Coding proposed by Li [4],[5],[15],[17],[18] is one of such schemes that can achieve this goal.

In vector transform coding scheme, transform is no longer a scalar based transform, it is a vector-based transform. This transform is designed to keep the correlation within vectors and reduce the correlation between vectors. A simplified vector transform scheme is shown in Figure 2.5. The idea of this simplified vector

transform is to first subsample an image into several subimages, then, do scalar transform (a transform like DCT) in each subimages, then group the corresponding vector pixels in each of the transformed subimages as a vector, then do VQ for the vectors. We can see the intra-vector correlation is well preserved since there are no transform used to decorrelated its components. And the correlation between vectors is reduced, since we applied the transform between them. In [15], Li has shown that image compression using VTC technology is much better than JPEG which uses both ST and SQ. One goal of this thesis is to show that the VTC technology works well in video applications.

2.6 Entropy Coding

Entropy coding is a technique for encoding discrete data into variable length codewords in an invertible fashion.

Entropy codes are often used in conjunction with scalar quantizers (to conserve the average bit rate) and are often fairly simple to implement when the input alphabets are of reasonable size. The overall variable rate code is then a simple cascade of a scalar quantizer, which performs the analog-to-digital conversion in a fixed rate manner, and a variable length noiseless code, which maps the quantizer output into a variable length binary index in a way that can be perfectly decoded by the receiver. It can be shown that in the high rate case, coupling simple scalar uniform quantizers with noiseless variable rate codes of vectors can achieve performance within approximately 1/4 bit of the Shannon optimum as given by the Shannon rate distortion function.

The goal of noiseless coding is to reduce the average number of symbols sent while suffering no loss of fidelity. Huffman coding is one kind of entropy coding schemes. Even though it was developed by D. A. Huffman in 1952, it's implementation, especially for high throughput, is still a challenge today. Arithmetic coding and run length coding are other two most often used lossless compression methods.

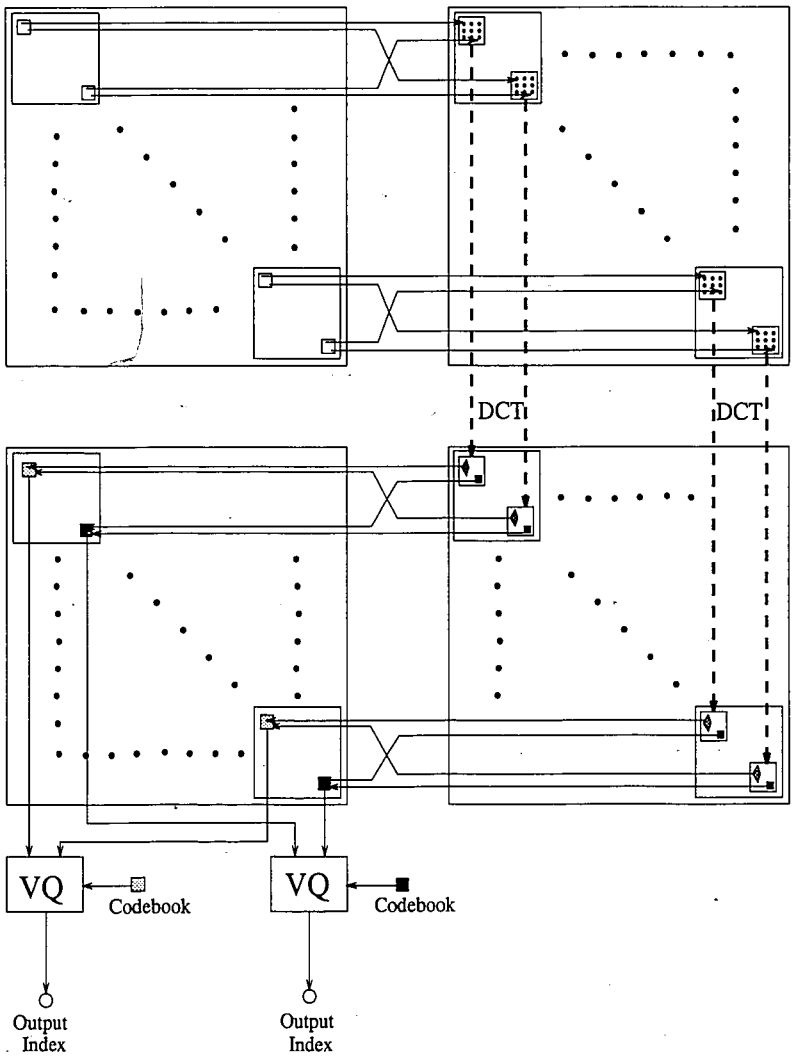

Figure 2.5: A VTC Scheme

2.7 Video Compression Standards

To facilitate world wide interchange of digitally encoded audiovisual data there is a demand for international standards for the coding methods and transmission formats. International standardization committees have been working on the specification of several compression algorithms, which are discussed briefly in the following sections [19].

2.7.1 JPEG

The ISO Joint Photographic Experts Group (JPEG) has developed a compression standard for continuous-tone still image applications such as graphic arts, color facsimile, and desktop publishing. The JPEG standard specifies both baseline and extended systems, which are intraframe coding schemes. In the baseline system, the input image is divided into disjoint 8x8 blocks. Two dimensional DCT is applied to each block, followed by a quantization to reduce the data dynamic range. The 2-D quantized DCT coefficients (8x8) are then zigzag scanned into a 1-D data sequence where the neighboring contiguous zeros are grouped together into a run length which can be coded more efficiently. The Huffman code is used to code the run length and nonzero coefficients. In the baseline system, the Huffman code information can be embedded in the bit stream and sent to the decoder for codebook generation. The JPEG standard specifies how the codebook is generated from the encoded information.

In addition to the baseline system, JPEG also specifies an extended system which offers more features (such as arithmetic coding and more precision in the pixel values) and additional coding structures (such as progressive transmission and lossless coding for various applications). The JPEG activity was started around 1987, and became an international standard in 1991.

2.7.2 H.261

In order to provide video phone and video conferencing services over ISDN, ITU-T (formerly CCITT) has completed and approved the H.261 standard in 1994. The H.261 standard specifies a real-time encoding-decoding system with a delay less than 150 ms. The algorithm is capable of running over a set of transmission rates of $p \times 64KB/s$ ($p = 1, 2, ..., 30$). The format for the input image is based on the Common Intermediate Format (CIF) which is 360 pixels by 288 lines for luminance and 180 pixels by 144 lines for chrominance. The frames are non-interlaced, and the input rate is 29.97 frames/second for NTSC-compatible systems. For video phone applications where low bit rates are required, another format, 1/4 CIF (QCIF) which is 180 pixels by 144 lines for the luminance and 90 pixels by 72 lines for chrominance signals, has also been defined in the standard [20],[21].

The coding scheme for the H.261 standard can be summarized as following. Each frame is divided into disjoint macroblocks, each of which consists of one 16x16 luminance (Y) block and two 8x8 chrominance (Cr and Cb) blocks. For each luminance block, we find from the previous frame a best-match block. The purpose here is to remove the temporal redundancy. This process is called *motion estimation.* For a typical video scene, it is highly likely that an object will sustain for some period. Motion estimation searches for a representation of the current macroblock from a previously coded picture. When a suitable representation is found, only information apart form that representation is needed to be coded, and compression is achieved. How to find the best-match block is not specified in the standard. Nevertheless, the most commonly used criterion is the minimization of the displaced block difference (DBD). A best-match motion vector (displacement) is obtained from this process.

The encoder then decides whether intra or predictive (inter-) mode should be used. In the predictive (inter-) mode, the motion-compensated predictive error, defined as the difference between the current block and the best-match block (the block in the previous frame displaced by the best-match motion vector), is coded. Otherwise, the current block is coded directly. If the best-match block data are not quantitatively close to the current block, the motion-compensated predictive

error will be large, and thus coding the current block directly is more advantageous. Note that in predictive mode, the best-match motion vector needs to be sent to the decoder for video reconstruction. In either intra- or predictive mode, a DCT followed by a quantizer is used to code the data. This DCT coding process is almost the same as that given in JPEG's standard. The quantizer step size is adjusted periodically to govern the resulting bit rate to the desired value. A possible method is to determine the quantizer step size directly form the buffer content.

2.7.3 MPEG-1

The ISO activity of the Moving Picture Experts Group (MPEG) was started in 1988 for CD-ROM applications at a bit-rate below 1.5 Mb/s. MPEG-1 was originally developed for storage of full-motion video. The recommended input picture size for the MPEG-1 standard is 360x240 pixels for luminance and 180x120 for chrominance., and the frame rate is 29.97 frames per second for NTSC-compatible systems, although these parameters can vary in the standard. The MPEG-1 coding scheme is very similar to that of ITU-T H.261. The major difference between the two is that MPEG-1 allows bi-directional motion compensation.

A video sequence is divided into groups of pictures (GOP's). There are three possible types of pictures in a GOP: I-picture, P-picture and B-picture. Coding of I- and P- pictures is similar to the scheme used for the JPEG and H.261 standards, i.e., intraframe and motion-compensated interframe techniques with DCT. Coding of B-pictures is slightly different from that of the P-pictures. In P-pictures, motion prediction is obtained from some previous frames only, i.e., forward motion compensation. However, in B-pictures, information from both previous and future frames has to be used for motion prediction, i.e., bi-directional motion compensation. For each macroblock in B-picture, we find the best-match blocks from the previous and the next I- or P-pictures. The best-matched block(s) from both can be used as the motion-compensated prediction. The choice is not specified in the standard. As in P-pictures, either the motion-compensated prediction error or the current block is coded with DCT followed by a scalar quantization. Since processing of B-pictures

requires the information in the previous and the next I- or P-pictures, the frame processing order cannot follow the natural sequence of frame numbers. Thus, given two B-pictures between two adjacent I- and P- pictures, after processing of Frame 0 (I-frame, intraframe coding) is completed, Frame 3 (P-frame, interframe coding) is processed, followed by frames 1 and 2 (B-frames), and so on. This process is repeated until the intraframe mode is triggered at the first frame of the next GOP, and another new cycle starts.

The MPEG-1 standard only specifies the bit stream syntax and decoding process. At a given bit rate, different standard-conforming encoding schemes are possible. It has been demonstrated that at 1.5 Mb/s, the video quality is comparable to the VCR's [22].

2.7.4 MPEG-2

After completing the MPEG-1 standard for compressed video targeted at storage applications around 1.5 Mb/s, the MPEG group continued to enact the MPEG-2 standard for broader and higher bit-rate applications including broadcasting, consumer electronics, and telecommunications. The MPEG-1 standard can provide a broad range of bit-rates due to its parameterization approach, although the standard was originally aimed at below 1.5 Mb/s for CD-ROM applications. However, its quality and flexibility are considered not ideal enough for the above-named applications at higher bit rates. To meet the higher quality and higher bit-rate objective, the MPEG-2 project was put together.

Roughly speaking, coding algorithms conforming to the MPEG-1 and MPEG-2 standards are similar. Nevertheless, the MPEG-2 standard accommodates more features than the MPEG-1 standard. Such features include interlaced video manipulation, scalability, compatibility, error resilience, and " hooks and options" for very high resolution video coding. One of the major differences between the coding schemes for the MPEG-1 and MPEG-2 standards is that the MPEG-2 standard is capable of handling interlaced video sequences adaptively with either frame or field modes, which the MPEG-1 standard can only deal with one fixed mode. Given an

interlaced video, the MPEG-1 encoder can only treat each field or each frame(by merging the adjacent two fields) as an integral unit, whereas the MPEG-2 encoder can in addition, exploit the correlation between the two fields in a frame and select the optimum mode. The MPEG-2 standard also provides more options for coding schemes, including choice of slanted scanning, linear or nonlinear quantization table, and the DC resolution for intra-macroblocks. Scalability can provide multiple-grade and multiple-bandwidth services. It can also be used for error resilience under error-prone environment. The MPEG-2 standard provides five types of scalability tools: data partitioning, SNR scalability, chrome simulcast, spatial scalability, and temporal scalability.

Since MPEG-2 is intended to be a generic standard to meet requirements of vastly different types of applications, ISO/MPEG has decided to associate the MPEG-2 standard with *Profiles* and *Levels*. A *Profile* uses a subset of the available syntactic elements to support a number of technical features and functionalities required by a cluster of similar application. The MPEG-2 standard has currently defined five Profiles:**Simple, Main, SNR, Spatial** and *High*. **Main Profile** aims to support the most common applications that are likely to be introduced first. **Simple Profile** allows lower memory and hardware complexity at the expense of fewer technical features and thus lower quality. The other three Profiles provide an environment for scalability and higher chrominance resolutions. *Levels* are defined by the range of parameters such as picture size, frame rate, bit rate, pixel rate, buffer size, etc.

Compared to H.261, the resulting quality from MPEG-1 and MPEG-2 is better due to the use of bidirectional prediction. However, they have a looser requirement on coding delay as is dictated by two-way applications. MPEG-2 coding strategy is also used by the Grand Alliance of HDTV standard working for the video compression subsystem.

2.8 About the Network Platform

2.8.1 What is TCP/IP

The platform for the system developed are UXIX based machines on a network which supports TCP/IP communication software. TCP/IP was developed by DOD to connect a number of different networks designed by different vendors into a network of networks (the "Internet"). TCP/IP data can be sent across a LAN, or any machine on any other network through gateways. TCP/IP forms the basis for one of the most widely used networking technologies in the world [23].

TCP/IP is actually a layered internet protocol suite. TCP and IP are two core protocols in the suite, so TCP/IP is usually used to refer the whole protocol family. In general, TCP/IP applications use 4 layers: an application protocol such as *mail*, a protocol such as TCP that provides services needed by many applications, IP, which provides the basic service of getting datagrams to their destination, the protocols needed to manage a specific physical medium, such as Ethernet or a point to point line. File transfer, remote login and computer mail, to name a few, are traditional TCP/IP applications (services) that run on top of TCP/IP.

TCP (the "transmission control protocol") is responsible for breaking up the message into datagrams (A datagram is a unit of data, which is what the protocols deal with), reassembling them at the other end, resending anything that gets lost, and putting things back in the right order. IP (the "internet protocol") is responsible for routing individual datagrams. TCP simply hands IP a datagram with a destination. IP doesn't know how this datagram relates to any datagram before it or after it.

TCP/IP programs generally conform to a client-server form of interaction. With client-server computing, an application running on one host issues requests for services, and those services are supplied by an application component running on another host on the internet. There are two types of server, iterative servers which process one client's request at a time, and concurrent servers which can process the requests of multiple clients concurrently.

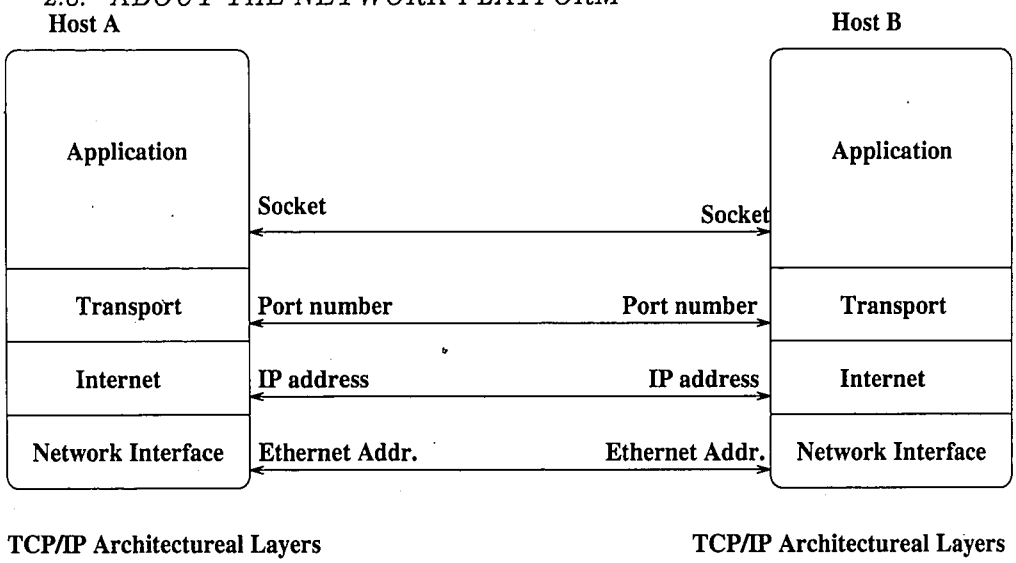

Figure 2.6: TCP/IP Architecture and Addressing

Each host in the internet is assigned at least one unique internet address. An internet address (IP address) uniquely identifies a particular point of attachment to the internet. Each application process running in a host is assigned a 16-bit TCP port number to differentiate it from other application process of the same Transport layer protocol in a host. However, for user applications, socket interface was developed to avoid dealing directly with the above two TCP/IP addressings. By using the socket interface, two application programs, one running in the local system and another running in the remote system, can communicate with one another in a standardized manner. The socket interface is the most widely used TCP/IP application programming interface today. Figure 2.6 shows the hierarchical addressing mechanism of TCP/IP network communication. So for any two processes to communicate, they have first to make a connection through the socket API system calls, which basically does the collection of the following information of the two processes:

Protocol-Identifier;

Local Internet Address;

Local Port Number;

Remote Internet Address;

Remote Port Number;

There are different types of sockets for applications of different purpose. The most common types are: stream sockets, datagram sockets and raw sockets. Stream sockets support a connection-oriented form of data transfer in which a stream of data can be reliably sent form one socket to another over a TCP connection. Datagram sockets support an unreliable datagram form of data transfer in which individual user datagrams can be sent from one socket to another. Raw sockets are sockets that provide access to the underlying IP and ICMP processes.

2.8.2 World Wide Web (WWW)

The World Wide Web is the universe of network-accessible information, an embodiment of human knowledge. It is an initiative started at CERN, now with many participants.

It has a body of software, and a set of protocols and conventions. WWW uses hypertext and multimedia techniques to make the web easy for anyone to roam, browse, and contribute to.

The World Wide Web software can pick up information from many information sources, using existing protocols. Among these are file and news transfer protocols, such as FTP, NNTP, Gopher and WWW's own protocol HTTP.

The Hypertext Transfer Protocol (HTTP) has been in use by the World Wide Web global information initiative since 1990. HTTP is an application-level protocol with the lightness and speed necessary for distributed, collaborative, hyper media information systems. It is a generic, stateless, object-oriented protocol which can be used for many tasks, such as name servers and distributed object management systems, through extension of its request methods (commands). A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred.

On the internet, the communication takes place over a TCP/IP connection. The protocol is basically stateless, a transaction consisting of:

- **Connection:** The establishment of a connection by the client to the server - when using TCP/IP, port 80 is the well-known port, but other non-reserved

ports may be specified in the URL (Universal Resource Locator, for specifications, refer to RFC822);

- **Request:** The sending, by the client, of a request message to the server;
- **Response:** The sending, by the server, of a response to the client;
- **Close:** The closing of the connection by either of both parties.

Currently, the most powerful HTTP client programs are the Mosaic and Netscape WWW browsers. The MOSAIC browser was developed by the Software Development Group (SDG) of the National Center for Super-computing Applications (NCSA). It offers a very nice and easy interface for people to browse the WWW on the Internet. It became so successful that a Netscape Communications Corporation was formed to develop and commercialize it under the name of Netscape. Because of its popularity, it became the de facto standard of browser. Based on this, we decided to use Netscape (Mosaic) as the interface to the internet of our VOD system.

One major reason that HTTP is very powerful is because of the support of the common gateway interface (CGI). The CGI is an interface for running external programs, or gateways, under an information server. What we refer to as gateways are really programs which handle information requests and return the appropriate document or generate a document on the fly. For instance, with CGI, an HTTP server can serve information which is not in a form readable by the client (such as SQL database), and act as a gateway between the two to produce something which clients can use. Gateway programs, or scripts, are executable programs which can be run by themselves. They have been made external programs in order to allow them to run under various (possibly very different) information servers interchangeably. Therefore, gateways can be used for a variety of purposes.

The video-on-demand applications is one of them that fits nicely in this WWW. The scheme we developed in the VOD system can be easily added on to the Internet. A user can visit this VOD site, and demand a video. The server sends over the player if the user site does not have a player that can play this compressed video. If however, the user site has a player already, the server simply starts sending the video sequence to the local player. A very important concept embedded in this scenario is

that the user does not need to have any knowledge about the compression technology implemented in the player, better yet, he does not need to worry about getting a player for the video he wants if he does not have one on his machine. This idea also extends to the video conferencing applications. In which case, the participants of a video conferencing are not limited to follow a video compression standard, instead they can use what ever compression technology they have one each one's machine, as long as they sent over the corresponding player during the initialization stage. In chapter 4, we'll discuss in detail how we used gateways to implement the compression technology independent video delivery feature of our VOD system.

Chapter 3

System Architecture

3.1 Overview

As we have shown in chapter 3, that the combination of vector transform and vector quantization should out perform VT and scalar DCT as used in the current compression standards. A simplified video-on-demand system is thus developed to test the overall compression performance of VT plus VQ. This VOD system employs vector subband coding and vector wavelet transform. It can provide basic VCR functions, such as forward, backward play and random positioning in the video sequence.

The system was first developed on a network of UNIX workstations in the X window environment, and later ported to networked PCs running MS Windows as well. Being an asymmetric system, the encoding of the video is done off line. The server only stores the compressed bit stream, and sends the bit stream over the network whenever demanded. The decoder decodes the bit stream and plays it back on the fly on the user's local machine. Figure 3.1 shows the system architecture.

3.2 Server

The server consists of three submodules, namely the fetching module, the processing module and the network interface. The fetching module locates the requested video

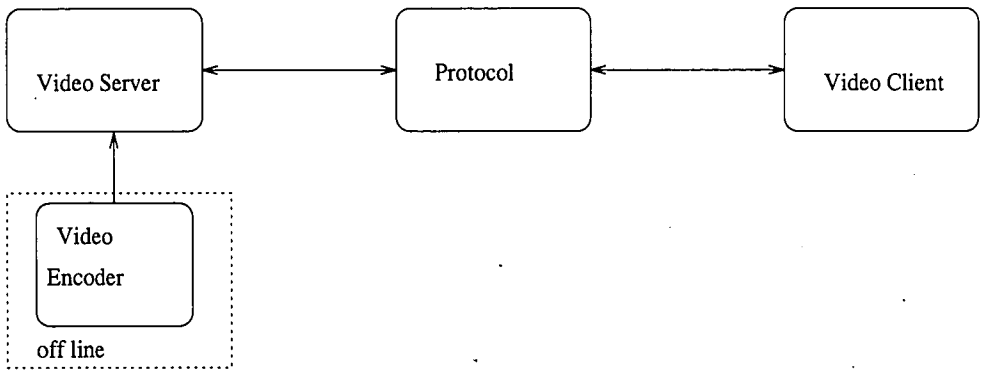

Figure 3.1: the VOD System Architecture

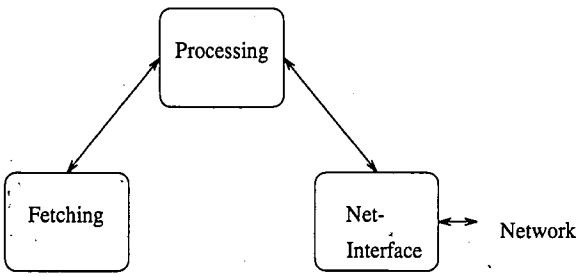

Figure 3.2: Structure of The Server

clip from the video archive, and loads it into the memory. The processing module processes the video clip just loaded in, and takes out the video bit stream to be sent out. This procedure can be dynamically adjusted based on updated user request. For example, sometimes, a user might want to skip a portion of the video. Once the server received this information, the processing procedure will do the skipping, and only sends out the necessary bit stream. The network interface will send out the packed bit stream, and communicates with the network interface on the client side. A diagram of the server is given in 3.2.

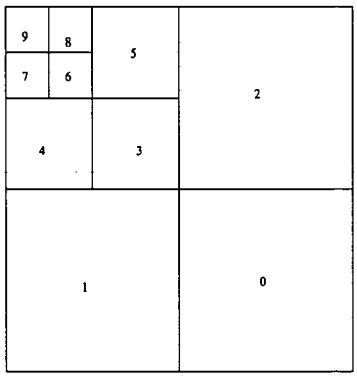

Figure 3.3: A Frame In Transform Domain with $L = 3$

3.2.1 Video Encoding

As for now, only intraframe coding is performed. Each frame is divided into blocks of size $N \times N$. The vector subband coding is performed separately for each of the three color components, thus vector subband filtering is performed three times for each block. As we have explained in chapter 2, human eyes are more sensitive to the low frequencies. Therefore, for an L-level subband coding, when going from ith level to the (i+1)st level, only the lowest subband is performed subband filtering. Therefore, each frame will look like Figure 3.3 in the transform domain after subband coding.

Next, vectors that include all three color information are formed in the transform domain. Thus the final vector size becomes tripled, $3 \times N \times N$. For example, the ith vector from Subband j would look like:

$$[x_{i,j,0}, y_{i,j,0}, z_{i,j,0}, x_{i,j,1}, y_{i,j,1}, z_{i,j,1}, \ldots x_{i,j,N-1}, y_{i,j,N-1}, z_{i,j,N-1}]$$

where x, y, z are the three color components.

Once all the vectors are formed, they are encoded by looking up the codebooks. Because the characteristics of each subband are different (some time vary different depending on the scenes of a video clip), each subband uses a different codebook. The codebook consists of codewords of different bit lengths. If the distortion measure does not satisfy the preselected criterion, the search in the codebook goes into a higher level, where more bits are allocated for each vector. Thus besides the index of a codeword, the classification number, i.e., the bit length is needed to enable

the decoding. Therefore, each vector is coded as a (class index) pair. The encoded frame thus is represented by a bit stream of these pairs. Notice that the information regarding the subband number is implicitly stored because the encoding is done in the increasing order of the subband numbers. The bit stream of a video clip is obtained by concatenating sequentially the bitstreams of all the frames in the clip.

3.2.2 Processing Module

The purpose of the processing module in the server is twofold. First, it takes the bitstream that needs to be sent out of the stored clip. Because some overhead like the location in the whole bitstream of each single frame is packed in the clip, which does not need to be sent over to the client. Second, it can process the user requests and take corresponding actions even after the sending of the clip has been started. One such example is that after watching a few frames, the user decides that this is the wrong clip and wants to stop it. The server should be able to stop loading the rest of the clip.

The processing module does its job by analyzing certain fields in the overhead as well as by analyzing the control messages received from the client by the network interface.

The processing module also performs queuing should there are more than one client requesting video clips. It does this by forking off a child process for each request.

3.2.3 Network Interface Module (NIM)

The major function of the network interface is of course the sending and receiving data to and from the network.

The data received by the NIM can be a name of a clip, the handshaking signal, or some special control requests. The data sent out is the bit stream of a frame and the size of it. This small overhead is to ensure the client receive the complete data.

To coordinate the various processing speed of the client machine and the dynamic traffic condition, sending buffer is used to buffer the bitstream. Handshaking is done

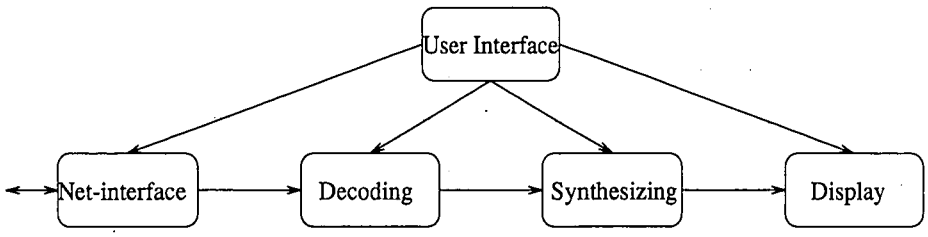

Figure 3.4: Structure of The Client

to ensure the synchronization of communication. Instead of flooding the bitstream out to the network, it waits for the ready signal, then sends the bitstream in the buffer out.

NIM interfaces with the network through a socket. Or we can say that the server provided its service at a specific socket. The connection between the server and a client is established at the initialization stage by the NIM module.

3.3 Client

Because this VOD system is asymmetric, client performs more tasks than the server does for individual video clips. These tasks can be casted as: network interfacing, decoding, synthesizing, displaying and user interfacing. See Figure 3.4 for the modules that perform these different tasks.

The user interface accepts a request from a user, it passes this information to the network interface. The NIM then sends this request to the server, and wait for the bitstream. Once the bitstream arrives, the NIM puts it into the frame buffer. It follows that the decoding module first decodes this bitstream into indices, then it looks up the codebook to get the vectors. These vectors are then used by the synthesize module to reconstruct a frame. The reconstructed frames are sent to the display module, which interfaces with the X server to playback the video. Because most workstations support only 8-bit color display, the display module also performs the 24 bit to 8 bit color truncation and dithering.

3.3.1 Network Interface Module

A counterpart of the server network interface, client NIM works very similarly. It first establishes the connection with the server by sending connect request to the socket where the server is listening. And then it sends out the request received by the user interface. Once a bitstream has arrived, it stores it in a frame buffer, and acknowledges the server with a short ready signal, then waits for another frame, if there is no overflow message from the decoding module.

3.3.2 Decoding Module

The decoding module does two kinds of decoding. First, it needs to decode the bitsteam of a frame in the buffer. Since variable length VQ is used, indices of vectors are of different bit-length. The received bitstream is a bitstream of (class, index) pair. The field of the class is of fixed length. The actual field length depends on the total number of classes. For example, if a 10-bit code-book is used, then 4 bits need to be reserved for the class field (4 bits can handle a codebook with size up to 16-bit). So decoder uses the bit-length info in the class field to read the following index, whose length could vary from 0 up to 10 bit in the previous example. The output of this first round of decoding is a sequence of (class index) pairs.

Secondly, for each pair of (class, index), its location in the sequence determines which subband it is in according to the protocol. And the corresponding subband codebook is used to find out the vector the index represented. The outcome of this second round of decoding is vectors stored in a buffer ready for the synthesizing module to use.

3.3.3 Synthesizing Module

As we can see, decoding and synthesizing is done reversely as compared to the encoding procedure.

First, each vector is decomposed into three vectors, one for each color component. The vectors of the same color component are put together to rebuild the transform

domain frame as depicted in Figure 3.3. Thereby three frames in the transform domain are formed for each frame in the video clip.

Second, subband decoding is performed for each transform domain frame. Using the same example given in Figure 3.3, first reverse subband coding is performed for subbands 6 through 9, the result is a subband in 2nd level, let's name it as subband $\hat{6}$. Next, this subband $\hat{6}$ together with subband 3 to 5 are synthesized to form subband $\hat{3}$, which in turn is combined with subband 2 to 0 to rebuild a single color component frame.

Once all the three color frames are rebuilt, they are ready to be converted to an appropriate representation for display.

3.3.4 Display Module

The display handles the display of video in the X window environment. The major task of the display is the dithering, which in essence is to convert 24 bit color frames into 8-bit, 16-bit color or grey-level representations that are supported by the machine. Once the conversion is done, the new representation of a frame is delivered to the X server, which takes over the display thereafter.

Basically, the Display Module is an X client, it requests and establishes the connection with the X server in the initialization stage. During this period, it allocates the available colors for the later display, it also allocates frame buffer from which the X server will load the frame [24]. It prepares the dithering tables during this time of period as well.

When it comes to dithering, it first quantizes the color representations into 8-bit representation for all the three color components. Usually, the 8 bits are divided as 3:3:2 or 4:2:2 among the three color components. Then different dithering algorithms can be used. In this system, ordered dithering is used for its superiority over error diffusion[25],[26].

Since there are quite some color representations of the original display, and the monitor usually supports only RGB format, the Display Module needs to do a conversion from an original format (suppose it's XYZ) XYZ to RGB after the

dithering. This final 8-bit RGB representation is put into the frame buffer for the X server to use.

3.3.5 User Interface

A simplified user interface was developed with two considerations in mind. One is that it should provide full VCR functions yet it should not consume much processing time. Another is the portability issue, the user interface should not be developed based on a complicated GUI software which may not be installed in many platforms. The user interface thus developed was solely based on Xlib which comes with every UNIX operating system. It provides a user simple and easy control. Such basic VCR functions have been provided: **TITLE, START, STOP, BACKWARD, FORWARD, GOTO,** etc. **TITLE** let user select from a list of clips, **START, STOP, BACKWARD, FORWARD** allow a user to control the play of the video to his or her like. **GOTO** enable a user to randomly position a specific frame in a clip.

Chapter 4

Implementation of the VOD System

The challenge in the implementation of the VOD system is the speed of the playback of the client. The goal is to be able to play back a video clip across a network of average workstations as close to real time (30 frames/second) as possible. In this chapter, we discuss some of the special implementation features designed with this speed issue kept in mind.

4.1 Encoding of Video

The resolution of the testing video clips is 352 by 224. The format of testing video clips used is YUV format. The spatial ratio of YUV is 4:4:4. As discussed later, any other video clips using different color space can be easily adopted into this system. The vector size used is 4×4. A 3-level subband coding is used. To avoid decoding complexity, a very simple Haar wavelet filter is used. By using this Haar wavelet filter, both encoding and decoding involves only addition and subtraction, no multiplications at all. There is barely noticeable quality deterioration, which justifies the use of Haar wavelet filtering. Codebooks (in this case: 10) are trained with maximum index bit length being 12. Again, to speed up the playback, the encoding is simplified to use only two classes of indices. Either the centroid (class

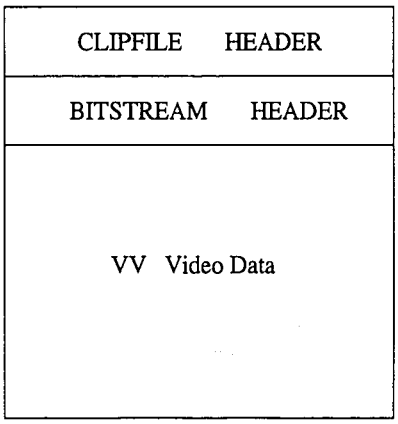

Figure 4.1: VV Video Clipfile Format

0) is used, or the 12-bit indices are used. This way, the compression performance is traded off for the decoding simplicity.

All the basic VCR functions we talk about in chapter 4 depend on the capability of the server to rapidly locate very single frame. For this purpose, the frame number and the offset of the frame in the video clip bit stream are stored at the beginning of the bitstream.

Figure 4.1 shows the format of the bit stream of a video clip.

Note that the location information of a frame in the bitstream is used only by the server, thus the bitstream header is not sent to the client across the network. However, the clipfile header might be sent over depending on the user's certain request. It could contain the following information: the image width and height, frames per second, total number of frames in the file and so on.

Recall in chapter 2, we discussed the YUV color space. Y is basically the luminance of the picture, U and V are the color information. As we know that the human eyes are more sensitive to the changes of Y component than changes of the U V components, we can afford to lose more precision in U V component than in Y component. So, one improvement we achieved in speeding up was through subsampling the U V frames in both directions by a factor of two. A low pass filter was applied to the sub-sampled U V frames to smooth them out. Thus we ended up with Y:U:V spatial resolution at 4:1:1. For the vector formalization, 4×4 pixels are still used for Y component, but only 2x2 pixels for both U and V component. The

vector size now becomes 24 instead of 48. What this down-sizing of vector leads to is the half size codebooks therefore speedup in both encoding and decoding.

4.2 Transmitting and Receiving Frames

The stream type socket is used for the transmission. This type of socket provides virtual circuit connection, thus guarantees that the bitstream be transmitted to the client in the sequenced order without error. The handshaking between the network interface modules on both sides is implemented by using a handshaking token–a 4-byte integer to synchronize the communication. Whenever the token is received, the NIM on the server side sends out one frame with an integer indicating the total number of integers attached to the beginning. Because the bitstream is sent out in multiples of 4 bytes, and the size of the bitstream of a frame may or may not be multiple of 4 bytes, stuffing is performed when the bitstream of a frame is not aligned.

At the receiving end, the NIM stores the first integer received, and checks if the total number of integers received matches with the first integer or not. If not, an error has occurred, and is reported to the User Interface Module.

The reason for this 4 bytes alignment is to accelerate the decoding of the bitstream by bit masking as discussed in the next section.

4.3 Decoder

The bitstream decoding is done by *getindex* submodule. A data structure called *layer_data* is used as in Figure 4.2.

Classes and indices are extracted by shifting and masking. An *integer* buffer (32 bit) *bfr* is used instead of a *short* buffer (16 bit) even the maximum length of an index is 12. This is to minimize the memory access time so as to reduce decoding time.

During the client initialization period, all the codebooks are sequentially loaded into one continuous codebook buffer in the order of highest subband codebook first

```
struct layer_data {
    int infile;
    unsigned char rdbfr[COM_BUFFER_SIZE];
    unsigned int bfr;
    int incnt;
    int bitcnt;
} ldata;
```

Figure 4.2: Data Structure for Bitstream Decoding

lowest subband codebook last. Because the decoding for the vectors is done in the exactly same order, the pointer to the beginning of each subband codebook gets incremented accordingly. So, for each index, the location of the corresponding vector in the codebook buffer can be very easily calculated.

We can see that in the above decoding, memory has been traded for the speed wherever possible.

4.4 Synthesizer

A diagram of the bit stream in terms of the organization of subbands is given in Figure 4.3. The procedures initially used to reconstruct a frame from such a bitstream is given in Figure 4.4:

In stage 1, function *PUTV* is used to reconstruct Y,U,V vectors in each subband from the combined vectors.

In stage 2, for each color component, inverse Haar wavelet transform *FOUR2ONE* is performed to rebuild the 2nd level Low-Low subbands from the 3rd level subbands.

In stage 3, for each color component, inverse Haar wavelet transform *FOUR2ONE* is performed to rebuild 1st level Low-Low subbands from the 2nd level subbands.

In stage 4, for each color components, inverse Haar wavelet transform *FOUR2ONE* is performed to reconstruct the three Y U V frames from the 1st level subbands.

Later, we realized that there is no need to separate Y U V during synthesizing, as

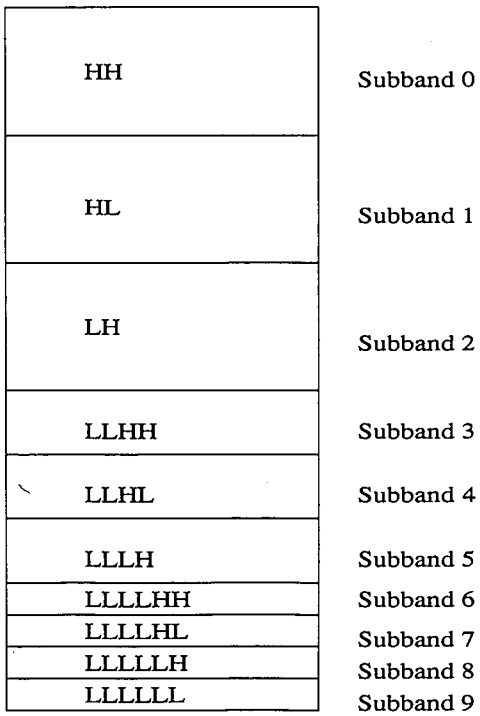

Figure 4.3: Subbands Organization in the Bitstream

long as the operations (i.e., addition and subtraction) are performed correspondingly, namely, Y +/- with Y, U +/- with U, and V +/- with V. In other words, the synthesizing can be performed at combined vector level. By doing so, stage 2 can be eliminated, which gave us speed gain at no performance loss.

Another place where we simplified the calculation is the manipulation of the factor $\sqrt{2}$ in the Haar wavelet filter. To avoid floating point computation, we didn't use this factor. To make up on the decoding side, we need to divide the result of the filter by $2 = \sqrt{2} \times \sqrt{2}$, which can be simply implemented as shifting one bit right.

4.5 Display

Display Module first creates three tables during the initialization period of the system. Two quantization tables and a color table.

In both of the quantization tables, there are total 2^{16} entries, with each entry corresponding to a one-byte value. One quantization table is used to map 16-bit

```
SYNTHESIS(bitstream, vector_sequence)
{
      PUTV ( bitstream, YHH, UHH, VHH );
      PUTV ( bitstream, YHL, UHL, VHL );
      PUBV( bitstream, YLH ,ULH, VLH );                          ; stage 1
      ......
      PUBV ( bitstream,  YLLLLLLH,  ULLLLLH,  VLLLLLH );
      PUBV ( bitstream,  YLLLLLLL,  ULLLLLL,  VLLLLLL );

      FOUR2ONE ( YLLLLLL,  YLLLLLLH,  YLLLLHL, YLLLLHH, YLLLL );
      FOUR2ONE ( ULLLLLL,  ULLLLLLH,  ULLLLHL, ULLLLHH, ULLLL);  ; stage 2
      FOUR2ONE ( VLLLLLL,  VLLLLLLH,  VLLLLHL, VLLLLHH, VLLLL);

      FOUR2ONE ( YLLLL, YLLLLH, YLLHL, YLLHH, YLL);
      FOUR2ONE ( ULLLL, ULLLLH, ULLHL, ULLHH, ULL);              ;stage 3
      FOUR2ONE ( VLLLL, VLLLLH, VLLHL, VLLHH, VLL);

      FOUR2ONE ( YLL,  YLH,  YHL, YHH,  Y );
      FOUR2ONE ( ULL,  ULH,  UHL, UHH,  U );                     ;stage 4
      FOUR2ONE ( VLL,  VLH,  VHL, VHH,  V );

}
```

Figure 4.4: Synthesis Procedures

UV (8:8) into 8-bit UV (4:4). The other quantization table is used to map 16-bit YUV (8:4:4) into 8-bit YUV (4:2:2). However the entry values for the second quantization table only range from 32 to 239. One straight forward solution could be using simply just one quantization table with 2^{24} entries. But that would require 2^7 times memory space than using two tables.

In the color table, there are total 208 entries corresponding to 208 (R,G,B) color pixels in the chosen color map of the X server. The indices of these entries range from 32 to 240. First color conversion is performed to convert these YUVs (4:2:2) into (R,G,B)s. The requests for these (R,G,B)s are then sent to the X server. If granted, the returned pixels by the X server are stored in the table. The reason why only 208 colors instead of 256 colors are allocated for the display is simply to avoid the flickering of screen. Because if all 256 colors were allocated to the video display, the other applications that are using some colors on the same screen will flicker (blackout). By reserving 48 colors (0-31 and 240-255) for other possible applications, this flickering problem can be avoided.

When the display module actually displays the decoded frames, the following actions are taken: First, every 4×4 block in the decoded frame is added by a 4×4 ordered dithering threshold mask as shown in figure 4.5. Then, each 24-bit (Y,U,V) triple is quantized into 8 bits by looking up the YUV quantization tables. Next, the color table is used to find all the color pixels. Finally, the pixel stream is sent to the X server for display. After a couple of testing runs of the above implementation, we noticed that the display module took up the largest chunk of time in the whole play back. To address this problem, the following modification was made.

As the ordered dithering performs very well in general, we don't see any particular reason to provide the flexibility of switching back and forth among different dithering algorithms. Once this is fixed, there is no need to perform the dithering during the run time when all the thresholds and distribution pattern are already known. An innovative approach is taken to modify the codebook by applying the ordered dithering threshold matrix to the vectors in the codebooks instead. So, the additions involved during the dithering are all removed, the dithering time is saved. All the display module does are mappings through table lookups.

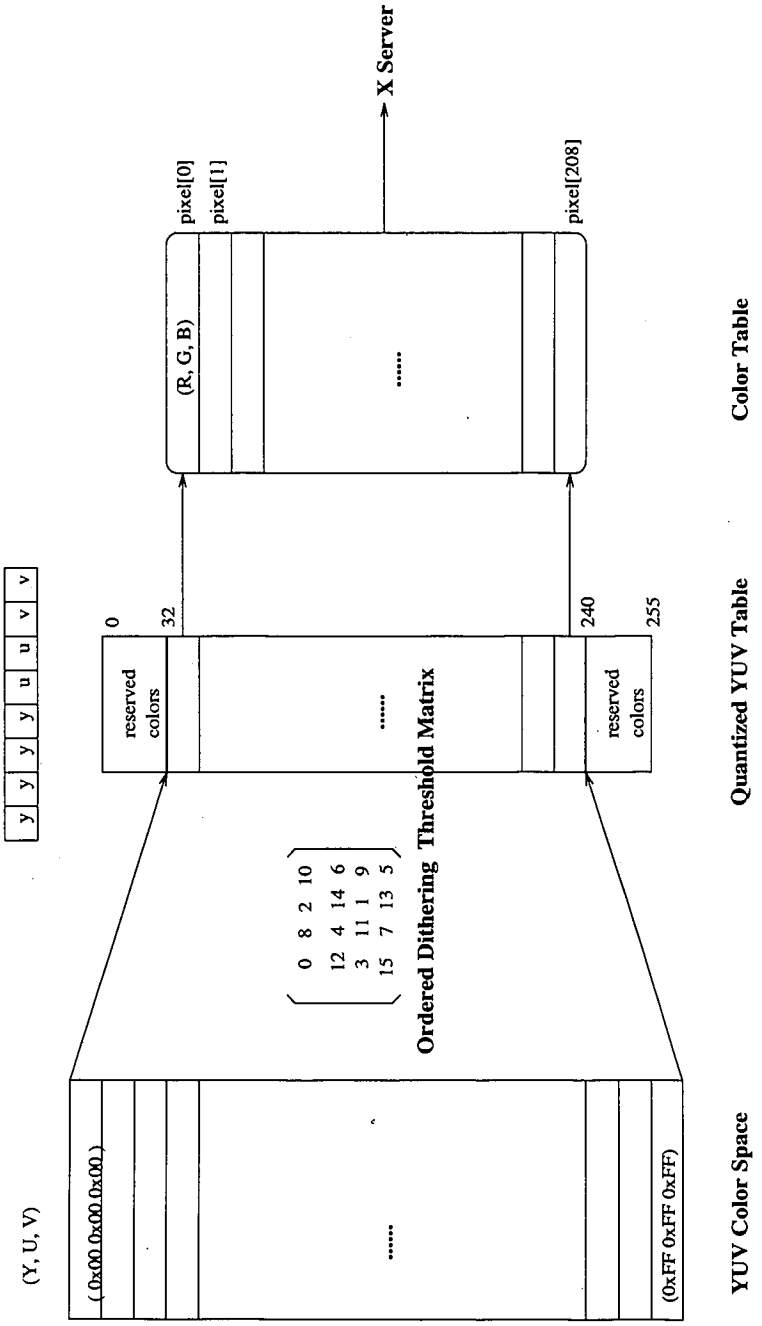

Figure 4.5: Color Conversion and Dithering in the Display Module

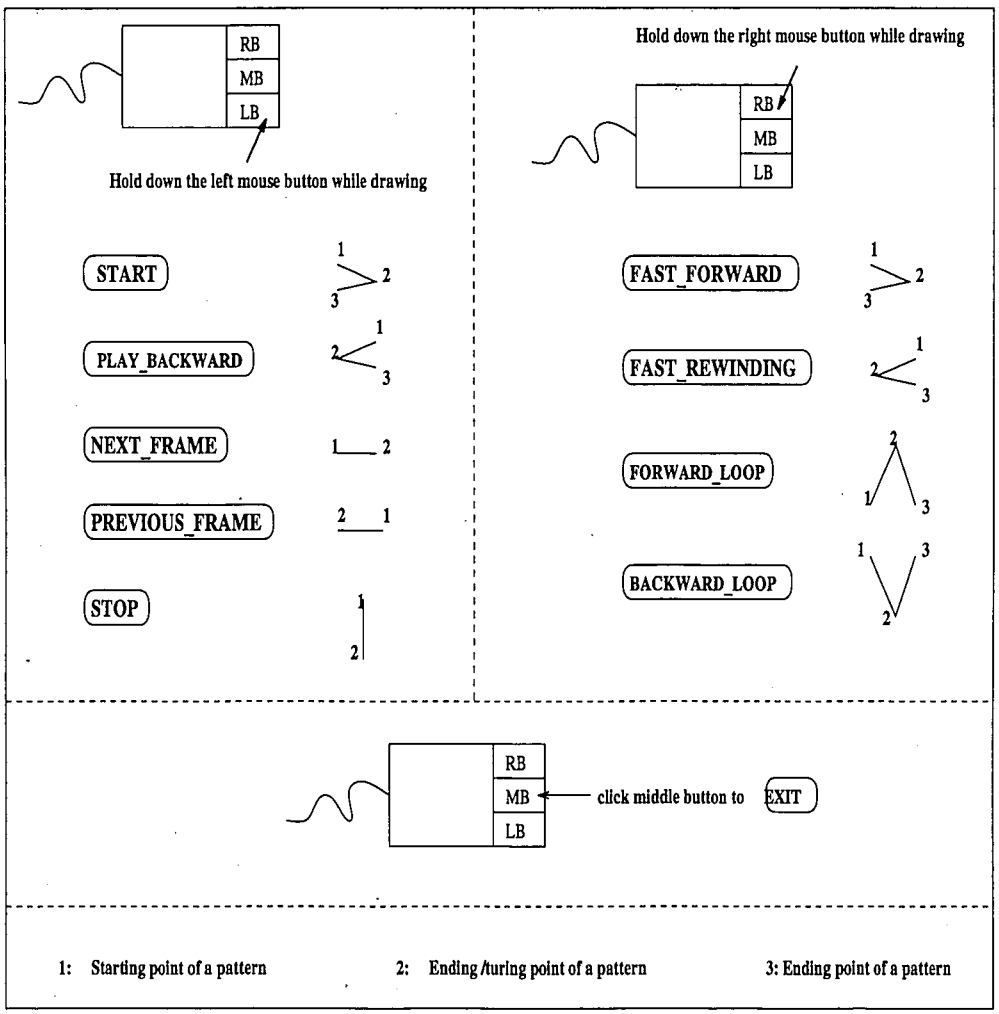

Figure 4.6: How to play the video

4.6 User Interface and Internet Interface

The user-interface is a very simplified one. However, it supports all the VCR functions shown in figure 4.6. At the lowest programming level of X Window, Xlib offers both portability of the code and the speed up for the display. Unlike Xt which are libraries built upon Xlib, and Motif which is built on top of Xt, Xlib allows for flexibility but lack the nice feel-and-look of common GUIs. Limited by the time, we developed, using directly Xlib system calls, an intuitive graphic user interface for a user to play the video and to give different kinds of commands..

This user interface works like this: a user issues different kinds of commands by drawing different kinds of patterns (as shown in figure 4.6) in the display window. Because all the points along the trace of this drawing are preserved by the X server, we can differentiate different patterns using a simple pattern recognition technique. Basically, this pattern recognition algorithm first tries to decide if there are two "lines", if so then the angle between two lines is calculated. Certain amount of tolerance is allowed so a user does not have to draw a straight line. In the worst case, if a pattern can not be recognized, it is simply ignored.

As we discussed in the previous chapter, an interface to the internet of the VOD system was developed utilizing the common gateway interface. The one-button feature scheme is depicted in figure 4.7:

The steps taken from a user click a button to order a video to the user start enjoying it are given as below:

- Step 1: A user clicks a button (Actually two, first he selects the "cool demo", which leads him to the site of video server on the internet. Then the second one "start", which allows him to start the order-wait-watch process. Some times a third button "OK" might show up to let the user confirm his desire to view video regardless the security issue.)
- Step 2: Transparent to programmers. The *browser* sends the user's request and information to the *httpd* server.
- Step 3 and Step 4: The *httpd* server starts a *cgi1* program, which sends the viewer program (binary executable) and codebooks (all uuencoded) to the user over the connection between the *httpd* and the *browser*.
- Step 5: The *browser* sets up the viewer program and starts it running (this is when the third button might show up if the *browser* takes the security issue seriously). The *cgi1* exits once the data transmission is over (stateless characteristic of HTTP). Hence the connection between the *browser* (user) and the *httpd* server ends.

  For the purpose of synchronization between the *viewer* and the video *provider*, a record file storing the user info (very short) is generated by *cgi1*.

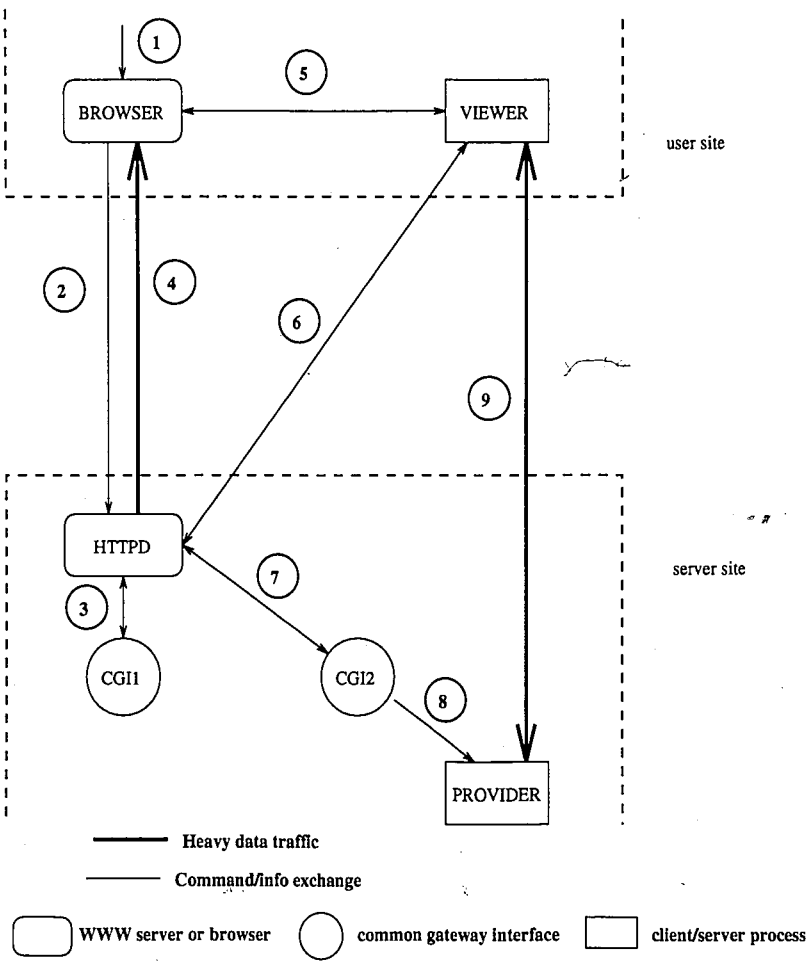

Figure 4.7: Implementation Scheme of One-Button Video-On-Demand Using CGIs

- Step 6: The *viewer* now running on the user machine wants to start the video *provider*, which run on the server machine. It does so by asking the *httpd* server to start up the *cgi2*. Meanwhile, it waits for a reply from *cgi2* about the video server's address.

  The reason for not starting the provider (or video-server) in *cgi1* is to avoid the scenario where the client got the viewer program but for some reason not starting it, and on server side the server is occupying the service address, and there is no way for the *httpd* to find out this so it can stop this meaningless waiting.

  Since the *httpd* has the power to allow multiple *cgi2*s running at the same time, by using a *cgi2* to dynamically allocate a port number, we can running multiple *cgi2*s to start multiple video *provider*s running at different ports to satisfy multiple requests.

- Step 7: The *httpd* starts the *cgi2* upon the request of the *viewer*. The *cgi2* finds an unused port number for the video *provider*, also it sends this video *provider*'s address to the *viewer* through the *httpd*.

- Step 8: The *cgi2* then starts the video *provider*. To make sure the *viewer* get the address of the video *provider* and successfully connect with it, *cgi2* waits until the video *provider* and *viewer* establishes the connection, then retires. Due to the fact that the traffic on the network can change dramatically and the user machine may have multiple users, it can not be predicted which of these two operations is faster, the transmission over network of the port number from the *cgi2* to the *viewer* or the system call to start the video *provider* locally.). If *cgi2* quits too soon, *viewer* might request connection to the video *provider* before the video *provider* gets started. This is avoided by using the request record file as a token, when the video *provider* loads in the information in the record file, it deletes it. *cgi2* then detects that the record file no longer exists, so it can retire safely. If it quits too early, then the record file won't get deleted. So, the user gets a "resume or ignore" message, from which point, it can resend the request.

- Step 9: The *viewer* directly communicate with the video *provider* happily thereafter.

It might look redundant to use multiple gateway programs in the above implementation scheme. Unfortunately, this is limited by the stateless characteristic of HTTP protocol. Nevertheless, we have proved that it is possible to realize compression independent technology video delivery by specifying a higher level protocol (In this case, it is the HTTP) rather than using a specific standard which uses a specific compression technology.

Chapter 5

Results and Discussion

The test sequence we used to obtain the following results is a YUV 4:4:4 football sequence which contains 145 frames at the resolution of 352 by 224. This sequence is obtained by first subsampling the original YUV 4:2:2 football sequence, which is 720 by 240 at 60 fields per second into 360 by 240, using only the odd fields. A horizontal low pass filtering is applied to the luminance component before the subsampling by a factor of 2. The low pass filter is given in equation 5.1. Then the spatial resolution is cut into 352 by 224 because the VTC requires that both dimensions be multiples of 32.

$$y_i = (-y_{j-3} + y_{j-1} \times 9 + y_j \times 16 + y_{j+1} \times 9 - y_{j+3})/32 \tag{5.1}$$

The MPEG-1 software encoder and decoder we used for comparison in this thesis are ftped from Berkeley, which implement the standard described in the ISO/IEC International Standard 11172 [27],[28]. Since the inter-frame compression is not implemented in the VOD system, to be fair, we compared our VOD system and MPEG-1 with the interframe compression of MPEG-1 turned off.

Since this VOD system is an asymmetric system as we discussed before, it is obviously that we need to evaluate it at encoding end and decoding end differently, where at the encoding end, the compression performance is the major concern, while at the decoding end, the playback speed is the most important issue.

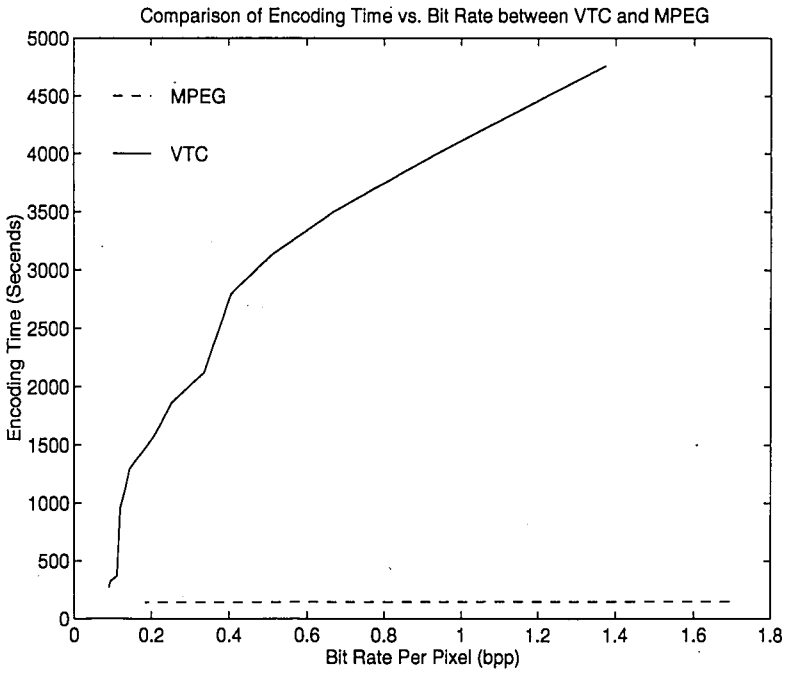

Figure 5.1: Comparison of the VOD System and A MPEG-1 Encoder: Complexity vs. Bit Rate

5.1 Encoding Results

- Complexity vs. Bit Rate

  The off-line encoding time is a very good indication of the complexity of a compression algorithm. So we used the encoding time instead of the number of computations as the measurement of complexity in the comparison between MPEG-1 and the encoder of our VOD system in Figure 5.1.

  In the VTC, the different bit rates are obtained through changing the vector quantization scale, which in turn changes the threshold of the VQ. In MPEG-1, the bit rates are changed through changing the scalar quantization factor.

  From figure 5.1 we can see that the encoding complexity of VTC is much higher than that of MPEG-1. This is understandable because VTC uses Vector Quantization which requires a lot of time in searching for the best match

whereas MPEG uses scalar quantization which is a very simple division operation, which also explains why the complexity of MPEG encoding is a almost constant. However, for the VTC, the complexity decreases dramatically with the decrease of bit rate, this is because more and more centroids are used when decrease the bit rate. The more centroids are used, the less time (computation) is needed for the search.

The idea is that by putting more computation on the encoding side, we can hopefully gain simplicity on the decoding side. This asymmetry of complexity of VTC is the very characteristic that makes it very useful in applications like VOD systems.

- SNR vs. Bit Rate

  As we discussed in Chapter 2, SNR is not an ideal distortion measurement for VQ, because it does not represent the human visual perception very well. But since there is no available perceptional distortion measurement, SNR is still a fair distortion indicator. To compare the compression performance between the VTC used in our VOD system and that of MPEG-1, we calculated the average peak SNR of the football sequence at different bit rate, and the results are plot in figure 5.2. From figure 5.2 we can see that at higher bit rate (smaller compression ratio), MPEG performs better than VTC in terms of peak SNR. However, at very low bit rate, VTC performs better than MPEG-1, and the turning point seems to be at bit rate 0.35 with Peak SNR around 30 db.

5.2 Decoding Results

The encoding results showed that VTC encoding is much more time consuming than MPEG encoding so we can, hopefully, simplify the decoding end.

First, we played our VOD system and MPEG player on two platforms, and got the average results shown in Table 5.1.

Note frame rate is used as an indication of the decoding time here. This is not quite accurate. Because we know that the playback time of the VOD system

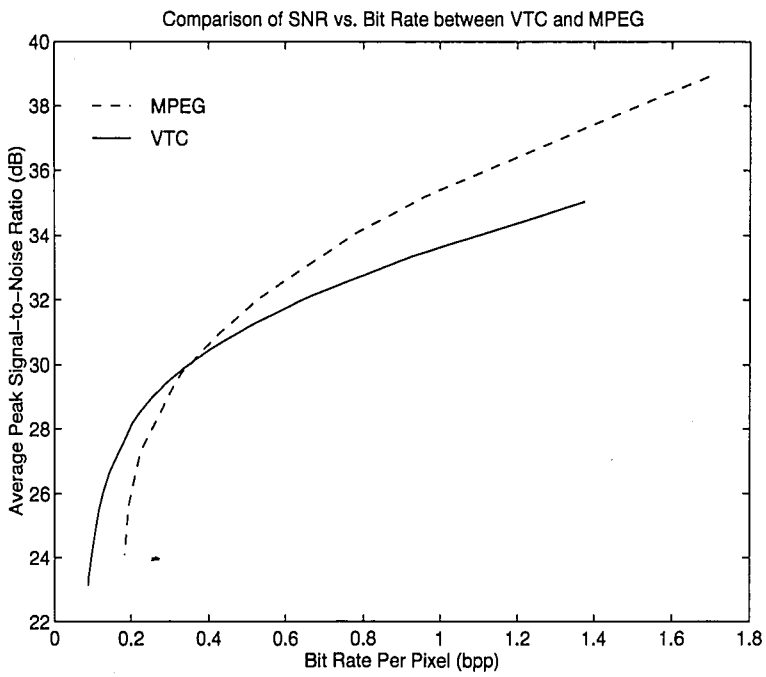

Figure 5.2: Comparison between the VTC and MPEG-1: Average Peak SNR vs. Bit Rate

compression program	play back speed (fps)	
	IBM RS6K	SunSparc 20
MPEG-1	4.5	18.1
VTC	11.0	19.9

Table 5.1: Play Speed Comparison

can be divided into three time components: the decoding, the synthesizing and the actual displaying. But because the displaying modules in both the VOD system and the MPEG-1 decoder use very similar techniques, this total playback time is an acceptable indicator of the decoding of these two algorithms. For the VTC, because the bitstreams actually go through the network while the MPEG does not, extra overhead are added there. However, this network overhead difference is minimized by running server and player of the VOD system on the same machine.

However, the seemingly different behavior on the two different platforms comes from the memory sharing feature implemented in the MPEG-1 player. In our VOD system, this feature has not been implemented yet. The memory sharing is disallowed on the IBM RS6K workstations that we tested these bitstreams on, the results are fair and representative. So, the playback of VTC is indeed faster than that of MPEG-1 as we expected. The playback results of these two algorithms at different bit rates on IBM RS6K are given in figure 5.3 and figure 5.4. Because, the decoding of VTC is basically a table look up, so the decoding time (or the playback time) is almost constant. But for DCT, the decoding time decreases with the decrease of bit rate. This can be explained: because the lower bit rate corresponds to larger quantization factor, which means more zero coefficients in the DCT transform domain, which means run length coding employed by MPEG works very well to compress the coefficients. In return, less coefficients are needed to reconstruct the images, therefore the decoding time goes down as the bit rate drops.

On the other hand, on the SunSparc 20 work station, shared memory is allowed, and MPEG-1 player takes full advantage it. So, the frame rate of these two players are very close. In table 5.2, the playback time decomposition of the VOD system also indicates this possible speed up. Because the display takes about half of the total playback time, by using shared memory to allow pipeline between display and the other two modules together, the playback can be approximately doubled. From this, we can draw the conclusion that once the shared memory is employed in our VOD system, the playback speed should at least be doubled. Decomposition of time spent for the playback of a compressed bitstream at a fixed bit rate (vq=3) is given in Table 5.2:

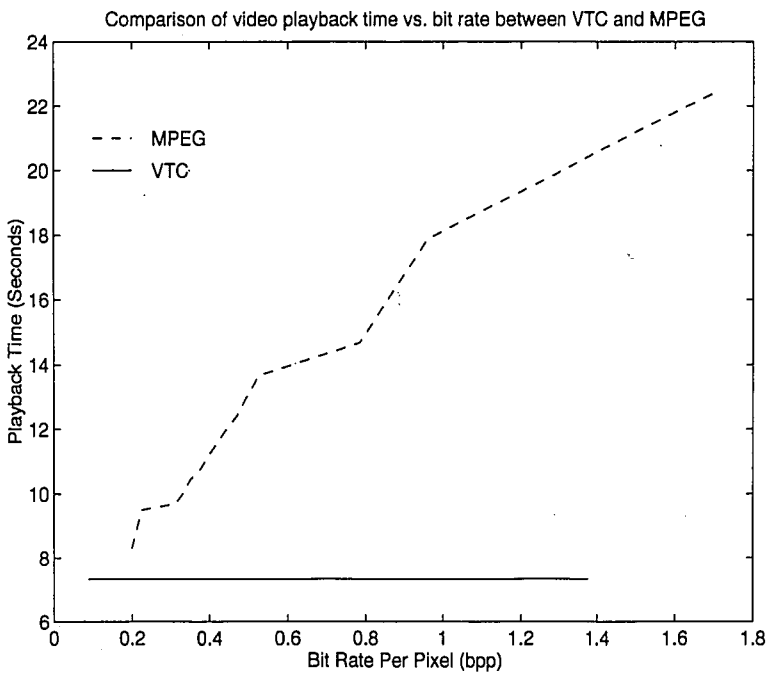

Figure 5.3: Comparison between the VTC and MPEG-1: Playback Time vs. Bit Rate

environment	decoding(%)	synthesis(%)	display(%)	fps
SunSparc20	29.5	25.5	45.0	19.90
IBM RS6K	30.5	19.0	50.5	11.02

Table 5.2: Decomposition of the VOD Playback Time

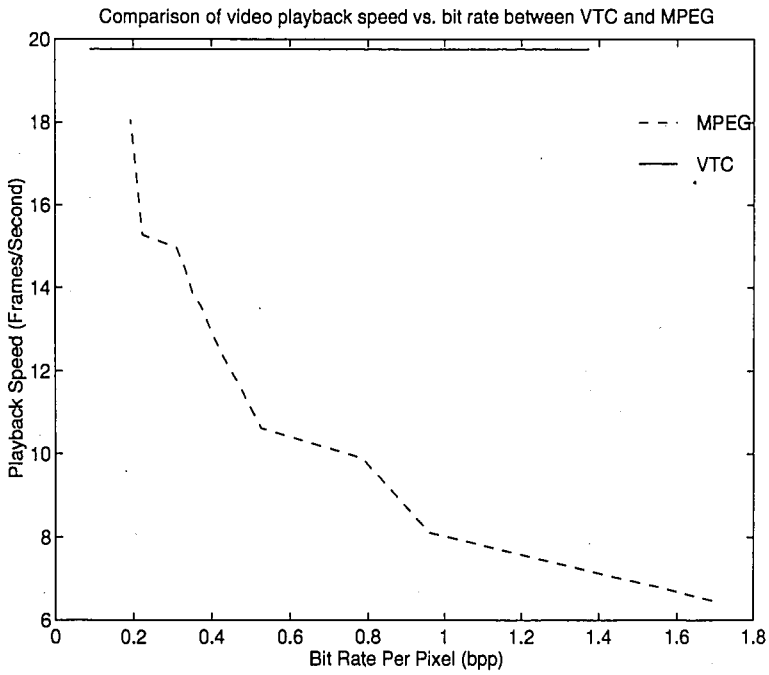

Figure 5.4: Comparison between the VTC and MPEG-1: Playback Speed vs. Bit Rate

Latency :

User response time is a very important aspect of a VOD system. It is the time period between the user making a demand and the moment the video player responds (such as start playing video or stop playing video, etc.). As we found out that because of the randomness of network traffic, the user response time changes in a vary large range.

Chapter 6

Conclusions and Future Work

Based on the above results of the developed VOD system, we can draw the following conclusions:

First, VTC compression algorithm is well suited in asymmetric information-on-demand technology, such as Video-On-Demand system over a computer network, because the playback is always faster than that of MPEG at comparable compression quality.

Second, as our one-button VOD feature demonstrated, it is possible to develop application level communication protocol for compression technology independent information delivery. The restriction of using a specific compression technology hence using a specific player for desktop applications can be eliminated. This allows people with different compression package to exchange information like image and video on a computer network more flexibly. More importantly, it avoids the problem of stagnant compression standards standing in the way.

The playback of our VOD system has not achieved real time yet. However, according to the above results, we believe that by implementing the shared memory technique we can easily achieve real-time on the SunSparc workstation. Even though it is a self-contained VOD system which supports full VCR functions, there are still many features to be implemented for the VOD system to become a full fledged one. Many of these features belong to broader areas of study, such as the ones listed in the following:

- **Compression Algorithms**

  Only intra-frame compression is implemented. Neither inter-frame compression (motion estimation and motion compensation) nor entropy coding has been applied, but these two methods can add considerable compression to the system at some extra computational overhead. The immediate next step is to add in the inter-frame compression to find out the computational cost for higher compression performance. As we said, this VTC technology is still not mature yet, more studies are needed to optimize the compression scheme meanwhile.

- **Operating System**

  As we saw in the results, the display module takes about half of the playback time, some programming technique like shared-memory can improve the speed, but more fundamentally, the Xlib functions were not designed for video display, so a better generic display method should be developed. Same goes to the current operating systems, now this is beyond the range of the topic of this thesis, but we believe that with video data becoming more and more important new operating systems should be designed and developed to be able to accommodate video data.

- **Video Server**

  Right now, the VOD server can fork off multiple children to handle multiple requests from different machines across the network. But, all these children processes are running on the same server machine, which brings down the user response time dramatically. The immediate next step is to distribute these requests to different machines on the network through distributed programming. In our study, we only used a very small testing sequence. In real situation, the video server needs to manage a huge video database. How to efficiently manage a big video archive and give prompt response to different kinds of requests still remains a big challenge.

- **Security**

  Security has always been a touchy issue. In the developed VOD system, this is totally relied on the HTTP. In our hypothesis, we assume that it is the

executable of player will be delivered to a user to get rid off the compression technology dependency, what HTTP does is simply warning the user that the forthcoming executable player is going to run on his/her machine. A user can refuse the playing at this moment, but once he/she accepts the player, the control is totally in the player program's hand. Since a user has no way to decide if the player is safe or malicious, this is obvious a security hole. However, our argument is that this problem exists in any distribution of commercial software.

Bibliography

[1] A. Jacquin, "Fractal image coding based on a theory of iterated contractive image transformations," in *Proceedings of SPIE: Visual Communications and Image Processing*, vol. SPIE Vol. 1360, pp. 227–239, 1990.

[2] Y. Fisher, "Fractal image compression," Tech. Rep. vol 12 pp 7.1-7.19, Department of Mathematics, Technion Israel Institute of Technology, 1992. SIGGRAPH '92 COURSE NOTES.

[3] K. Aizawa and T. S. Huang, "Model-based image coding: Advanced video coding techniques for very low bit-rate applications," *Proceedings of the IEEE*, vol. 83, pp. 259–271, February 1995.

[4] W. Li, "Vector transform and image coding," *IEEE Trans. on Circuit and Systems for Video Technology*, vol. 1, pp. 297–307, Dec. 1991.

[5] W. Li, "On vector transformation," *IEEE Trans. on Signal Processing*, pp. 3114–3126, Nov. 1993.

[6] J. D. Foley, A. van Dam, S. Feiner, and J. Hughes., *Computer Graphics: principles and practices.* Addison-Wesley, 2 ed., 1990.

[7] D. Bourgin, "Color space FAQ." comp.graphics newsgroup on the internet, February 1995.

[8] P. Heckbert, "Color image quantization for frame buffer display," *ACM, Computer Graphics*, pp. 297–304, 1982.

[9] S. Wong, S. Wan, and P. Prusinkewicz, "Variance-based color image quantization for frame buffer display," *Color Research and Application*, vol. 15, pp. 52–58, February 1990.

[10] M. Gervautz and W. Purgathofer, "A simple method for color quantization: Octree quantization," in *Graphics Gems* (A. S. Glassner, ed.), pp. 287–293, Cambridge, MA: Academic Press Professional, 1990.

[11] A. Dekker, "Kohonen neural networks for optimal colour quantization," *Network: Computation in Neural Systems*, vol. 5, pp. 351–367, 1994.

[12] O. A. Verevka and J. W. Buchanan, "Local k-means algorithm for color image quantization," *Proceedings of GI*, to appear in 1995.

[13] B. Smith, "Multimedia system." Cornell University, CS610 Course Notes, Fall, 1994.

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

[15] W. Li and Y.-Q. Zhang, "Vector-based signal processing and quantization for image and video compression," *Proceedings of the IEEE*, vol. 83, pp. 317–334, February 1995.

[16] A. Gersho and R. M.Gray, *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, 1992.

[17] W. Li, J. Wus, and Y.-Q. Zhang, "New vector subband coding for image and video compression," *Proc. NJIT Symp.*, February 1994.

[18] W. Li, J. Wus, and Y.-Q. Zhang, "Image coding using vector filter bank and vector quantization," *Proc. IEEE ISCAS*, pp. 133–136, May 1994.

[19] P. Pirsch, N. Demassieux, and W. Gehrke, "VLSI architectures for video compression–a survey," *Proceedings of the IEEE*, vol. 83, February 1995.

[20] CCITT, "Video codec for audiovisual services at $p \times 64$ kbit/s," tech. rep., CCITT, 1990; revised 1993. Recommendation H.261.

[21] S. Okubo, "Reference model methodology–a tool for the collaborative creation of video coding standards," *Proceedings of the IEEE*, vol. 83, February 1995.

[22] L. Chiariglione, "The development of an integrated audiovisual coding standard: MPEG," *Proceedings of the IEEE*, vol. 83, February 1995.

[23] J. Martin and J. Leben, *TCP/IP Networking, Architecture, Administration, and Programming.* P T R Prentice Hall, 1994.

[24] J. Gettys and R. W. Scheifler, *Xlib - C Language X Interface.* MIT X Consortium Standard, first revision ed., August 1991.

[25] K. D. Patel, B. C. Smith, and L. A. Rowe, "Performance of a software MPEG video decoder," *First ACM International Conference on Multimedia*, pp. 75–82, February 1993.

[26] L. A. Rowe, K. D. Patel, B. C. Smith, and K. Liu, "MPEG video in software: Representation, transmission, and playback," *IS&T/SPIE Symposium on Electronic Imaging: Science & Technology*, February 1994.

[27] L. A. Rowe, K. Gong, K. Patel, and B. Smith, *MPEG-1 Video Software Encoder.* Computer Science Division-EECS, Univ. of Calif. at Berkeley, 1.3 ed., March 1994.

[28] L. A. Rowe, K. Patel, and B. Smith, *MPEG Video Software Decoder.* Computer Science Division-EECS, Univ. of Calif. at Berkeley, 2.0 ed., January 1993.

Appendix A

Biography

Keren Hu was born in Chengdu, Sichuan Province of China on September 8, 1968 to Ping Hu and Deshu Xie. She graduated from the University of Science and Technology of China in July 1991 with a B.S degree in Electrical Engineering. Upon graduation, she was awarded the Regent's Fellowship to attend Texas A&M University. While at Texas A&M University, she studied and researched in the application of Artificial Intelligence in geographic information system. In 1993, she was awarded scholarship to attend Lehigh University. Since September 1993, she has attended Lehigh University performing research in the areas of image and video compression and VLSI signal processing. Her current research interests include digital signal processing, image processing and VLSI.

END OF TITLE