SPATIAL DATA STRUCTURE INDEXING FOR VIDEO DATABASES

by

Enhai Xie

B.Sc. (Analytical Chemistry), Hebei University, P. R. China, 1982
M.Sc. (Physical Chemistry), Hebei University, P. R. China, 1990
M.Sc. (Physics), University of Oldenburg, Germany, 1991

A Thesis Submitted in Partial Fulfilment of
The requirements for the Degree of

Master of Computer Science

in the Graduate Academic Unit of Computer Science

Supervisor: Bradford G. Nickerson, Ph. D., Faculty of Computer Science

Examining Board: Bernd J. Kurz, Ph. D., Faculty of Computer Science, Chair

Alejandro Lopez-Ortiz, Ph. D., Faculty of Computer Science

David J. Coleman, Ph. D., Faculty of Engineering

This thesis is accepted

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

December, 1999

ABSTRACT

This thesis explores the application of k-d range search data structures to digital video data. Content-based queries based on colour histogram indexing and colour space indexing for MPEG-2 video I-frames are defined and implemented. Experimentation using 100 MPEG-2 video clips comprising 5706 I-frames was carried out.

Two spatial data structures (a 4-d tree and a colour space tree) can satisfy queries such as "find MPEG-2 video I-frames with $> t$ % pixels in colour range $[R_L, R_H]$" or "find MPEG-2 video I-frames with $> t$ % pixels in color range $[R_L, R_H]$, $[G_L, G_H]$, $[B_L, B_H]$, simultaneously ", respectively.

Prototype software was written in C++ on a UNIX system to test both histogram and colour space range search. A 4-d tree containing 5706 I-frames from 100 MPEG-2 video clips was built in 71seconds. The average histogram range search time was 37 seconds on a Sun Microsystems ULTRA 5 workstation. A colour space tree of 300 I-frames required 5491 seconds to build, and 2 seconds to answer a colour space query on a Sun Microsystems ENTERPRISE 250 workstation.

ACKNOWLEDGMENTS

I wish to thank Dr. Bradford G. Nickerson, my supervisor, for his able supervision and continual guidance throughout the problem formulation stage, the conduct of the research and preparation of this thesis. Through weekly discussions, his valuable advice and insightful thoughts have greatly contributed in the overall shaping of this research work. I also wish to express my profound gratitude to the Natural Sciences and Engineering Research Council of Canada and the Faculty of Computer Science, University of New Brunswick for funding my research work.

Many thanks are due to Dr. Bernd J. Kurz, Dr. Alejandro Lopez-Ortiz, and Dr. David J. Coleman for their valuable comments and suggestions that improved the overall presentation of the work.

Thanks are also extended to Kirby Ward and Sean Seeley in the Faculty of Computer Science for the continued technical advice and support to complete the experimental portion of this research.

Lastly, I thank my wife Ying Gao and children, for their support, encouragement, and drive, which contributed immensely to the production of this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1 INTRODUCTION

Rapid advances in electronic imaging, digital video capture and storage, and the wide availability of digital data via the Internet have contributed to a large amount of digital imagery and video being available online. Video has become an important element of multimedia computing and communication environments, with applications as varied as broadcasting, education, publishing, and military intelligence.

Digital video indexing techniques are becoming increasingly important with the advent of broadband networks (e.g. those that use a high-speed Asynchronous Transfer Mode (ATM) protocol) and video compression standards (Moving Picture Experts Group) [Furh95, Grin98, Hask97]. MPEG-2 is currently being deployed in digital video service roll-outs worldwide and is the compression standard used in the majority of digital broadcast satellite systems. The ATM Forum adopted specifications for transporting MPEG-2 compressed digital video over ATM networks. Potential application areas include image and video databases, digital TV and World Wide Web searching of video.

Tools for storing and indexing such types of data in a video database are starting to become available [Adal96, Adje97, Zhan95a]. The application of k-d range search data structures to non-traditional spatial data is explored here. For example, point-based spatial indexing methods for video data is investigated. Fundamental research investigating the types of keys necessary and useful for searching in video databases is required.

The primary aim of this thesis is to study methods of indexing video databases so as to store and search video data accurately and efficiently. Experimentation has been done with 100 MPEG-2 video clips containing 5706 I-frames (see chapter 2 for a further definition of I-frames).

1.1 Background and Motivation

Video indexing is the process of identifying the range of video frames in which a certain pattern (an object, target, face, or signature) appears. Due to the multitude of parameters involved (each video sequence typically has associated text and/or speech associated with it) and the amount of data that must be processed, video indexing is an extremely computationally intensive operation and is therefore rarely available to the end user. The lack of standard techniques for video indexing further complicates the problem [Mitk 98].

Traditional database systems index their data by key attributes which are usually numeric or fixed-length text data. Keywords are by far the predominant method used to index video data [Adje97]. Adali et al [Adal96], for example, developed indexing structures based on keywords and a query-processing algorithm for querying the video data. Two examples they give are "Find all video frames in which John Wayne appears" and "Find all video frames in which Dean Martin appears and Fred Astaire is dancing with Ginger Rogers".

To accelerate searching of specific data items, most database systems use a tree indexing mechanism. Classical indexing trees such as k-d trees and quad trees [Same90a,

Same90b] are formed by recursively partitioning the feature space with partitions that are perpendicular to the co-ordinate axes. In this thesis, we will investigate whether or not these point-based spatial indexing methods are useful for indexing digital video data.

As more and more video data are acquired, managing them in a video or multimedia database becomes an important issue to be resolved. One objective of researchers in this area has been the development of techniques that provide the ability to store and retrieve images or videos based on their contents. One way to make video content more accessible is to store it in a database. An indexing scheme is useful when it is associated with the coding scheme used for storing the images or videos in the database.

Generally speaking, an index consists of a collection of entries, one for each data item, containing the key for that item, and a reference pointer which allows fast access to that item. To build a content-based index, we must begin by identifying the elemental index units for video content. In the case of text, these units are words and phrases, the entities we find in the index of any book. For content-based indexing of video, the search keys we consider are the histogram values and pixel values of frames within a video stream.

1.2 Literature Review

There are several notable papers on the field of video indexing. Existing work on video indexing can be grouped into two main categories; content-based indexing and content-independent indexing.

1.2.1 Content-independent indexing

Most previous research in video indexing has been based on indexing text [Davi93, Rowe94]. Text indexes tend to be structured as trees. Thus, what the keys are and what the users search for become the important issues. Smoliar et al [Soml94] give a tree structure of topical categories for a documentary video about engineering at the National University of Singapore. The tree structure represents relations of specialization and generalization among these categories. Zhang et al [Zhan95a] developed a text index scheme for a video news database based on a tree of topical categories, where news items are assigned to leaves of the tree. Each node of the tree then corresponds to a collection of news items that have some amount of topical information in common. Adali et al [Adal96] describe how video data can be organized and structured so as to facilitate efficient querying. They develop a formal model for video data and show how the segment tree structure provides an elegant way of storing such data.

1.2.2 Content-based indexing

Content-based indexing creates indexes to facilitate fast content-based retrieval of video objects in large databases. Intensive research has focused on this field in recent years, with the goal of indexing the video data using certain features derived directly from the data. With video data, the video sequence is first separated into its constituent scenes, then representative abstractions (usually key frames) are selected to represent each scene. Further indexing on the video is based on the key frame [Adje97]. Smoliar et al [Soml94] developed a prototype system with fast image-indexing abilities. This system

automatically computes numerical index keys based on colour distribution, prominent colour region segmentation, and colour histogram (as a texture model) for each image. Zhang et al [Zhan95b] describe techniques for use in the pixel domain for dealing with the representation of video clips, as well as content-based retrieval techniques using key frames and temporal properties of video clips. They present techniques for video parsing in the pixel domain followed by key frame extraction. The representation of video clips is based on several types of features, including colour histogram and moment features, texture features, shape features, and edge features. Flickner et al [Flic95] describe the QBIC system, which performs content-based retrieval based on colour, shape, texture, and sketches in large image and video databases. Ardizzone et al [Ardi96a, Ardi96b] also dealt with content-based video indexing based on motion, color, and texture and other global features. Pentlant et al [Pent94] described a photo-book system, which is a set of interactive tools for browsing and searching images for faces, shape, and texture.

1.3 Thesis Objectives

The main objective of this thesis is to determine what the appropriate keys for indexing video data are, and to explore the use of appropriate k-d spatial data structures to index and allow for quick searching of video data. An index of object keys is used in a database system to speed up searches for specific objects or objects falling within a specified key range. Video typically has text descriptions and/or other types of information associated with it such as the title, abstract, subject, geographic location, release or creation date of video, name of organization that produced the video, duration

of the video, video director, actors, actresses, Golden Globe and Academy Award nominations and wins. It should be possible to index the text descriptions of the video concurrently with other numerical keys (e.g. date, duration, or geographical location). The specific objectives are stated as follows:

- investigate the k-d tree data structure to store and index MPEG-2 video data.
- determine which keys are appropriate to use for indexing MPEG-2 video data.
- experimentally investigate content-based indexing using colour histogram based querying such as "find MPEG-2 video I-frames with $> t$ % pixels in color range $[R_L, R_H]$".
- experimentally investigate content-based indexing using color space based querying such as "find MPEG-2 video I-frames with $> t$ % pixels in color range $[R_L, R_H]$, $[G_L, G_H]$, $[B_L, B_H]$, simultaneously ".

1.4 Thesis Overview

The thesis is organized as follows:

Chapter 2 introduces a content-based video indexing method for MPEG-2 video data. It gives an introduction on MPEG-2 video compression standard. Colour histogram indexing and colour space indexing methods for that kind of data as well as their range queries are also presented.

Chapter 3 analyses and illustrates the experimental results. The methods of sampling and pre-processing data are developed and illustrated in detail. Index construction, data structure implementations and range search results are also discussed

in this chapter.

Chapter 4 summarizes the conclusions of the thesis and future work.

Chapter 2 CONTENT-BASED VIDEO INDEXING

2.1 MPEG-2 Video

The Moving Picture Experts Group's MPEG-2 video standards are the most recently adopted and internationally accepted compression standards for digital video. They provide low-to-medium bit rate, high-quality video and include standards for compressing, multiplexing, and reconstructing video streams. MPEG-2 is currently used in digital video service roll-outs worldwide and is the compression standard used in the majority of digital broadcast satellite systems.

2.1.1 Birth of MPEG-2 Video

MPEG (**M**oving **P**icture **E**xperts **G**roup) was started in 1988 as a working group within ISO/IEC (the **I**nternational **O**rganization for **S**tandardization and the **I**nternational **E**lectro-**T**echnical **C**ommission) with the aim of defining standards for digital compression of audio-visual signals. MPEG's first project, MPEG-1, was published in 1993 as ISO/IEC 11172 [ISO93]. It is a three-part standard defining audio and video compression coding methods and a multiplexing system for interleaving audio and video data so that they can be played back together. MPEG-1 principally supports video coding up to about 1.5 Mbit/s giving quality similar to VHS (**V**ideo **H**ome **S**ystem). It is used in the CD-I (**C**ompact **D**isk-**I**nteractive) and Video-CD systems for storing video and audio on CD-ROMs.

During 1990, MPEG recognised the need for a second, related standard for coding video for broadcast formats at higher data rates. The MPEG-2 standard [ISO94] is capable of coding standard-definition television at bit rates from about 3-15 Mbit/s and high-definition television at 15-30 Mbit/s. MPEG-2 is an extension of the MPEG-1 digital video compression standard. MPEG-2 is directed at broadcast formats at higher data rates; it provides extra algorithmic "tools" for efficiently coding interlaced video, supports a wide range of bit rates and provides for multichannel surround sound coding. MPEG-2 decoders will also decode MPEG-1 bit streams. Table 2.1 summarizes MPEG-2 compression capabilities for different types of audio and video data.

Table 2.1 MPEG-2 compression capabilities [from Hask97].

	Video Resolution (pels x lines x frames/s)	Uncompressed Bit rate (RGB)	Compressed Bit rate
Film (USA and Japan)	(480 x 480 x 24Hz)	133 Mbits/s	3 to 6 Mbits/s
NTSC video	(480 x 480 x 29.97Hz)	168 Mbits/s	4 to 8 Mbits/s
PAL video	(576 x 576 x 25Hz)	199 Mbits/s	4 to 9 Mbits/s
HDTV video	(1920 x 1082 x 30Hz)	1493 Mbits/s	18 to 30 Mbits/s
HDTV video	(1028 x 720 x 60Hz)	1327 Mbits/s	18 to 30 Mbits/s
ISDN videophone (CIF)	(325 x 288 x 29.97Hz)	73 Mbits/s	64 to 1920 kbits/s
PSTN videophone (QCIF)	(176 x 144 x 29.97Hz)	18 Mbits/s	10 to 30 kbits/s
Two-channel stereo audio		1.4 Mbits/s	128 to 384 kbits/s
Five-channel stereo audio		3.5 Mbits/s	384 to 968 kbits/s

As stated by Gringeri et al [Grin98], the underlying philosophy of MPEG-2 video compression is lossy coding followed by the assumption of lossless transmission. That is, the algorithms used for reconstruction of MPEG-2-based moving images at the decoder assume lossless and constant delay transmission over the network models.

2.1.2 Video Fundamentals

Imaging can be of various types, such as normal photography, X-rays, electronic documents, electronic still pictures, motion pictures, and TV. Video can be thought of as comprised of a sequence of still pictures of a scene taken at various subsequent intervals in time. Each still picture represents the distribution of light energy and wavelengths over a finite size area and is expected to be seen by a human viewer.

Television services in North America currently broadcast NTSC (National Television Standards Committee) video at a frame rate of 30 Hz, in accordance with electrical power here. Each frame consists of two interlaced fields, giving a field rate of 60 Hz. The first field of each frame contains only the odd numbered lines of the frame (numbering the top frame line as line 1). The second field contains only the even numbered lines of the frame and is sampled in the video camera 16.67 ms after the first field. It is important to note that one interlaced frame contains two fields. PAL (Phase Alternate Line) television transmission is similarly interlaced but with a frame rate of just under 25 Hz.

In video systems other than television, non-interlaced video is commonplace (for example, most computers output non-interlaced video). In non-interlaced video, all the lines of a frame are shown from top to bottom in one field. Non-interlaced video is also termed "progressively scanned" or "sequentially scanned" video.

If interlaced TV displays are used with computers, the result is interline flicker, line crawling, and other problems. To avoid these problems, computer displays use non-interlaced (also called progressive or sequential) displays with refresh rates of higher than 60 frames per seconds (fps), typically 70 to 75 fps.

2.1.3 Bit Rate Reduction Principles

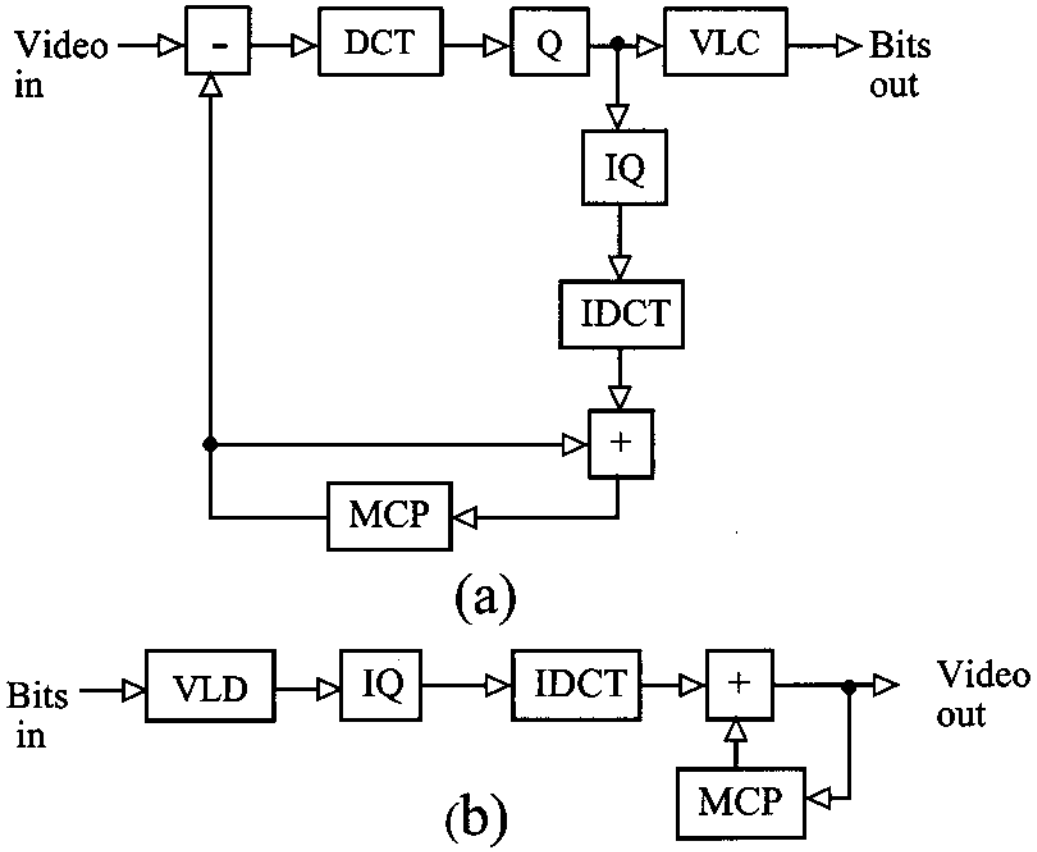
In a lossless environment, a bit rate reduction system operates by removing redundant information from the signal at the encoder prior to transmission and re-inserting it at the decoder. An encoder and decoder pair are referred to as a "codec".

In video signals, two distinct kinds of redundancy can be identified. The first is spatial and temporal redundancy. Pixel values are not independent, but are correlated with their neighbours both within the same frame and across frames. So, to some extent, the value of a pixel is predictable given the values of neighbouring pixels. The second is psycho-visual redundancy. The human eye has a limited response to fine spatial detail, and is less sensitive to detail near object edges, around scene-changes, and in bright areas (among others). Consequently, controlled impairments introduced into the decoded picture by the bit rate reduction process should not be visible to a human observer.

Two key techniques employed in an MPEG codec are **D**iscrete **C**osine **T**ransform (DCT) coding and motion-compensated interframe prediction. Using the DCT, spatial redundancy within a frame can be used. Temporal redundancy can be used due to the fact that successive frames are often almost identical. The difference between two successive frames is coded using motion-compensated interframe prediction.

Codec structure

Figure 2.1 shows the basic architecture of a typical MPEG-2 encoder and decoder. In an MPEG-2 system, the DCT and motion-compensated interframe prediction are combined. The encoder subtracts the motion-compensated prediction from the source picture to form a "prediction error" picture. The prediction error is transformed with the DCT, the coefficients are quantized and these quantized values are coded using a **V**ariable **L**ength-**C**oder (VLC) to form a bit-stream for transmission. In the decoder, the quantized DCT coefficients are reconstructed and inverse transformed to produce the prediction error. This is added to the motion-compensated prediction generated from previously decoded pictures to produce the decoded output. The exact detail of this process are not germane to our research; Haskell et al [Hask97] contains more detail for interested readers.



(I)DCT = (inverse) discrete cosine transform

(I)Q = (inverse) quantization

MCP = motion-compensated prediction

VLC = variable length-coder

VLD = variable length-decoder

Figure 2.1 (a) A typical MPEG-2 encoder. (b) A typical MPEG-2 decoder (adapted from Haskell et al [Hask97]).

Intra-frame DCT coding

A two-dimensional **D**iscrete **C**osine **T**ransform (DCT) is performed on small blocks (8 pixels by 8 lines) of each component of the picture to produce blocks of DCT coefficients. The magnitude of each DCT coefficient indicates the contribution of a particular combination of horizontal and vertical spatial frequencies to the original picture block. The coefficient corresponding to zero horizontal and vertical frequency is called the DC coefficient.

The DCT does not directly reduce the number of bits required to represent the block. The reduction in the number of bits follows from the observation that, for typical blocks from natural images, the distribution of coefficients is non-uniform. The transform tends to concentrate the energy into the low-frequency coefficients and many of the other coefficients are near-zero. The bit rate reduction is achieved by not transmitting the near-zero coefficients and by quantizing and coding the remaining coefficients. The non-uniform coefficient distribution is a result of the spatial redundancy present in the original image block.

The serialisation and coding of the quantized DCT coefficients exploits the likely clustering of energy into the low-frequency coefficients and the frequent occurrence of zero-value coefficients. The block is scanned in a diagonal zigzag pattern starting at the DC coefficient to produce a list of quantized coefficient values, ordered according to the scan pattern.

2.1.4 Frame Types

Three "frame types" are defined in MPEG -2. The frame type defines which prediction mode is used to code each block. Each video sequence is divided into one or more groups of pictures, and each group of pictures is composed of one or more pictures of three different types, I-, P-, and B-. Figure 2.2 illustrates the types of frames occurring in an MPEG-2 encoding of a time sequence of 13 video frames.

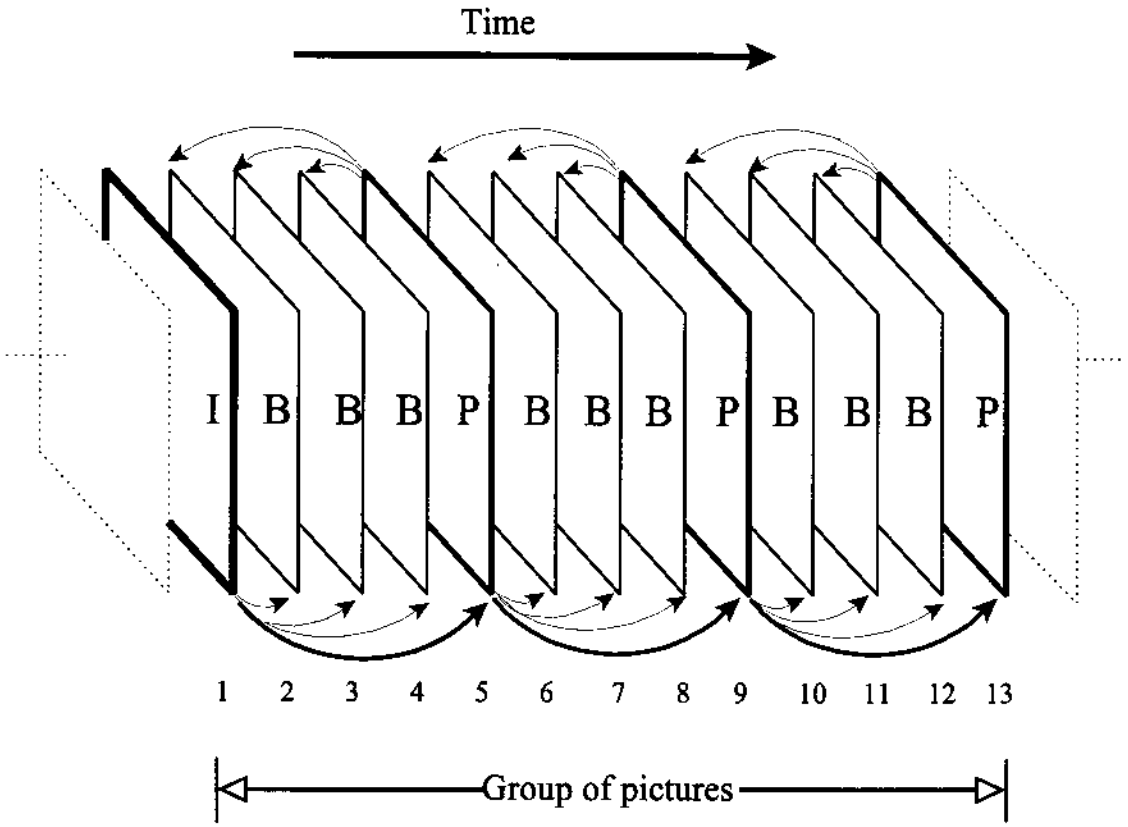


Figure 2.2 A typical group of pictures in display order [from Mitc97].

* "Intra" frames (I-frames) are coded without reference to other frames. Moderate compression is achieved by reducing spatial redundancy, but not temporal redundancy. They are used periodically to provide access points in the bit stream where decoding can begin.

I-frames must appear regularly in the stream, since they are needed to decode subsequent inter-coded frames such as P-frame and B-frames. The decoding process cannot begin until an I-frame is received. An I-frame is usually inserted into a stream approximately every half second [Grin98].

* "Predictive" frames (P-frames) can use the previous I- or P-picture for motion compensation and may be used as a reference for further prediction. Each block in a P-frame can either be predicted or intra-coded. By reducing spatial and temporal redundancy, P-frames offer increased compression compared to I-frames.

* "Bidirectionally-predictive" frames (B-frames) can use the previous and next I- or P-frames for motion-compensation, and offer the highest degree of compression. Each block in a B-frame can be forward, backward or bidirectionally predicted or intra-coded. To enable backward prediction from a future frame, the coder reorders the pictures from natural "display" order to "bit stream" order so that the B-frame is transmitted after the previous and next pictures it references. This introduces a reordering delay dependent on the number of consecutive B-frames.

The different picture types typically occur in a repeating sequence, termed a "Group of Pictures" or GOP. Since MPEG sometimes uses information from future pictures in the sequence, the coding order, the order in which compressed pictures are

found in the bit-stream, is not the same as the display order, the order in which pictures are presented to a viewer. The coding order is the order in which the pictures are received by the decoder. The group of pictures illustrated in Figure 2.2 in display order is shown in coding order in Figure 2.3.

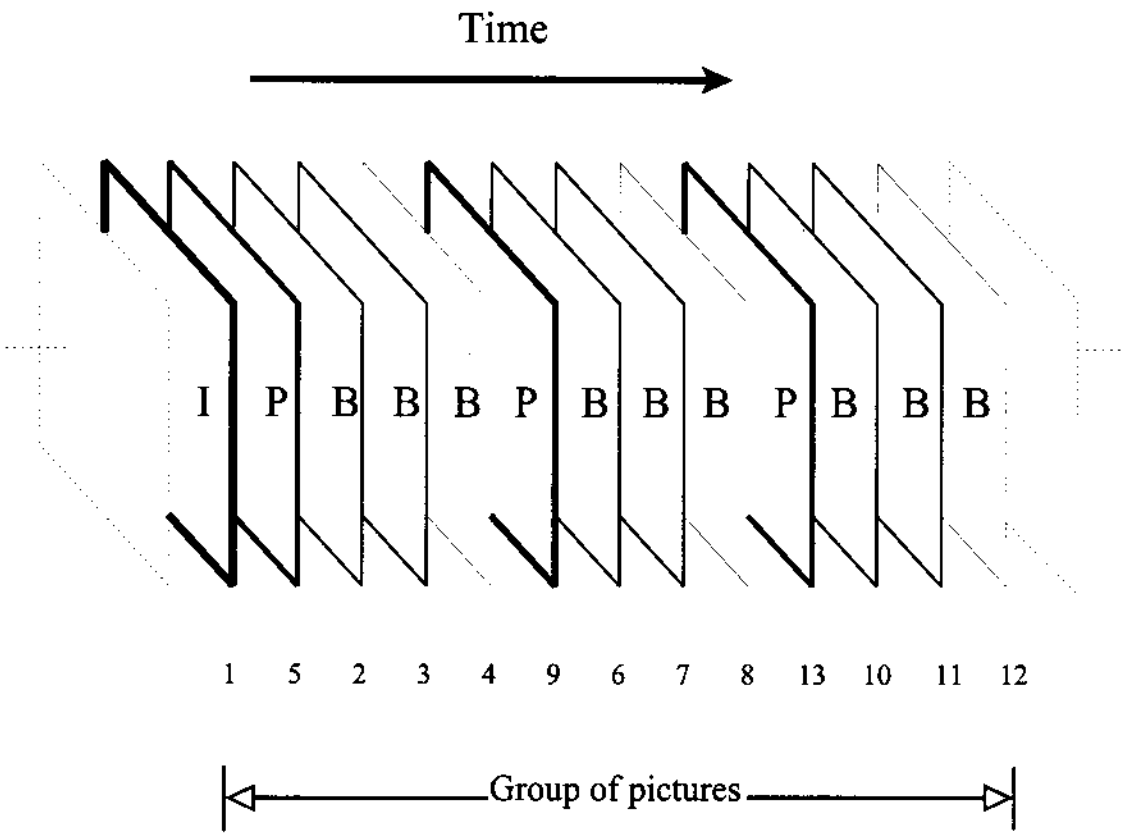


Figure 2.3 A typical group of pictures in coding order [from Mitc97].

2.2 Colour Histogram Indexing

So far, indexing images by global colour distribution has been achieved by using colour histograms. This technique provides a good approach for retrieving images that have similar overall colour content.

2.2.1 Colour Fundamentals

One of the initial studies of colour was done by Sir Isaac Newton in the eighteenth century. From his research, Newton concluded that seven colours were needed to represent all the combinations of visual colours. It was Thomas Young, James Forbes, and James Clerk Maxwell in the nineteenth century who showed that only three primary colour are needed in different combinations to represent the visible spectrum of light [Weeks96].

Basically, the colours that human beings perceive in an object are determined by the nature of the light reflected from the object. For example, green objects reflect light with wavelengths primarily in the 500 to 570 nm (10^{-9} m) range, while absorbing most of the energy at other wavelengths.

Chromatic light spans the electromagnetic energy spectrum from approximately 400 to 700 nm. Three basic quantities are used to describe the quality of a chromatic light source: radiance, luminance, and brightness. Radiance is the total amount of energy that flows from the light source. Luminance gives a measure of the amount of energy an observer perceives from a light source. Brightness embodies the achromatic notion of intensity.

Owing to the structure of the human eye, all colours are seen as variable combinations of the three so-called primary colours red (R), green (G), and blue (B). For the purpose of standardization, the CIE (Commission International de l'Eclairage) designated in 1931 the following specific wavelength values to the three primary colours: blue = 435.8 nm, green = 546.1 nm, and red = 700 nm [Gon92].

The characteristics generally used to distinguish one colour from another are brightness, hue, and saturation. As already indicated, brightness embodies the chromatic notion of intensity. Hue is an attribute associated with the dominant wavelength in a mixture of light waves. Thus hue represents the dominant colour as perceived by an observer. Saturation refers to relative purity or the amount of white light mixed with a hue. Hue and saturation taken together are called chromaticity, and therefore, a colour can be characterized by its brightness and chromaticity.

2.2.2 Colour Models

Colour is probably one of the most intuitive characteristics that people use to recognise an image. Since the perception of colour is influenced by many different factors (e.g. surrounding colours, overall lighting conditions, or a user's point of view), it becomes an arduous task to determine these factors automatically.

The purpose of a colour model is to facilitate the specification of colour in some standard. In essence, a colour model is a specification of a 3-D coordinate system and a subspace within that system where each colour is represented by a single point.

Most colour models in use today are oriented either toward hardware (such as for

colour monitors and printers) or toward applications where colour manipulation is a goal (such as in the creation of colour graphics for animation). The hardware-oriented models most commonly used in practice are the RGB (red, green, blue) model for colour monitors and a broad class of colour video cameras; the CMY (cyan, magenta, yellow) model for colour printers; and the YIQ model, which is the standard for colour TV broadcasts. In the third model the Y corresponds to luminance, and I and Q are two chromatic components called inphase, and quadrature, respectively. Among the models frequently used for colour image manipulation are the HSI (hue, saturation, intensity) model and the HSV (hue, saturation, value) model.

The RGB colour model

The simplest of the colour models is the RGB colour model. This model uses the three NTSC primary colours to describe a colour within a colour cube. Each colour component represents an orthogonal axis in a three-dimensional Euclidean space as shown in Figure 2.4. On computers it is common to describe colour by three components; normally red, green, and blue. These are related to the excitation of red, green, and blue phosphors on a computer monitor.

The RGB colour model treats a colour image as a set of three grayscale images, each of which represents one of the red, green, and blue components of a colour image. Characterization by colour can be performed by using the red, green, and blue components (RGB), as each colour pixel is a combination of the three components.

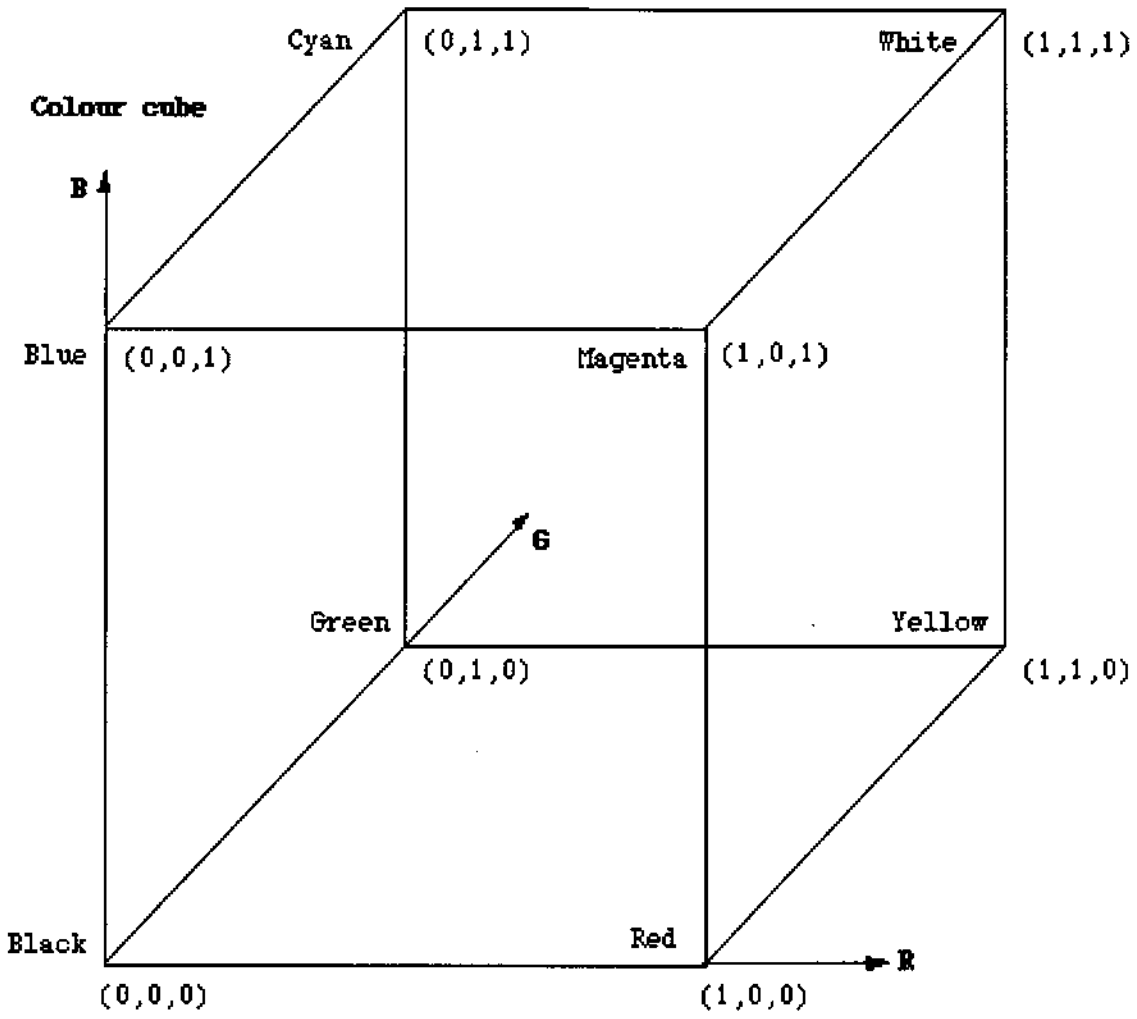


Figure 2.4 The RGB colour model.

Typically, 256 gray levels are used to represent a grayscale image to meet the computer storage requirement of one byte per pixel of storage. RGB colour images need one byte per pixel for each of the colour planes, or three times the storage of a grayscale image of the same spatial dimensions. Each pixel in a colour image requires 3 bytes or 24 bits to represent all of the possible colours. Colour images of this type are now standard

and are often referred to as 24-bit or true-colour images.

The YIQ colour model

The YIQ model is used in commercial colour television broadcasting. Basically, YIQ is a recording of RGB for transmission efficiency and for downward compatibility with black-and white television. The recorded signal is transmitted using the NTSC system[Prit77]. The YIQ model uses a 3-D Cartesian coordinate system, with the visible subset being a convex polyhedron that maps into the RGB cube.

The RGB-to-YIQ mapping is defined as follows [Gon92]:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2.1}$$

2.2.3 Colour Histogram

A semantic representation for colour is the use of a colour histogram that captures the colour composition of images. Using the RGB colour space, the histogram comprises a set of "bins" each representing a colour that is obtained by a range of red, blue, and green values. The number of pixels of an image falling into each of these bins can be obtained by counting the pixels with the corresponding colour.

The popularity of colour histograms stems from several factors, including

(a) Colour histograms are computationally simple to evaluate,

(b) Small changes in camera viewpoint tend not to effect colour histograms, and

(c) Different objects often have distinctive colour histograms.

We chose to use the RGB model for colour in this thesis due to the ready availability of data in this format. The data structures and algorithms developed here can also be applied directly to other colour models.

The normalized histogram of an N x M image is defined as the fraction of pixels within the image at a given intensity; i.e.

$$h_i = \frac{n_i}{MN} \quad for \quad 0 \le i \le G_{max} \tag{2.2}$$

where h_i = the fraction of pixels in the image with intensity i, n_i is the number of pixels at i, NM is the total number of pixels within the image and G_{max} is the maximum intensity value of the image. For an image which has 256 intensity values, $G_{max} = 255$. An important property of a normalized histogram is that the sum of each histogram value over the range of intensity present within an image equals one; i.e.

$$\sum_{i=0}^{G_{max}} h_i = 1 \tag{2.3}$$

Figure 2.5 shows a sample MPEG-2 I-frame, and Figures 2.6, 2.7, and 2.8 show the corresponding normalized histograms of this sample I-frame over the range of red, green, and blue values, respectively.

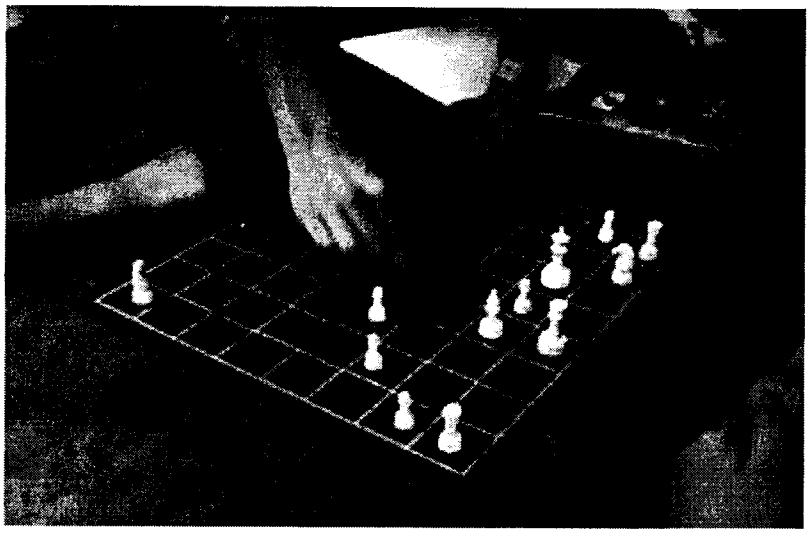


Figure 2.5 Sample MPEG-2 I-frame.

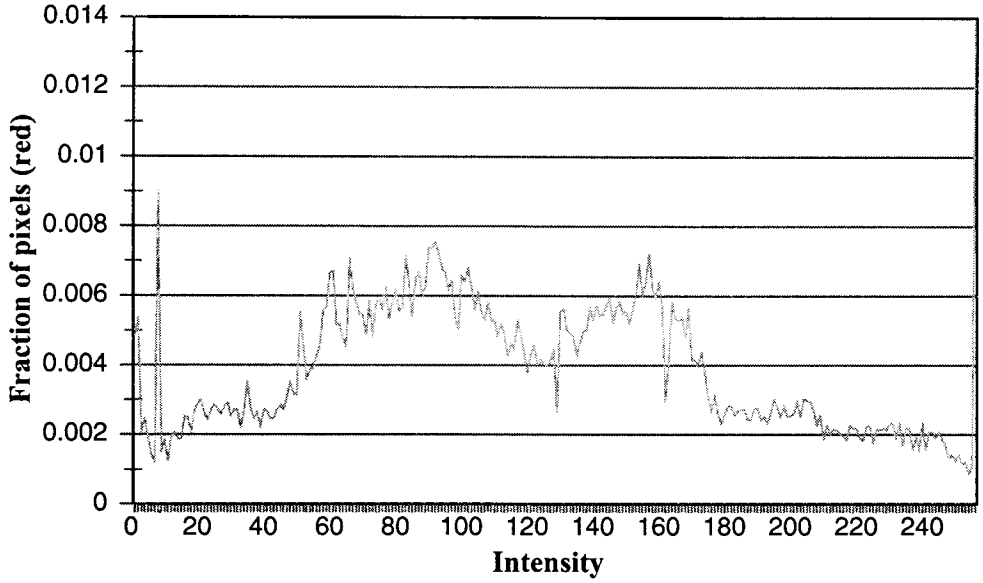


Figure 2.6 Normalized histogram of a sample MPEG-2 I-frame over the range of red values.

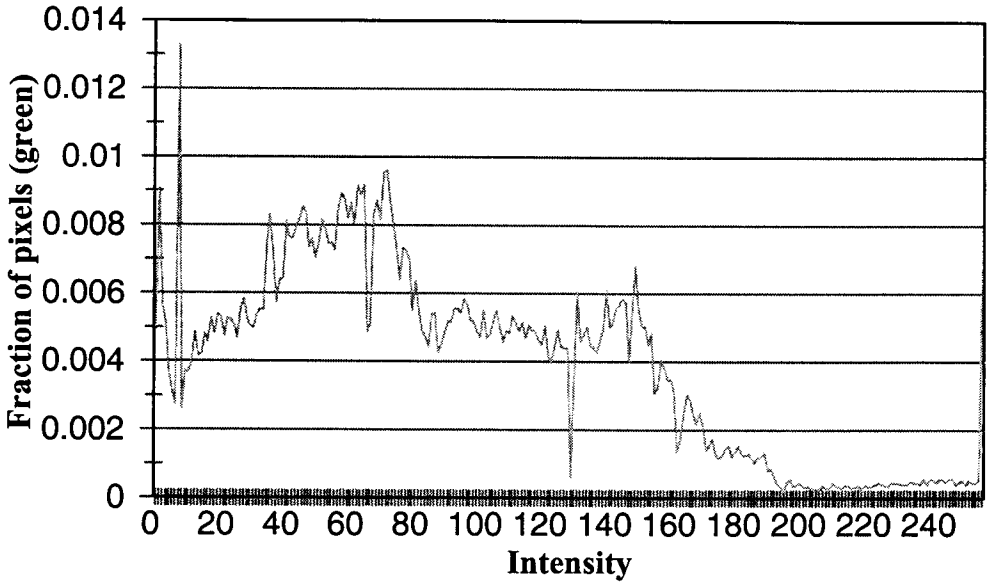


Figure 2.7 Normalized histogram of a sample MPEG-2 I-frame over the range of green values.

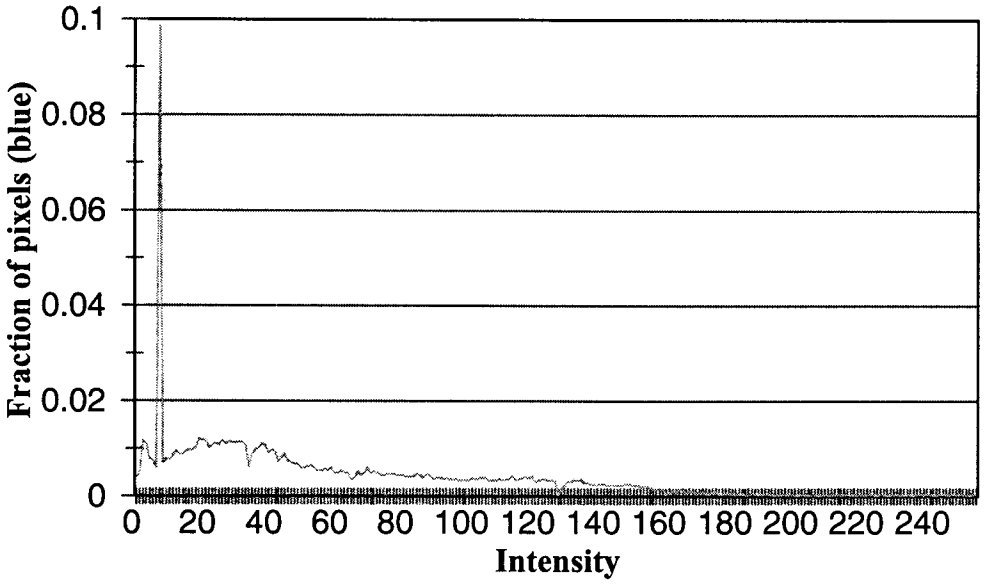


Figure 2.8 Normalized histogram of a sample MPEG-2 I-frame over the range of blue values.

2.2.4 Range Search

We consider a MPEG-2 I-frame as three different images, one for each of the colour components. That is, we separate the three colour components and treat them individually. We then build histograms for each of them. The histograms are used as the keys to index digital video data stored in a k-d tree spatial data structure which can easily handle range queries.

2.2.4.1 K-D Tree

In the term k-d tree, k denotes the dimensionality of the space being represented. In principle, it is a binary search tree with the distinction that, at each depth, a different attribute (or key) value is tested when determining the direction to branch. In two dimensions (i.e., a 2-d tree), we compare x coordinate values at the root and at even depths (assuming that the root is at depth 0) and y coordinate values at odd depths.

A k-d tree can be built by recursively subdividing the space in one dimension at each level in the tree. For example, a 3-d tree in three dimensions can be created by alternatively using the x-coordinate , y-coordinate and z-coordinate as discriminators at different depths of the trees. Figure 2.9 shows a 3-d tree data structure containing nine 3-d points.

First, point (32, 45, 4) is inserted into the 3-d tree as the root node. When the second point (29, 25, 3) is inserted into the 3-d tree, its x-coordinate value 29 is compared with the x-coordinate value 32 of the root node at the depth 0. Since 29 is less than 32, the point (29, 25, 3) is inserted into the 3-d tree as the left child of the root node.

Similarly, point (51, 47, 1) is the right child of the root.

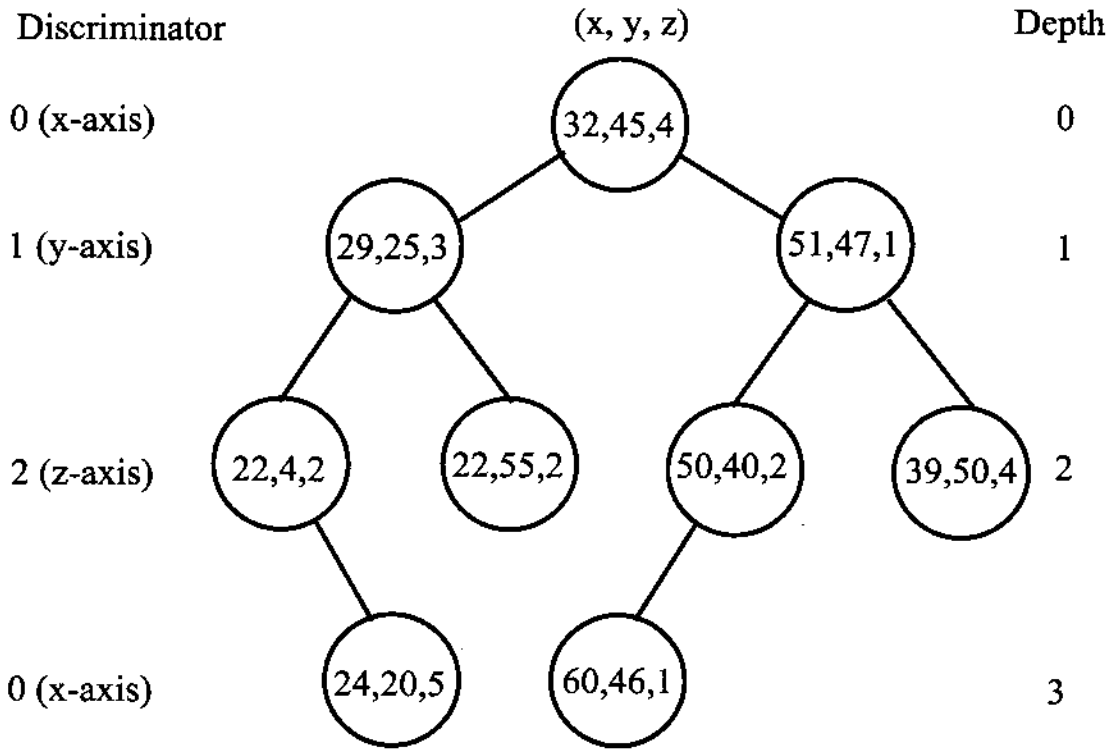


Figure 2.9 A 3-d tree data structure.

At depth 1, assuming that the points are inserted in the order (22, 4, 2), (22, 55, 2), (50, 40 ,2), and (39, 50, 4), the y-coordinate values of the points are tested at depth 1 to determine the direction to branch. For example, point (22, 4, 2) has x-coordinate value 22, which is less than 32 (the root node x value), so we branch left. Again, 4 is less than 25, the y value for the left child of the root, so we branch left again, and insert point (22, 4, 2) as the left child of point (29, 25, 3).

At depth 2, assuming that the points are inserted in the order (24, 20, 5), and (60, 46, 1), the z-coordinate values of the points are tested at depth 2 to determine the direction to branch. For example, point (24, 20, 5) has x-coordinate value 24, which is less than 32 (the root node x value), so we branch left. Again, 20 is less than 25, the y value for the left child of the root, so we branch left again. The z-coordinate 5 is more than 2, the z value of the left child of the node containing point (29, 25, 3), so we branch right and insert point (24, 20, 5) as the right child of the node containing point (22, 4, 2).

A range query with respect to a k-d tree rooted at T, for example, to a 2-d tree defines a circle of radius R centred at location (X = A, Y = B), and expects to find all points in the 2-d tree that lie within the circle. Range search for all keys within ± R of X = A, Y = B in the 2-d tree rooted at T is performed as shown in the Figure 2.10.

```
KdSearch (A, B, R, T)
{
      if T != NULL
      {
            P = NEW KdTreeNode(A-R, B-R);
            if (KdCompare (P, T) == Left)
                  KdSearch(A, B, R, T.Left);
            ToRight = 0;
            ToLeft = 0;

            if (T.X ≥A-R && T.Y ≥ B-R)
                  ToRight = 1;
            if (T.X ≤A+R && T.Y ≤ B+R)
                  ToLeft = 1;
            if (ToRight && ToLeft)
                  Report the point is in range;
            P.X = A+R;
            P.Y = B+R;
            if (KdCompare (P, T) == Right)
                  KdSearch(A, B, R, T.Right);
      }
}
```

Figure 2.10 2-d range search algorithm for a 2-d tree.

The performance of the k-d tree is as follows: the preprocessing requires O(NlogN) time, the storage is O(kN), and the time required for a range search is $O(kN^{1-1/k})$ with a complete binary k-d tree of N points [Samet 1990a].

2.2.4.2 Types of Histogram Range Queries

When designing indexing schemes to store MPEG-2 video data, we need to take into account the kinds of queries that will be asked of such a system. We built a histogram index based on the type of user query. Some examples of range queries to be used are illustrated in Figure 2.11.

1. Find MPEG-2 video I-frames with > t % pixels in colour range $[R_L, R_H]$.
2. Find MPEG-2 video I-frames with > t % pixels in colour range $[G_L, G_H]$.
3. Find MPEG-2 video I-frames with > t % pixels in colour range $[B_L, B_H]$.
4. Find MPEG-2 video I-frames with < t % pixels in colour range $[R_L, R_H]$.
5. Find MPEG-2 video I-frames with < t % pixels in colour range $[G_L, G_H]$.
6. Find MPEG-2 video I-frames with < t % pixels in colour range $[B_L, B_H]$.
7. Find MPEG-2 video I-frames with between t_1 % pixels and t_2 % pixels in colour range $[R_L, R_H]$.
8. Find MPEG-2 video I-frames with between t_1 % pixels and t_2 % pixels in colour range $[G_L, G_H]$.
9. Find MPEG-2 video I-frames with between t_1 % pixels and t_2 % pixels in colour range $[B_L, B_H]$.
10. Find MPEG-2 video I-frames with R-intensity whose fraction > t % of the total.
11. Find MPEG-2 video I-frames with G-intensity whose fraction > t % of the total.
12. Find MPEG-2 video I-frames with B-intensity whose fraction > t % of the total.
13. Similarly for fraction < t % of the total.
14. Find MPEG-2 video I-frames with R-intensity whose fraction between t_1 % and t_2 % of the total.
15. Similarly for Green (G) and Blue (B) colours.

Figure 2.11 Example range queries on MPEG-2 video I-frame histograms.

Examples of where these queries might be useful are to detect "blue screen" frames (e.g. 95% of pixels are in range [$B_L = 220$, $B_H = 255$]), or to detect I-frames with images primarily of sky or water, or to detect a green colour for grass.

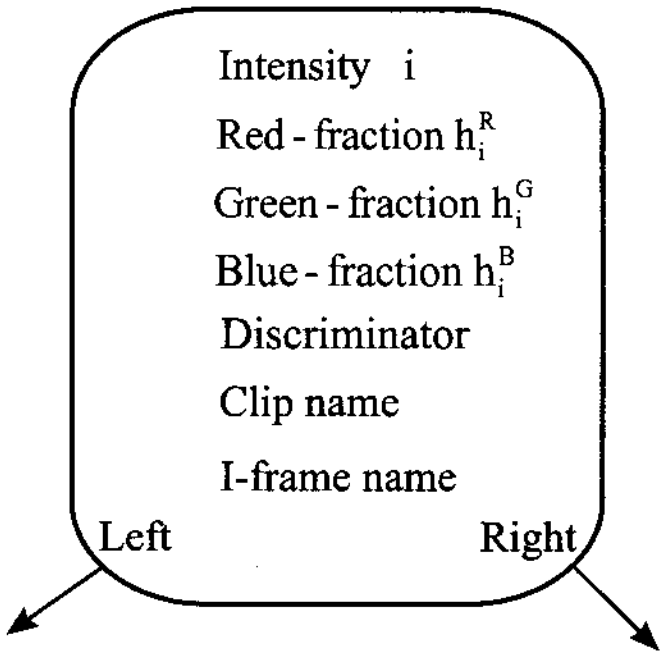
2.2.5 Data Structures

To satisfy the queries listed above, we used a combination of k-d tree and hash table data structures. The details about these data structures are discussed in the following sections.

2.2.5.1 Node Structure

After sampling and pre-processing the MPEG-2 video clip I-frame data (which constructs R, G, and B histograms by three attributes red, green, and blue (see chapter 3 for details)), a 4-d tree node is defined for indexing video data as shown in Figure 2.12. Each node contains four keys and is conveniently processed as a point in a four-dimensional key-space. The dimensions of the key-space or search are those of value of given intensity i which is from 0 to 255, red colour histogram (fraction of total pixels h_i^R) at intensity i, green colour histogram value h_i^G at intensity i, and blue colour histogram value h_i^B for each of the MPEG-2 video clip I-frames. From equation (2.2), all h_i values are between 0.0 and 1.0. Each node also contains two non-key attributes corresponding to the MPEG-2 video clip name and its I-frame name. The names show the actual locations of the MPEG-2 video clip and I-frame in the file system. Right and Left

in each node are pointers to the node's children.




```
struct KdTreeNode
{
      int   i;    // The value of intensity (0 to 255)

      // The data in the node; (r, g, b fraction)
      double R-frac, G-frac, B-frac;

      DType   Disc;         // Discriminatior
      String  clip;         // The video clip name
      String  I-frame;      // The I-frame name

      KdTreeNode* Left;     // Left child
      KdTreeNode* Right;    // Right child
}
```

Figure 2.12 Node structure for a 4-d tree.

2.2.5.2 Construction of a 4-d Tree

To get a balanced 4-d tree(which means that left and right subtrees of every node in this tree have the same height), the tree was balanced on intensity. That means the tree was constructed by inserting intensities in the following order: 127, 63, 31, 15, 7, 3, 1, 0, 2, 5, 4, 6, 11, 9, 8, 10, 13, 12, 14, ... , 247, 243, 241, 240, 242, 245, 244, 246, 251, 249, 248, 250, 253, 252, 254, 255. This order corresponds to an pre-order traversal of a complete binary tree of 256 nodes. Figure 2.13 illustrates part of the corresponding tree.

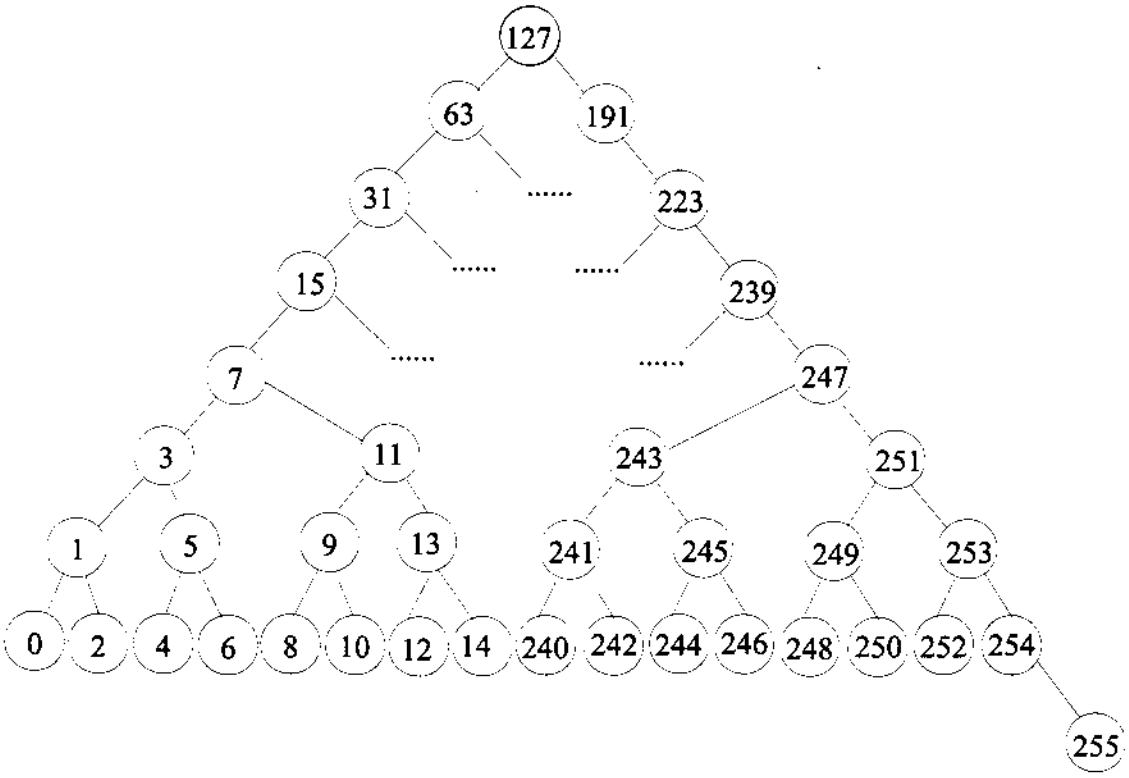


Figure 2.13 Part of a complete binary tree of 256 nodes.

The other keys were inserted in the order which they were computed. Thus, starting at the root level 0, the successive discriminators are 0, 1, 2, 3,... corresponding to intensity i, red colour histogram h_i^R, green colour histogram h_i^G, and blue colour histogram h_i^B, respectively. These correspond to tree depths (0, 1, 2, 3), (4, 5, 6, 7), (8, 9, 10, 11), etc.

The 4-d data points, each of them with corresponding attributes (i, h_i^R, h_i^G, h_i^B), are inserted into 4-d tree in a manner analogous to a binary search tree. In essence, we search for the desired point based on the values of i, h_i^R, h_i^G, and h_i^B, comparing i values at (0, 4, 8 ...) depths of the tree, h_i^R values at (1, 5, 9...) depths, h_i^G values at (2, 6, 10...) depths, and h_i^B values at (3, 7, 11...) depths. The Insert algorithm is shown in Figure 2.15. It makes use of the KdCompare algorithm (Figure 2.14) to determine which direction to branch. When we reach the bottom of the tree (i.e., when a NULL pointer is encountered), we find the location where the point is to be inserted.

```
// Return the child type of the 4-d tree rooted
          // at node Q in which node P belongs.
          KdCompare(KdTreeNode P, KdTreeNode Q)
          {
               if (Q.Discriminator  == i)
               {
                    if (P.i < Q.i) return Left;
                    else return Right;
               }
               else if (Q.Discriminator == red )
               {
                    if (P. h_i^R < Q. h_i^R ) return Left;
                    else return Right;
               }
               else if (Q.Discriminator == green)
               {
                    if (P. h_i^G < Q. h_i^G ) return Left;
                    else return Right;
               }
               else
               {
                    if (P. h_i^B < Q. h_i^B ) return Left;
                    else return Right;
               }
          }
```

Figure 2.14 The algorithm used for 4-d tree node comparison to determine branching direction.

```
// Add P into subtree rooted at T
KdInsert(KdTreeNode P, KdTreeNode T)
{
      if (T == NULL)
      {
            T = P;
            T.Discriminator = i;
            return;
      }

      KdTreeNode  F,
      SaveRoot = T;

      while (T != NULL &&
                        (P.Coordinates(i, h_i^R , h_i^G , h_i^B ) !=
                         T.Coordinates(i, h_i^R , h_i^G , h_i^B )))
      {
            F = T;  // Remember parent of T
            Direction  = KdCompare(P, T);
            if (Direction == Left)
                  T = T.Left;
            else
                  T = T.Right;
      }

      if (T == NULL) // P is not already in tree.
      {
            if (Direction == Left )
                  F.Left = P;
            else
                  F.Right = P;

            if (F.Discriminator == i)
                  P.Discriminator = red;
            else if (F.Discriminator == red)
                  P.Discriminator = green;
            else if (F.Discriminator == green)
                  P.Discriminator = blue;
            else
                  P.Discriminator = i;
      }
      T = SaveRoot;
}
```

Figure 2.15 K-d Insert algorithm for colour histogram data.

2.2.5.3 Hash Table Structure for Accumulating Search Results

A range search with respect to a 4-d tree will follow the algorithm given in Figure 2.10. To query the histogram of a specific MPEG-2 video I-frame in a given intensity range, such as $[R_L, R_H]$, and to accumulate the search results, a hash table data structure is employed. When the range search method finds a 4-d tree node is in range, this node is inserted into the hash table. This will apply to all nodes that are in range.

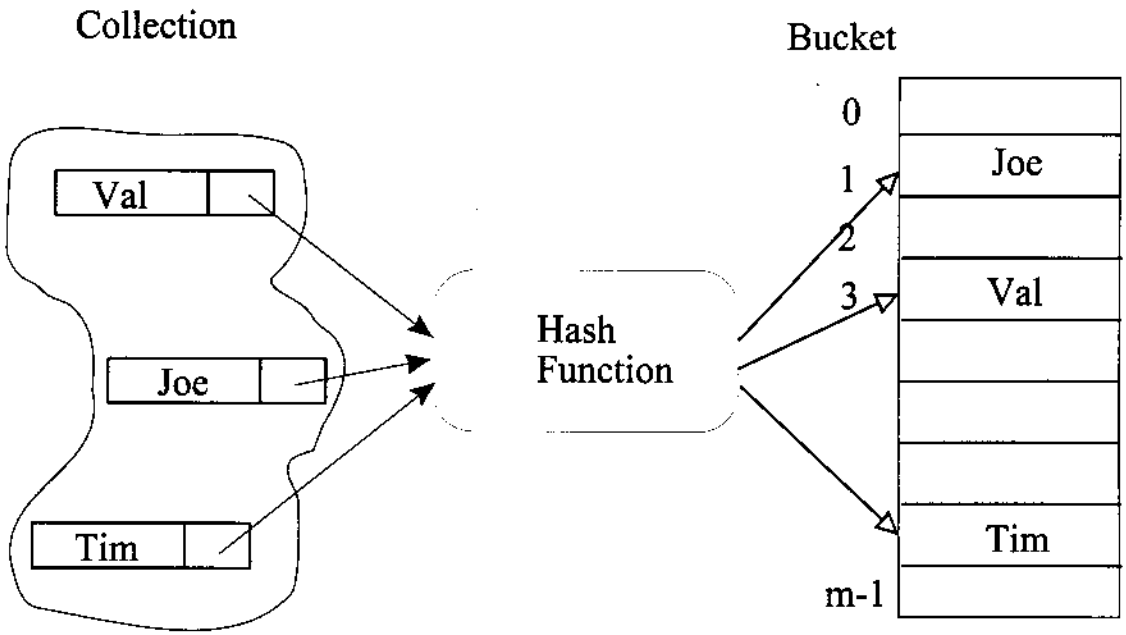


Figure 2.16 Insertion of a character string key into a hash table with m buckets.

In the hash strategy, data elements are kept in an array-based data structure, called the hash table. The hash function converts the key into an integer suitable to index an array where the key is stored. A data element may be simple or complex, but must be identified by a single key, which is used for searching. Hashing is a technique used to perform insertions, removes, and finds in constant average time. Figure 2.16 illustrates the insert operation for a character string key into a hash table of size m.

We use the MPEG-2 video clip name and I-frame name concatenated together as the key to produce the bucket through the hash function. The hash function used is shown in Figure 2.17.

```
unsigned int Hash (Key, TableSize)
{
      HashValue = 0;
      for (i = 0; i < Key.Length( ); i++)
      {
            HashValue = (HashValue << 5) ^ Key[i] ^ HashValue;
            // << Left Shift operator, ^ XOR operator
      }
      return HashValue % TableSize;  // % the mod operator
}
```

Figure 2.17 Hash function used for producing the hash value.

For example, with MPEG-2 video clip name = ~/mpeg/wf174.mpg, I-frame name = ~/I_frames/wf174/rec174.tga, giving a key of "~/mpeg/wf174.mpg ~/I_frames/wf174/rec174.tga", and TableSize = 211, the hash function returns bucket 22.

When searching, the updated total fraction for red, green, and blue is calculated and stored in the same bucket for hits on the same MPEG-2 video clip name and I-frame name. Figure 2.18 shows the histogram range search algorithm for range queries on MPEG-2 video. The algorithm can satisfy the range queries given in Figure 2.11. The colour range [i_low, i_high] can be [R_L, R_H], [G_L, G_H], or [B_L, B_H], respectively. Figure 2.19 illustrates the range search process.

```
// range search method for the 4-d tree
// T points to the 4-d tree root
KdSearch (i_low, i_high,  KdTreeNode T)
{
      if (T != NULL)
      {
            P = new KdTreeNode (i_low);
            if (KdCompare( P, T) == Left)
                  KdSearch (i_low, i_high, T.Left);

            ToRight = 0; ToLeft = 0;
            if (T.i >= i_low )
                  ToRight = 1;
            if (T.i <= i_high)
                  ToLeft = 1;

            if (ToRight && ToLeft)
            {
                  InsertHash (clip_name, I_frame_name, h_i^R , h_i^G , h_i^B );
                  j = Hash (clip_name + I_frame_name);
                  Hash_Array[j].R_fract += h_i^R ;
                  Hash_Array[j].G_fract += h_i^G ;
                  Hash_Array[j].B_fract += h_i^B ;
            }

            P.i = i_high;
            if (KdCompare( P, T ) == Right)
                  KdSearch(i_low, i_high, T.Right);
      }
```

Figure 2.18 The histogram range search algorithm.

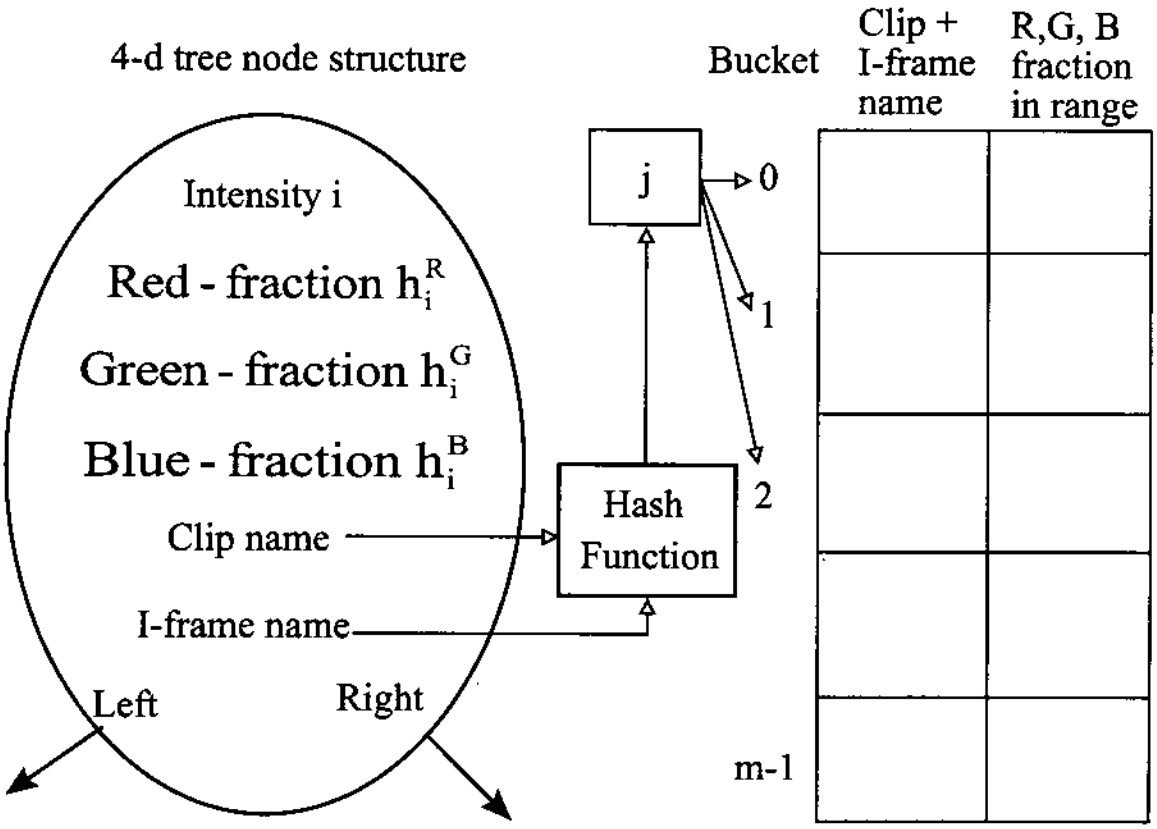


Figure 2.19 A data structure with a 4-d tree node and hash table structure for collecting histogram range search results.

2.3 Colour Space Indexing

In this method, we count the number of pixels in one I-frame having a colour (i.e. red, green, and blue) value within a specified colour space (e.g. R, G, B) range. The R,G, B values are used as the keys to index digital video data stored in a k-d tree spatial data structure.

If we consider the mathematical space of colours (as described in the RGB model), we can model it as a cube, in which each of the axes corresponds to one of the primary colours R, G and B. Each of these components can take 256 different values (for an image in 24 bit colour), meaning that there are 2^{24} (16,777,216) possible points in the colour space. This method has the advantage of allowing complete colour space searches compared to the histogram method, as it considers all three colour components together, simultaneously.

2.3.1 Range Search

Example queries to be used for colour space searching are shown in Figure 2.20. Using colour space indexing, the range queries follow the k-d tree range search algorithm. The number of pixels in a given MPEG-2 I-frame which fall in the given range search window are counted and stored in an associated array. The total number of pixels for the I-frame is also stored in that array, allowing the percentage of pixels for the I-frame to be calculated and used as the search criteria.

1. Find MPEG-2 video I-frames with $> t$ % pixels in color range $[R_L, R_H]$, $[G_L, G_H]$, $[B_L, B_H]$, simultaneously.
2. Find MPEG-2 video I-frames with $< t$ % pixels in color range $[R_L, R_H]$, $[G_L, G_H]$, $[B_L, B_H]$, simultaneously.
3. Find MPEG-2 video I-frames with between t_1 % and t_2 % pixels in color range $[R_L, R_H]$, $[G_L, G_H]$, $[B_L, B_H]$, simultaneously.

Figure 2.20 Sample colour space range queries.

These queries could be used, for example, to detect "mainly black" or "mainly white" frames.

2.3.2 Data Structures

To satisfy queries listed in Figure 2.20, we use a combination of k-d tree and AVL tree data structures. Each k-d tree node is associated with an AVL tree.

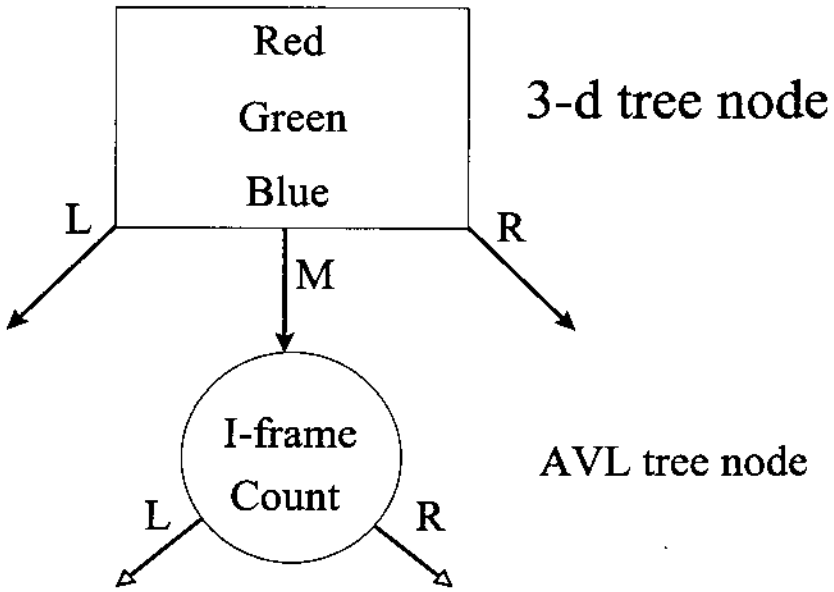
2.3.2.1 AVL Tree

The AVL tree is a binary search tree with the additional balance property that, for any node in the tree, the height of the left and right sub-trees can differ by at most 1. This balance condition ensures that the depth of the tree is $O(\log N)$.

2.3.2.2 Node Structures

There are two kinds of nodes in this data structure as shown in Figure 2.21. The first node type is a 3-d tree node. After sampling and pre-processing the MPEG-2 video clip I-frames data (see chapter 3 for details), a 3-d tree node is used for indexing the video data. Each node contains a 3-tuple key treated as a point in three dimensions. The (R, G, B) 3-tuple contains the red, green, and blue values of a single pixel. Each 3-d tree node also contains R and L pointers that point to two children as well as an M pointer that points to an AVL tree.

The second node type is as AVL tree node. This node has two fields. One expresses the MPEG-2 video I-frame name which is represented by an integer. The other is a counter of pixels having this colour in this I-frame. It also contains R and L pointers to the node's two children.




```
struct KdColourTreeNode
{
      // The data in the node (r,g,b value)
      char  Red, Green, Blue;
      DType              Disc;   // Discriminator
      KdColourTreeNode* L;       // Left child
      KdColourTreeNode* R;       // Right child
      AVLTreeNode*       M;      // points to AVL tree
}


struct AVLTreeNode
{
       int I-frame, count;       // The data in the node
       AVLTreeNode* L;           // Left child
       AVLTreeNode* R;           // Right child
}
```

Figure 2.21 Node structures for the colour space tree.

2.3.2.3 Construction of Tree Data Structures

To get a balanced 3-d tree, a dummy I-frame is introduced. The dummy I-frame is constructed and inserted into the 3-d tree in such a way that the 3-d tree is complete (i.e. perfectly balanced). That means the tree was constructed by inserting the 3-d nodes in the following order (red, green, blue): (127,127,127), (63,127,127), (191,127,127), (63,63,127), (63,191,127), (191, 63, 127), (191,191,127)... (255, 255, 255). The algorithm shown in Figure 2.22 was used to generate the pixel values in this order. The height of the 3-d tree generated by inserting the dummy I-frame was 21. This height is controlled by the variable "limit" in the following algorithm.

```
main (void)
{  Rmin =  Gmin =  Bmin = 0;
   Rmax =  Gmax =  Bmax = 255; limit = 3;
   GenR (Rmin, Rmax, Gmin, Gmax, Bmin, Bmax, limit);
}
GenR (Rmin, Rmax, Gmin, Gmax, Bmin, Bmax, limit)
{  if (Rmax - Rmin < limit) return;
   Rmid =(Rmax + Rmin)/2; Gmid =(Gmax + Gmin)/2; Bmid =(Bmax + Bmin)/2;
   cout << Rmid << " " << Gmid << " " << Bmid << endl;
   GenG (Rmin, Rmid, Gmin, Gmax, Bmin, Bmax, limit);
   GenG (Rmid, Rmax, Gmin, Gmax, Bmin, Bmax, limit);
}
GenG (Rmin, Rmax, Gmin, Gmax, Bmin, Bmax, limit)
{  if (Gmax - Gmin < limit) return;
   Rmid =(Rmax + Rmin)/2; Gmid =(Gmax + Gmin)/2; Bmid =(Bmax + Bmin)/2;
   cout << Rmid << " " << Gmid << " " << Bmid << endl;
   GenB (Rmin, Rmax, Gmin, Gmid, Bmin, Bmax, limit);
   GenB (Rmin, Rmax, Gmid, Gmax, Bmin, Bmax, limit);
}
GenB (Rmin, Rmax, Gmin, Gmax, Bmin, Bmax, limit)
{  if (Bmax - Bmin < limit) return;
   Rmid =(Rmax + Rmin)/2; Gmid =(Gmax + Gmin)/2; Bmid =(Bmax + Bmin)/2;
   cout << Rmid << " " << Gmid << " " << Bmid << endl;
   GenR (Rmin, Rmax, Gmin, Gmax, Bmin, Bmid, limit);
   GenR (Rmin, Rmax, Gmin, Gmax, Bmid, Bmax, limit);
}
```

Figure 2.22 The algorithm to generate the dummy I-frame pixel values.

The algorithm for insertion into the colour space tree is given in Figure 2.23. It makes use of the KdColourCompare algorithm (Figure 2.24) to determine which direction to branch. Figure 2.25 illustrates the colour space tree data structure.

```
// Add P into colour space subtree rooted at T
Insert (KdColourTreeNode P, KdColourTreeNode T, I-frameName)
{    if (T == NULL)
     {          T = P;
          T.Discriminator = red;
          T.M.Insert_AVL (I-frameName);
          return;
     }

     KdTreeNode  F, SaveRoot = T;
     while (T != NULL && (P.coordinates(Red, Green, Blue)
                    !=  T.coordinates(Red, Green, Blue)))
     {    F = T;  // Remember parent of T
          Direction = KdColourCompare(P, T);
          if (Direction == L)  T = T.L;
          else T = T.R;
     }

     if (T == NULL) // P is not already in tree.
     {
          if ( Direction == L )
          {    F.L = P;
               F.L.M.Insert_AVL(I-frameName);
          }
          else
          {    F.R = P;
               F.R.M.Insert_AVL (I-frameName);

          }
          if (F.Discriminator == red )
               P.Discriminator = green;
          else if (F.Discriminator == green)
               P.Discriminator = blue;
          else
               P.Discriminator = red;
     }
     else
          T.M.Insert_AVL (I-frameName);
     T = SaveRoot;
}
```

Figure 2.23 The algorithm for colour space tree insertion.

```
// Return the child type of the 3-d tree root at node
// Q in which node P belongs.
KdColourCompare(KdColourTreeNode P, KdColourTreeNode Q )
{     if (Q->Discriminator == r)
      {     if (P.Red < Q.Red) return L;
            else return R;
      }
      else if ( Q.Discriminator == g)
      {     if (P.Green < Q.Green) return L;
            else return R;
      }
      else
      {     if (P.Blue < Q.Blue) return L;
            else return R;
      }
}
```

Figure 2.24 The algorithm used for 3-d tree node comparison to determine branching direction.

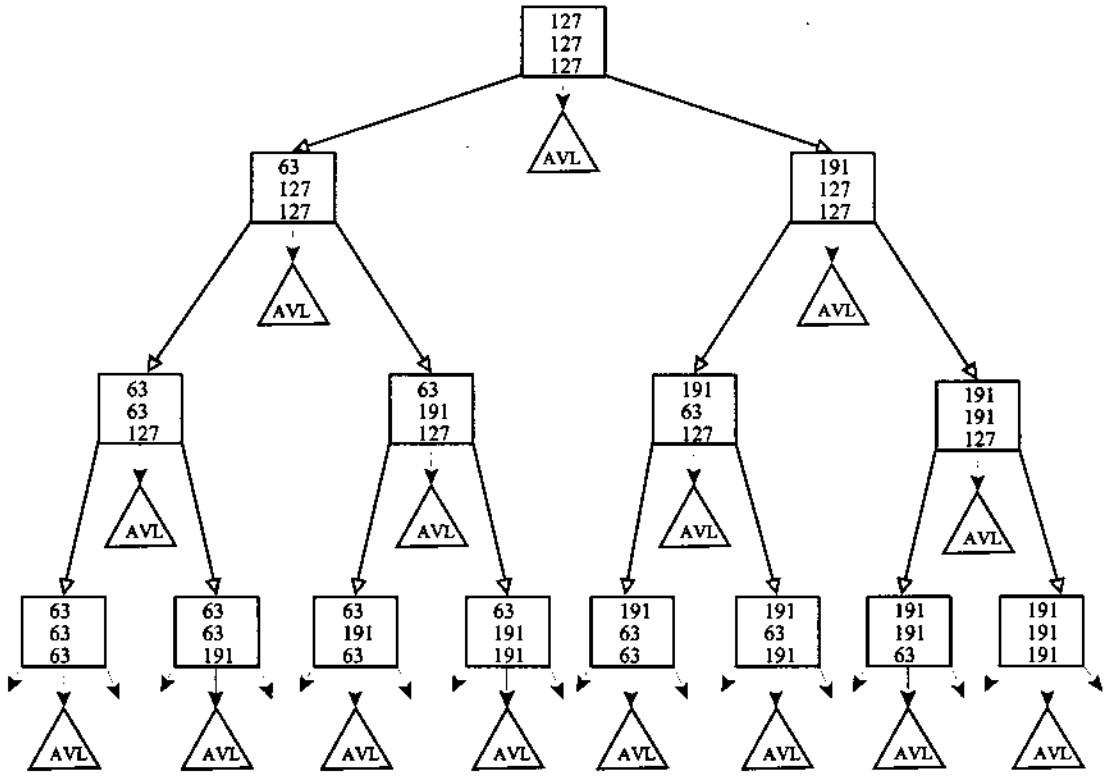


Figure 2.25 Top part of a colour space tree resulting from the dummy I-frame insertion.

2.3.2.4 Range Search

We define a parallel array to accumulate the counts of pixels of I-frames which are in range. Each element of the array contains the I-frame name, count of pixels in range, and total pixels for this I-frame. The array index represents the name of the I-frame. A range search with respect to a colour space tree will follow the algorithm shown in Figure 2.10. When range searching, and a 3-d tree node is in range, the associated AVL tree nodes are examined to sum up the counts for each I-frame in the parallel array. After range searching, the histogram for each I-frame can be calculated. Figure 2.26 shows this structure.

For example, to compute pixel counts of I-frames shown in Figure 2.26, for a range search having $R_L \leq 63$, $R_H \geq 191$, $G_L \leq 127$, $G_H \geq 127$, $B_L \leq 127$, $B_H \geq 127$, we first reach the 3-d tree node (127,127,127). Following the middle link, we get the count for name "1" be 2 and name "0" be 3. Secondly, we reach the node (63,127,127) and sum up the counts for name "1", i.e., 2 + 4 = 6 and name "0", i.e. 3 + 8 = 11. Finally we reach the node (191,127,127), where the same process is carried out. Now the count for name "0" is equal to15, i.e, 3+8+4 = 15 and for name "1" is 12, i.e, 2+4+6=12. So the relative pixel counts for name "0" and name "1" are calculated as $\frac{15}{84480}$ and $\frac{12}{76800}$, respectively.

To satisfy the queries as shown in Figure 2.20, the final step is to search the array and report the result. For example, after building the colour space tree with 17 I-frames, the range query "Find MPEG-2 video I-frames with > 50% pixels in colour range [0, 132], [0, 213] , [150, 255], simultaneously" is employed. After the range search is

completed,

we go through the array and calculate the histogram for each I-frame to see if there is any histogram value is greater than 50%. If so, the I-frame name, count, total pixels, and histogram are printed out. Figure 2.27 shows an example of a range search result.

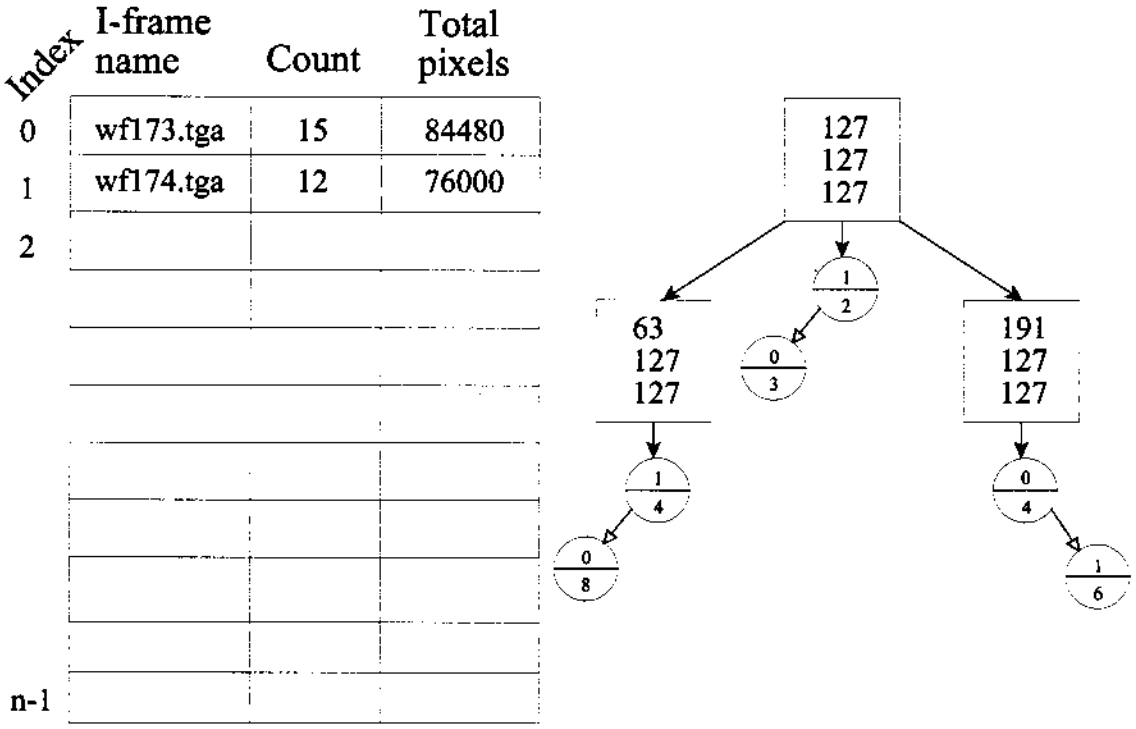


Figure 2.26 A colour space tree and the accumulator array.

```
After range search
Find the I-frame whose pixel% > 50%.
/drives/somato/grads/g01hy/I_frames/wf151/rec104.tga 83085 84480
The pixel% = 98.3487%
/drives/somato/grads/g01hy/I_frames/wf151/rec119.tga 82654 84480
The pixel% = 97.8385%
```

Figure 2.27 An example of a colour space range search result.

Chapter 3 EXPERIMENTAL RESULTS

3.1 Sample Data

As the Internet and Word Wide Web explode into common usage, the MPEG-2 video clips were obtained by using World Wide Web browsers (such as Netscape Navigator) and search engines (such as Alta Vista and Excite) to search through the Internet. 100 MPEG-2 video clips which include 5706 I-frames were used in this experiment. These video clips come in a variety of types, including news, cartoons, advertisements, scenes, and sports videos. Data describing 30 of these video clips is listed in Table 3.1. The table also shows the size in bytes and the duration in "mm ss" format which is standard for minutes and seconds for each video clip.

With the new technologies, PC- or workstation-based vision systems can provide a user with a tremendous amount of flexibility in what the user can do with the stored video clip display capability. For example, with the Microsoft Windows Media Player, the user can play MPEG-2 video clips forward or backward, and resize it. The duration of playback also can be counted using the software. MPEG-2 requires higher data rates but delivers higher image resolution, and picture quality. MPEG-2 also requires dedicated hardware for playback but looks better than a television screen [Vau96].

Table3.1 MPEG-2 video clip samples (30 out of 102).

No.	Name	Size (bytes)	Duration (mm ss)
1	512av.mpg	1254432	00 11
2	akl.mpg	1043990	00 05
3	b2future.mpg	5060531	00 17
4	bird.mpg	950312	00 13
5	c-saw.mpg	769345	00 05
6	camera.mpg	678832	00 07
7	creature.mpg	6684910	00 32
8	crissy.mpg	846924	00 04
9	crissy2.mpg	826844	00 04
10	destruct.mpg	1112642	00 06
11	drill1.mpg	701970	00 04
12	drill2.mpg	683439	00 04
13	fantasia.mpg	5234174	01 04
14	gump.mpg	5018811	00 19
15	hammer.mpg	714289	00 05
16	hell.mpg	1334446	00 09
17	hookip.mpg	1611440	00 07
18	hookip2.mpg	612386	00 03
19	hur.mpg	1064494	00 05
20	isdndual.mpeg	510308	00 09
21	jel_fish.mpg	1900617	00 14
22	jet.mpg	827839	00 36
23	mvcdsmpl.mpg	1049008	00 05
24	napier.mpg	1048086	00 05
25	port.mpg	1046038	00 05
26	producer.mpg	1494812	00 08
27	ski.mpg	1041930	00 05
28	skybird.mpg	514584	00 09
29	wf170.mpg	1380356	00 06
30	wrench.mpg	716467	00 05

3.2 Pre-processing

In this research, we used programs adapted from software by the MPEG Software Simulation Group[MSSG99] to process the data which can be used for the tree constructions described in Chapter 2. The MPEG Software Simulation Group is currently developing MPEG software with the purpose of providing aid in understanding the various algorithms which comprise an encoder and decoder, and giving a sample implementation based on advanced encoding models. The implementation of the codec converts uncompressed video frames into MPEG-1 and MPEG-2 video coded bit-stream sequences, and vice versa.

3.2.1 I-frame Extraction and RGB Format Generation

The implementation of the decoder from The MPEG Software Simulation Group [MSSG99] is made of 16 C source code files comprising a total of 8540 lines of source code. Each file has several functions. There are a total of 227 functions. There is a very complicated relationship among the files and functions. After using cross-referencing analysis (using "GNU cxref" which is a C program cross-referencing and documentation tool for UNIX [Demon99]), we found a file store.c that contains picture output routines. One of the functions of this file is to generate Truevision TGA ("Targa") files that contain 24-bit R, G, B values in uncompressed format for all I-, B-, and P-frames of input MPEG-2 video clips. Since we deal exclusively with I-frames, one condition statement was added to the store.c file (see line 10 in Figure 3.1) in such a way that only the I-frames are produced when MPEG-2 video clips are decoded. Figure 3.1 shows the

algorithm used to write I-frames in TGA format. The I-frame's TGA files are used for either histogram calculation or as the input of colour space tree insertion.

```
/* store.c, picture output routines*/
...
01 static void store_ppm_tga(char *outname, unsigned char *src[],
02 int offset, int incr, int height, int tgaflag);
03 static void putbyte(int c);
04 static void putword(int w);
...
/* store as PPM or uncompressed Truevision TGA ('Targa') file */
05 static void store_ppm_tga(outname,src,offset,incr,height,tgaflag)
06 {  int i, j, y, u, v, r, g, b, crv, cbu, cgu, cgv;
07    unsigned char *py, *pu, *pv;
08    static unsigned char tga24[14] = {0,0,2,0,0,0,0,
09                                      0,0,0,0,0,24,32};
10    if(picture_coding_type == I_TYPE) /* April 21 1999 added
                                           to test I-frame*/
      {
            ...
11          if (tgaflag)  /* TGA header */
12          {     for (i=0; i<12; i++)
13                      putbyte(tga24[i]);
14                putword(horizontal_size); putword(height);
15                putbyte(tga24[12]); putbyte(tga24[13]);
            }
16          else  /* PPM header */
                  ...
            /* matrix coefficients */
17          crv = Inverse_Table_6_9[matrix_coefficients][0];
                  ...
18          for (i=0; i<height; i++)
19          {     py = src[0] + offset + incr*i;
20                pu = u444 + offset + incr*i;
21                pv = v444 + offset + incr*i;
22                for (j=0; j<horizontal_size; j++)
23                {     u = *pu++ - 128; v = *pv++ - 128;
24                      y = 76309 * (*py++ - 16);
25                      r = Clip[(y + crv*v + 32768)>>16];
26                      g = Clip[(y - cgu*u - cgv*v + 32768)>>16];
27                      b = Clip[(y + cbu*u + 32786)>>16];
                  }
28                if (tgaflag)
29                      putbyte(b); putbyte(g); putbyte(r);
30                else
31                      putbyte(r); putbyte(g); putbyte(b);
            }
      }
32 static void putbyte(c)
33 {  *optr++ = c;
34    if (optr == obfr + OBFRSIZE)
35          write(outfile,obfr,OBFRSIZE); optr = obfr;
   }
36 static void putword(w)
37    putbyte(w); putbyte(w>>8);
```

Figure 3.1 Part of the algorithm for generating I-frame Truevision TGA files from MPEG-2 encoded video clips.

3.2.2 Histogram Computation

After obtaining each I-frame's TGA file, the next step is to extract the important information such as horizontal size and height as well as the number of pixels in given R, G, and B values from the TGA file in order to calculate the total pixels and the R, G, and B histograms as defined in equation (2.1). In this way, a colour histogram file for each I-frame can be created. The file format is illustrated in Figure 3.2.

```
127 4.616477e-03 5.078125e-03 2.746212e-03
063 4.912405e-03 8.984375e-03 4.971591e-03
031 2.864583e-03 5.089962e-03 1.105587e-02
015 2.769886e-03 5.196496e-03 1.057055e-02
007 2.107008e-03 4.580966e-03 8.510890e-03
003 1.657197e-03 3.444602e-03 7.125947e-03
001 1.420455e-03 2.367424e-03 6.841856e-03
000 7.895360e-03 1.133996e-02 9.798769e-02
002 1.361269e-03 3.065814e-03 7.232481e-03
005 1.941288e-03 3.681345e-03 7.788826e-03
004 1.609848e-03 3.302557e-03 8.309659e-03
006 2.710701e-03 4.474432e-03 8.510890e-03
011 2.391098e-03 4.853220e-03 8.652936e-03
009 2.225379e-03 5.078125e-03 8.747633e-03
008 2.237216e-03 4.900568e-03 8.925189e-03
010 2.059659e-03 5.480587e-03 9.647254e-03
013 2.710701e-03 5.587121e-03 9.457860e-03
012 2.095170e-03 5.338542e-03 9.280303e-03
014 2.651515e-03 5.504261e-03 9.943182e-03
023 2.840909e-03 5.007102e-03 1.195549e-02
019 2.521307e-03 5.421402e-03 1.150568e-02
017 2.272727e-03 5.184659e-03 1.117424e-02
016 2.758049e-03 5.007102e-03 1.084280e-02
018 2.781723e-03 5.243845e-03 1.038116e-02
021 2.402936e-03 5.125473e-03 1.206203e-02
020 2.746212e-03 4.876894e-03 1.081913e-02
022 2.758049e-03 5.149148e-03 1.278409e-02
027 2.793561e-03 5.397727e-03 1.083097e-02
025 2.698864e-03 4.711174e-03 1.228693e-02
024 2.805398e-03 5.362216e-03 1.181345e-02
026 2.781723e-03 5.433239e-03 1.202652e-02
029 2.604167e-03 4.900568e-03 1.151752e-02
...
```

Figure 3.2 An example of part of a colour histogram file formatted from left to right as intensity, R-fraction, G-fraction and B-fraction (the complete file has 256 lines).

To get the information for calculating the colour histogram, an analysis is made based on the algorithm listed in Figure 3.1 to determine which byte position is used to store horizontal size, height and R, G, B values. After the "for" loop at line 14, the file pointer position is in byte 12. The horizontal size is written to the file as a word, i.e., 2 bytes, meaning that it occupies bytes 12 and 13. Then, the height is written to the file as bytes 14 and 15. Bytes 16 and 17 are occupied by the last two elements of character array tga[24]. Starting with byte 18, the R, G, B values are written to the file in B, G, R order. Figure 3.3 illustrates this format. H_S_1 represents the least significant 8 bits of the horizontal size, and H_S_2 represents the most significant 8 bits. Similarly, H_1 and H_2 represent the image height.

byte 0	byte 1	byte 2	byte 3	.....	byte12	byte13	byte14	byte15
0	0	2	0	.....	H_S_1	H_S_2	H_1	H_2
byte 16	byte 17	byte 18	byte 19	byte 20	byte 21	byte 22	byte 23	.....
24	32	B	G	R	B	G	R	.....

Figure 3.3 The storage format of a TGA file.

Based on the analysis above, when a TGA file is opened, we can set the file pointer to byte position 12. Starting from that position, the horizontal size, height, and R, G, B data can be extracted and used to compute the colour histogram. Figure 3.4 shows this process.

```
Compute_Histogram ( )
{     // open the I_frame input file, e.g."rec119.tga"
      if ((in = fopen(INFNAME, "rt")) == NULL)
      {
            fprintf(stderr, "Cannot open I_frame input file.\n");
            return 1;
      }
      /*seek the position 12 for Horizontal size(H_S1)*/
      fseek(in , 12L, SEEK_SET);
      ch = fgetc(in); /*get the char*/
      X = (int) ch;   /*convert into int*/
      /* seek the position 13 for Horizontal size(H_S2)*/
      fseek(in , 13L, SEEK_SET);
      ch = fgetc(in); Y = (int) ch << 8;
      Horizontal_size = X + Y;

      /* seek the position 14 for Height (H1)*/
      fseek(in , 14L, SEEK_SET);
      ch = fgetc(in); X = (int) ch;
      /* seek the position 15 for for Height (H2)*/
      fseek(in , 15L, SEEK_SET);
      ch = fgetc(in); Y = (int) ch << 8;
      Height = X + Y;

      /* computing the total number of pixels*/
      Total_pixel = Horizontal_size * Height;

      /* read the R G B data from position byte 19*/
      fseek(in , 19L, SEEK_SET);
      for (j = 0; j < Height; j++)
      {
            for (k = 0; k < Horizontal_size; k++)
            {
                  ch = fgetc(in);  X = (int) ch;  /* B */
                  ch1 = fgetc(in); Y = (int) ch1; /* G */
                  ch2 = fgetc(in); Z = (int) ch2; /* R */

                  *(R[j] + k) = Z; /*stroe R G B to arrays*/
                  *(G[j] + k) = Y;
                  *(B[j] + k) = X;
            }
      }
      /* compute the histograms */
      for (i = 0; i < HIGH; i++)
      {
            R_frac[i] = (double) R_pixel_count[i]/Total_pixel;
            G_frac[i] = (double) G_pixel_count[i]/Total_pixel;
            B_frac[i] = (double) B_pixel_count[i]/Total_pixel;
      }

}
```

Figure 3.4 The algorithm for extracting information from a TGA file.

For example, an MPEG-2 video clip named "Wfl74.mpg" was decoded by the decoder, and an I-frame TGA file called "rec119.tga" was generated with horizontal size and height being 352 and 240, respectively. The size of "rec119.tga" is 253458 bytes. We applied the algorithm listed in Figure 3.4 and got that byte 12 is "96" and byte 13 is "1". After shifting "1" in byte 13 to the left by 8 bits, we get byte 13 as 256. The horizontal size is equal to 96 + 256 = 352. Similarly, the value of height = 240.

3.3 Constructing the Index

After pre-processing the MPEG-2 video clip data, the experimental test set-ups are built to construct the index for both histogram and colour space cases.

3.3.1 Colour Histogram

One hundred MPEG-2 video clips containing 5706 I-frames are used in this part. A histogram file formatted as shown in Figure 3.2 is generated for each I-frame. For every video clip, we created a file called "insert_clipname.dat". This file contains the clip location, each I-frame location in the file system and the 256 histogram values of each I-frame. It is used as the input file when a 4-d tree is constructed. Figure 3.5 illustrates an example of this kind of file. The file is named "insert_wfl74.dat".

```
~/mpeg/wf174.mpg
~/I_frames/wf174/rec104.tga
127 4.616477e-03 5.078125e-03 2.746212e-03
063 4.912405e-03 8.984375e-03 4.971591e-03
031 2.864583e-03 5.089962e-03 1.105587e-02
015 2.769886e-03 5.196496e-03 1.057055e-02
007 2.107008e-03 4.580966e-03 8.510890e-03
...
252 9.469697e-04 6.155303e-04 2.012311e-04
254 6.036932e-04 5.208333e-04 1.302083e-04
255 1.369555e-02 6.664299e-03 8.285985e-05
~/I_frames/wf174/rec134.tga
127 5.859375e-03 4.616477e-03 2.462121e-03
063 4.924242e-03 9.446023e-03 5.042614e-03
031 2.852746e-03 6.463068e-03 1.091383e-02
015 2.616004e-03 4.924242e-03 1.130445e-02
007 2.035985e-03 4.166667e-03 8.771307e-03
...
253 1.231061e-03 4.498106e-04 2.012311e-04
252 1.017992e-03 6.865530e-04 2.722538e-04
254 1.006155e-03 4.853220e-04 2.840909e-04
255 1.419271e-02 7.232481e-03 1.396780e-03
~/I_frames/wf174/rec29.tga
127 4.427083e-03 5.018939e-03 3.338068e-03
063 5.018939e-03 9.114583e-03 5.326705e-03
031 2.781723e-03 6.072443e-03 1.029830e-02
015 3.030303e-03 5.516098e-03 1.128078e-02
...
007 1.811080e-03 3.989110e-03 8.451705e-03
003 1.053504e-03 2.201705e-03 8.049242e-03
001 1.017992e-03 1.669034e-03 6.664299e-03
252 9.232955e-04 3.196023e-04 7.102273e-05
254 5.800189e-04 2.012311e-04 3.551136e-05
255 1.336411e-02 4.415246e-03 5.918561e-05
~/I_frames/wf174/rec59.tga
127 4.959754e-03 4.462595e-03 2.959280e-03
063 5.243845e-03 9.079072e-03 5.468750e-03
031 2.769886e-03 5.977746e-03 1.072443e-02
015 2.521307e-03 6.238163e-03 1.193182e-02
007 1.929451e-03 3.562973e-03 8.534564e-03
003 1.148201e-03 2.426610e-03 7.634943e-03
001 1.124527e-03 2.166193e-03 6.865530e-03
000 9.339489e-03 1.332860e-02 1.102983e-01
002 1.160038e-03 2.130682e-03 7.634943e-03
005 1.455966e-03 3.089489e-03 8.333333e-03
004 1.526989e-03 2.805398e-03 7.836174e-03
006 1.751894e-03 3.349905e-03 8.996212e-03
...
```

Figure 3.5 An example of part of the insert_wf174.dat histogram file.

We also stored all file names of this type of file into another file named filename.dat. Figure 3.6 shows this file format. When the 4-d tree is build, the "filename.dat" file is opened first to get different input file names. Once each input file is opened, the data in the file can be inserted into a 4-d tree to build the colour histogram indexing. The 4-d data structure has been discussed in Chapter 2. The experimental test architecture is illustrated in Figure 3.7. Member functions KdInsert and KdSearch are part of the Kdtree object stored in file Kdtree_RGB.cc.

```
insert_akl.dat
insert_andrew.dat
insert_atom.dat
insert_b2future.dat
insert_badday.dat
insert_berr.dat
insert_bird.dat
insert_c-saw.dat
insert_camera.dat
insert_creature.dat
insert_crissy.dat
insert_destruct.dat
insert_dmskib.dat
insert_drill1.dat
insert_drill2.dat
insert_dtskib.dat
insert_emily.dat
insert_explode1.dat
insert_fantasia.dat
insert_fractal1.dat
insert_fractal2.dat
...
insert_ft1skib1.dat
insert_Ft1skib3.dat
insert_gump.dat
insert_hammer.dat
insert_hardenbe.dat
insert_heihachi.dat
insert_wf172.dat
insert_wf173.dat
insert_wf174.dat
insert_wrench.dat
insert_zih.dat
```

Figure 3.6 Part of the "filename.dat" file (the complete file has 100 lines, one per MPEG-2 video clip).

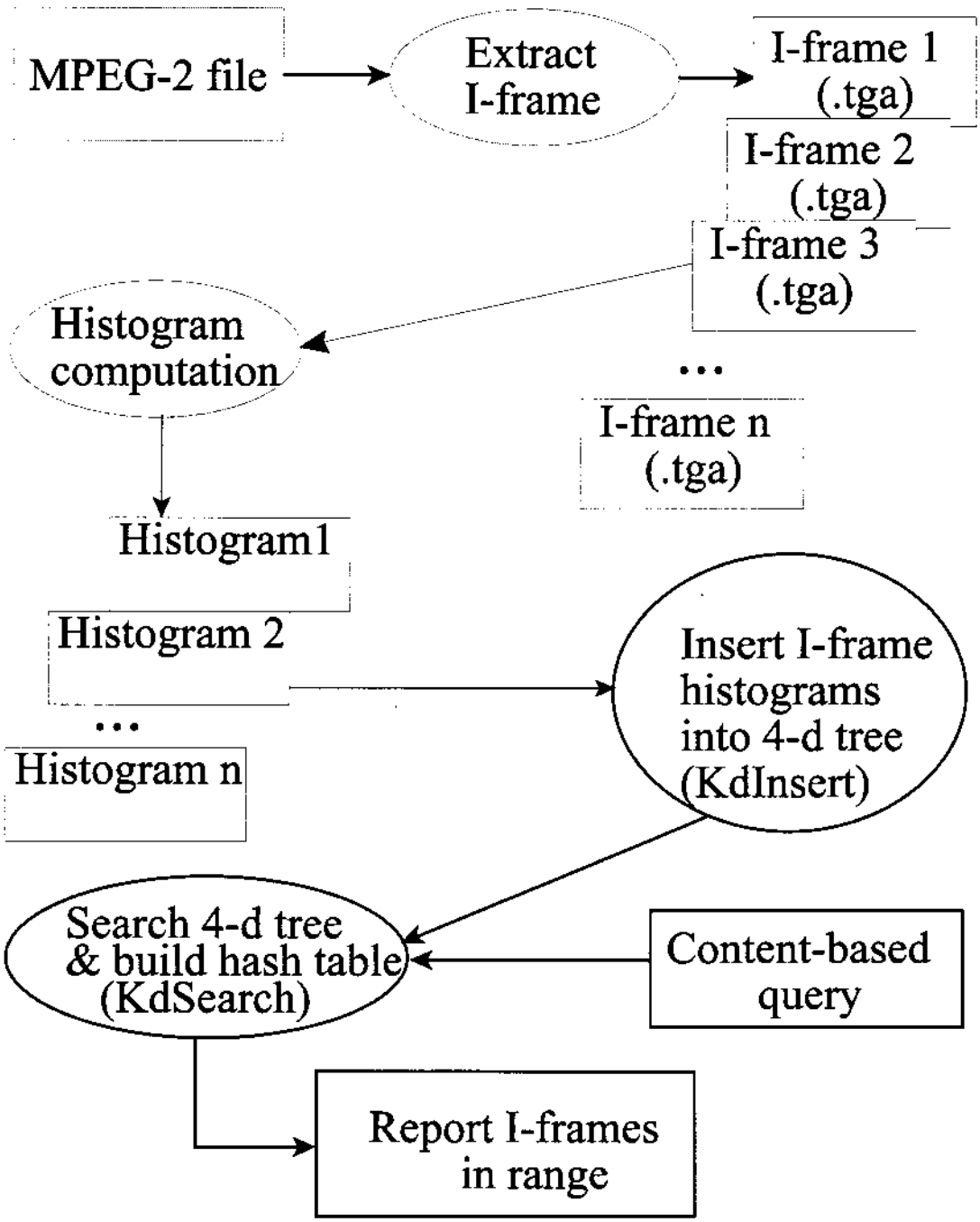


Figure 3.7 Experimental test architecture for colour histogram index testing.

In this experiment, there are one hundred MPEG-2 video clip files, 5706 I-frame TGA files, 5706 histogram files (all histograms for one video clip are contained in one .dat input file), and a filename.dat file involved in the computation. A total number of 5907 files overall are used for the test data. Figure 3.8 illustrates the file system that involves the 4-d tree construction.

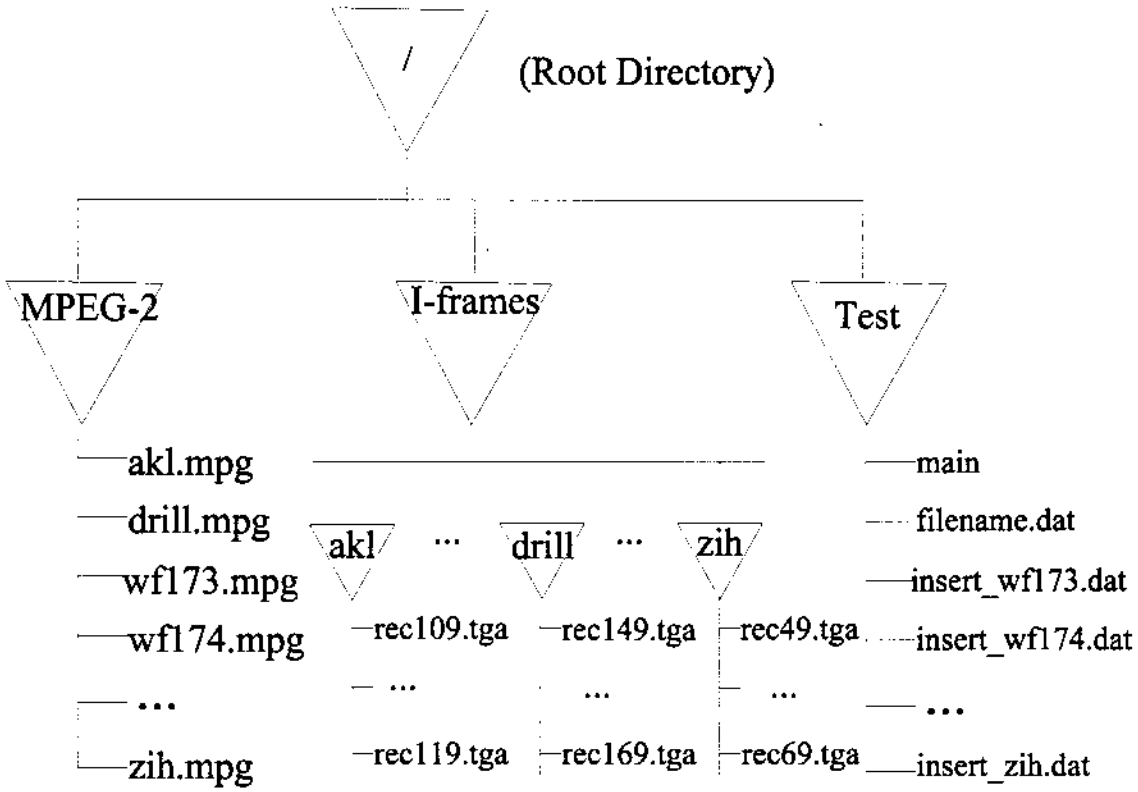


Figure 3.8 Directory structure of file system for constructing a 4-d tree.

3.3.2 Colour Space

Three hundred I-frames from 100 video clips were used to do the experiment on colour space tree construction. Each I-frame's TGA file can be inserted into the colour space tree directly. The construction of the colour space tree starts with the dummy I-frame insertion as we discussed in Chapter 2. The details of colour space tree data structure are also shown in Chapter 2. Figure 3.9 illustrates the experimental test set-up for the colour space tree indexing. Member functions KdInsert and KdSearch are part of the Kdtree object stored in file Kdtree_Cspace.cc.

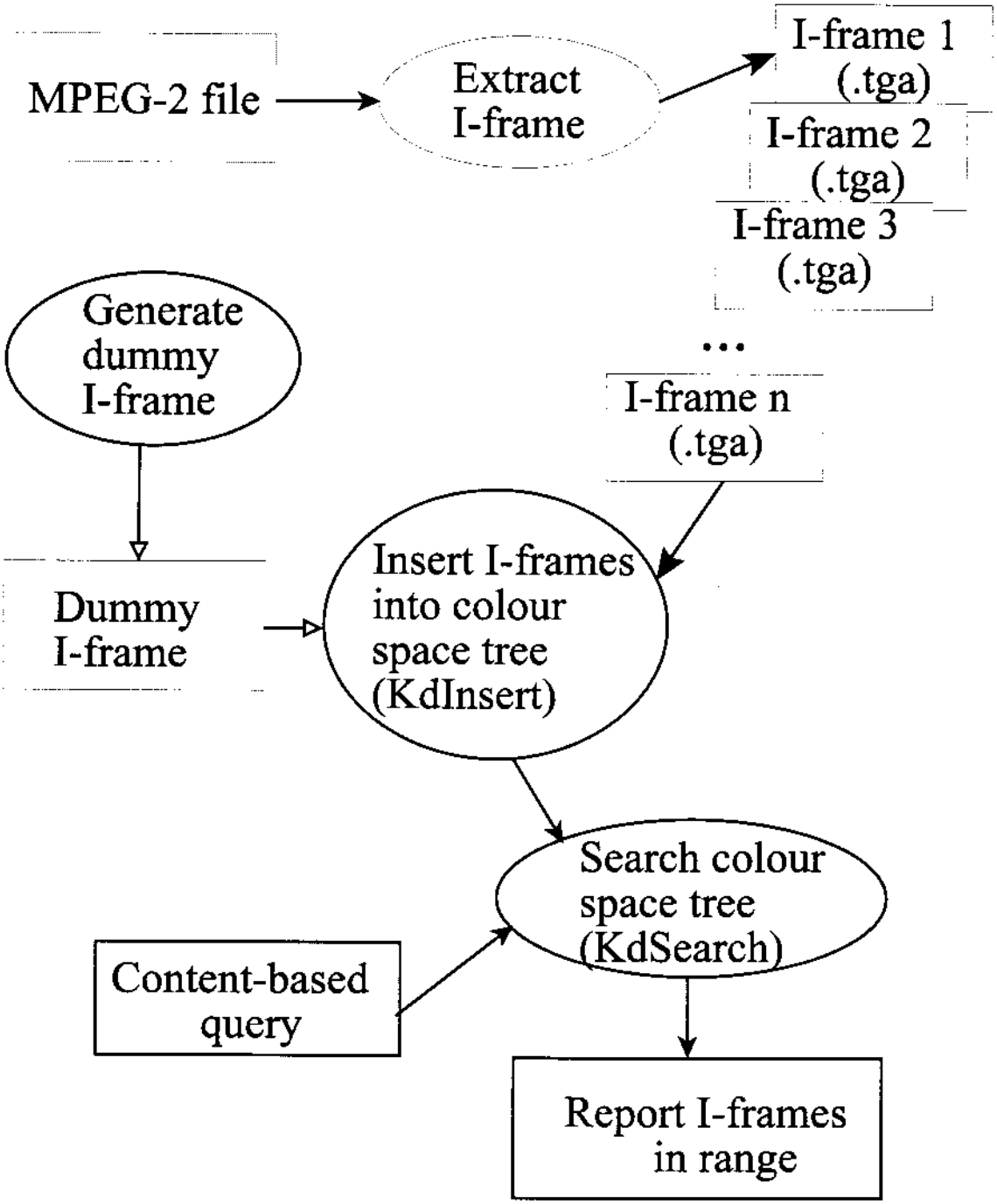


Figure 3.9 The experimental test set-up for colour space tree testing.

3.4 Range Search Results

The colour histogram index testing was performed on somato, a Sun Microsystems ULTRA 5 workstation, using the Solaris 2.6 operating system. Somato has 128 MB of RAM and up to 768 MB virtual memory to be used. It is located in the Artificial Intelligence Laboratory of the Faculty of Computer Science, University of New Brunswick and its central processing unit/processor (UltraSPARC-IIi) runs at 270 MHZ. The C++ language was used for the implementation of prototype software on the UNIX system. There are 3137 lines of C++ code including 214 lines of comments involved the implementation.

The colour space tree testing was performed on jupiter, a Sun Microsystems ENTERPRISE 250 workstation, using the Solaris 7 operating system. Jupiter has 1024 MB of RAM and up to 4 GB virtual memory to be used. It is located in the Computing Services Department of the University of New Brunswick and is a two-processor (UltraSPARC-II) multi-user system which runs at 400 MHZ. The implementation of prototype software on the UNIX system was written in the C++ language. There are 1893 lines of C++ code including 153 lines of comments.

It took 71 seconds to generate the 4-d tree on somato for all 5706 I-frames. Figure 3.10 shows an example of colour histogram index testing results based on the range query "Find MPEG-2 video I-frames with > 10% pixels in colour range [R_L = 40, R_H = 50] ". After range searching, 416 I-frames are found in the range. The result also shows the location of each MPEG-2 video clip and the I-frame that is in range. It took 8 seconds to perform this range search on somato. Table 3.2 shows other single colour histogram

range search results. The times are CPU times for one run of the range search program.

```
This video clip's I-frame matches the range search.
~/mpeg/lily.mpg          ~/I_frames/lily/rec37.tga
This video clip's I-frame matches the range search.
~/mpeg/270898seg7.mpg    ~/I_frames/270898seg7/rec5969.tga
This video clip's I-frame matches the range search.
~/mpeg/270898seg7.mpg    ~/I_frames/270898seg7/rec3539.tga
This video clip's I-frame matches the range search.
~/mpeg/heihachi.mpg      ~/I_frames/heihachi/rec1197.tga
This video clip's I-frame matches the range search.
~/mpeg/heihachi.mpg      ~/I_frames/heihachi/rec57.tga
This video clip's I-frame matches the range search.
~/mpeg/introsg1.mpg      ~/I_frames/introsg1/rec17.tga
This video clip's I-frame matches the range search.
~/mpeg/spacewal.mpg      ~/I_frames/spacewal/rec207.tga
This video clip's I-frame matches the range search.
~/mpeg/score.mpg         ~/I_frames/score/rec2157.tga
This video clip's I-frame matches the range search.
~/mpeg/310798seg3.mpg    ~/I_frames/310798seg3/rec29.tga
This video clip's I-frame matches the range search.
~/mpeg/fractal1.mpg      ~/~/I_frames/fractal1/rec93.tga
This video clip's I-frame matches the range search.
~/mpeg/gump.mpg          ~/I_frames/gump/rec269.tga
This video clip's I-frame matches the range search.
~/mpeg/270898seg7.mpg    ~/I_frames/270898seg7/rec5249.tga
This video clip's I-frame matches the range search.
~/mpeg/wf121.mpg         ~/I_frames/wf121/rec209.tga
This video clip's I-frame matches the range search.
~/mpeg/270898seg7.mpg    ~/I_frames/270898seg7/rec2834.tga
This video clip's I-frame matches the range search.
~/mpeg/270898seg8.mpg    ~/I_frames/270898seg8/rec4379.tga
This video clip's I-frame matches the range search.
~/mpeg/310798seg3.mpg    ~/I_frames/310798seg3/rec59.tga
This video clip's I-frame matches the range search.
~/mpeg/270898seg7.mpg    ~/I_frames/270898seg7/rec3224.tga
This video clip's I-frame matches the range search.
~/mpeg/pvr.mpg           ~/I_frames/pvr/rec532.tga
This video clip's I-frame matches the range search.
~/mpeg/270898seg7.mpg    ~/I_frames/270898seg7/rec6779.tga
This video clip's I-frame matches the range search.
~/mpeg/hardenbe.mpg      ~/I_frames/hardenbe/rec312.tga
......
```

Figure 3.10 An example result of the colour histogram index testing.

Table 3.2 (part 1 of 2) Single histogram range search results.

Query	No. of I-frames found	Time required(s)
1. Find MPEG-2 video I-frames with > 80 % pixels in histogram range [$R_L = 0, R_H = 155$].	2863	74.64
2. Find MPEG-2 video I-frames with > 80 % pixels in histogram range [$G_L = 0, G_H = 155$].	3202	74.85
3. Find MPEG-2 video I-frames with > 80 % pixels in histogram range [$B_L = 0, B_H = 155$].	3071	75.11
4. Find MPEG-2 video I-frames with < 80 % pixels in histogram range [$R_L = 0, R_H = 155$].	2574	74.69
5. Find MPEG-2 video I-frames with < 80 % pixels in histogram range [$G_L = 0, G_H = 155$].	2242	74.38
6. Find MPEG-2 video I-frames with < 80 % pixels in histogram range [$B_L = 0, B_H = 155$].	2366	75.09
7. Find MPEG-2 video I-frames with between 70% pixels and 90% pixels in histogram range [$R_L = 0, R_H = 155$].	1737	73.80
8. Find MPEG-2 video I-frames with between 70% pixels and 80% pixels in histogram range [$G_L = 0, G_H = 155$].	1490	73.16
9. Find MPEG-2 video I-frames with between 70% pixels and 80% pixels in histogram range [$B_L = 0, B_H = 155$].	1683	73.75
10. Find MPEG-2 video I-frames with R-intensity = 127 whose fraction > 0.34 % of the total.	2422	0.35
11. Find MPEG-2 video I-frames with G-intensity = 127 whose fraction > 0.34 % of the total.	2418	0.35

Table 3.2 (part 2 of 2) Single histogram range search results.

Query	No. of I-frames found	Time required(s)
12. Find MPEG-2 video I-frames with B-intensity = 127 whose fraction > 0.34 % of the total.	1634	0.24
13. Find MPEG-2 video I-frames with R-intensity = 127 whose fraction < 0.34 % of the total.	2962	0.43
14. Find MPEG-2 video I-frames with G-intensity = 127 whose fraction < 0.34 % of the total.	2969	0.43
15. Find MPEG-2 video I-frames with B-intensity = 127 whose fraction < 0.34 % of the total.	3749	0.54
16. Find MPEG-2 video I-frames with R-intensity = 127 whose fraction between 0 % and 0.48 % of the total.	3675	0.53
17. Find MPEG-2 video I-frames with G-intensity = 127 whose fraction between 0 % and 0.48 % of the total.	3731	0.54
18. Find MPEG-2 video I-frames with B-intensity = 127 whose fraction between 0 % and 0.48 % of the total.	4393	0.64

The colour space tree took 5491 CPU seconds (approximately 140 hours by the clock on the wall) to construct for 300 I-frames on jupiter. We made a decision to limit the colour space tree to this number due to the time required. Figure 3.11 shows an example of colour space tree testing results based on range query "Find MPEG-2 video I-

frames with > 50% pixels in colour range [R_L= 0, R_H = 132], [G_L = 0, G_H = 213], [B_L = 150, B_H = 255], simultaneously ". It took 2 seconds to perform this range search on jupiter. The time is the CPU time for one run of the range search program.

```
202 /homes/g/g01hy/somato/I_frames/wf151/rec104.tga 65983 84480
The pixel% = 78.1049%
203 /homes/g/g01hy/somato/I_frames/wf151/rec119.tga 61967 84480
The pixel% = 73.3511%
204 /homes/g/g01hy/somato/I_frames/wf151/rec134.tga 61982 84480
The pixel% = 73.3688%
205 /homes/g/g01hy/somato/I_frames/wf151/rec14.tga 70315 84480
The pixel% = 83.2327%
206 /homes/g/g01hy/somato/I_frames/wf151/rec149.tga 67568 84480
The pixel% = 79.9811%
207 /homes/g/g01hy/somato/I_frames/wf151/rec164.tga 76048 84480
The pixel% = 90.0189%
208 /homes/g/g01hy/somato/I_frames/wf151/rec179.tga 77556 84480
The pixel% = 91.804%
209 /homes/g/g01hy/somato/I_frames/wf151/rec194.tga 69535 84480
The pixel% = 82.3094%
210 /homes/g/g01hy/somato/I_frames/wf151/rec209.tga 64414 84480
The pixel% = 76.2476%
211 /homes/g/g01hy/somato/I_frames/wf151/rec224.tga 60264 84480
The pixel% = 71.3352%
212 /homes/g/g01hy/somato/I_frames/wf151/rec239.tga 66161 84480
The pixel% = 78.3156%
213 /homes/g/g01hy/somato/I_frames/wf151/rec254.tga 65135 84480
The pixel% = 77.1011%
214 /homes/g/g01hy/somato/I_frames/wf151/rec269.tga 65069 84480
The pixel% = 77.023%
215 /homes/g/g01hy/somato/I_frames/wf151/rec284.tga 69814 84480
The pixel% = 82.6397%
216 /homes/g/g01hy/somato/I_frames/wf151/rec29.tga 71487 84480
The pixel% = 84.62%
217 /homes/g/g01hy/somato/I_frames/wf151/rec299.tga 67787 84480
The pixel% = 80.2403%
218 /homes/g/g01hy/somato/I_frames/wf151/rec314.tga 74705 84480
The pixel% = 88.4292%
219 /homes/g/g01hy/somato/I_frames/wf151/rec329.tga 58379 84480
The pixel% = 69.1039%
220 /homes/g/g01hy/somato/I_frames/wf151/rec344.tga 68262 84480
The pixel% = 80.8026%
221 /homes/g/g01hy/somato/I_frames/wf151/rec359.tga 66866 84480
The pixel% = 79.1501%
222 /homes/g/g01hy/somato/I_frames/wf151/rec374.tga 67617 84480
The pixel% = 80.0391%
......
```

Figure 3.11 An example result of the colour space tree testing.

Figure 3.12 shows an example of one I-frame that is reported in range as listed in Figure 3.10. It is located in /homes/g/g01hy/somato/I_frames/wf151/ and named "rec104.tga". There are 65983 pixels out of total 84480 in the given range based on range query "Find MPEG-2 video I-frames with > 50% pixels in colour range $[R_L = 0, R_H = 132]$, $[G_L = 0, G_H = 213]$, $[B_L = 150, B_H = 255]$, simultaneously ". Figure 3.13 shows a representation of this colour range. The pixel% is calculated as 78.1049%.

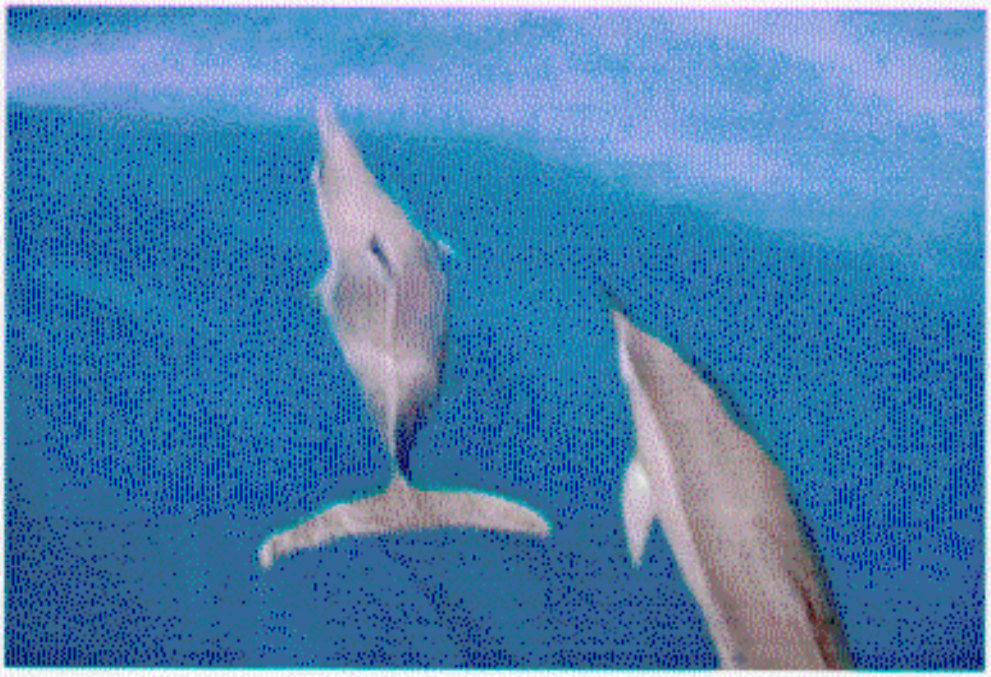


Figure 3.12 An example I-frame whose pixel% is greater than 50% in the given colour range ($R_L = 0, R_H = 132, G_L = 0, G_H = 213, B_L = 150, B_H = 255$).

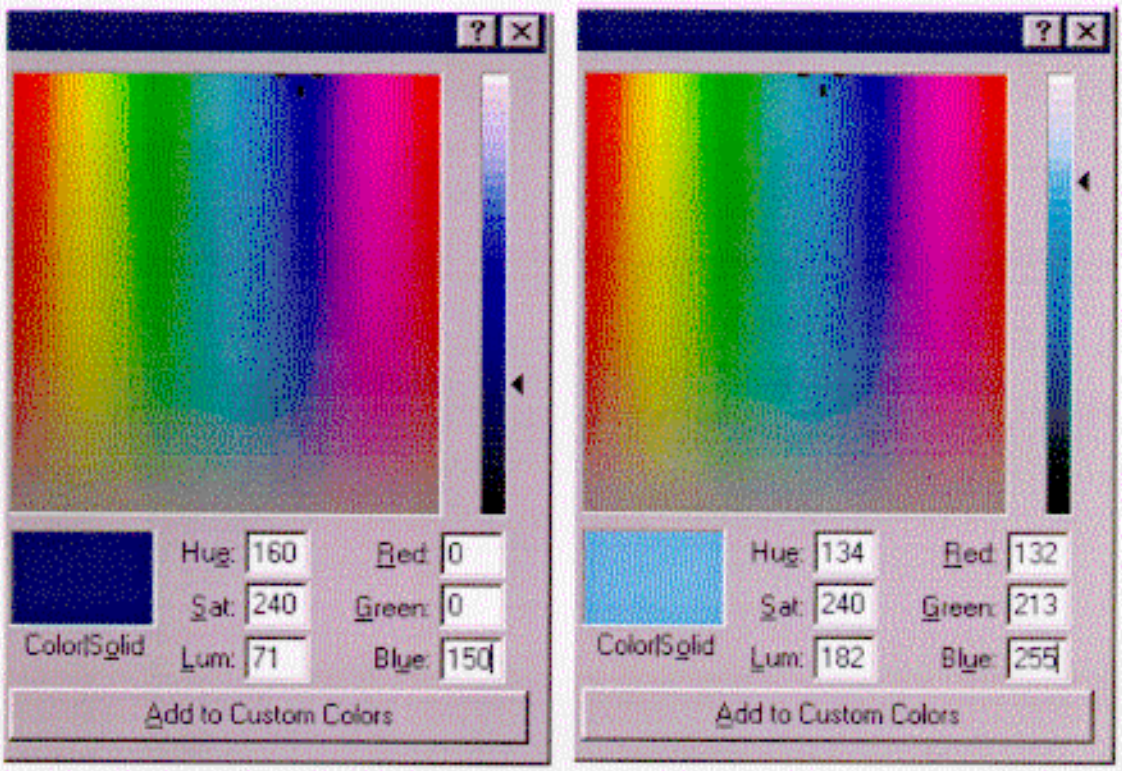


Figure 3.13 A presentation of the colour range (R_L= 0, R_H = 132, G_L = 0, G_H = 213, B_L = 150, B_H = 255).

Chapter 4 CONCLUSIONS AND FUTURE WORK

4.1 Summary

In this thesis, the application of k-d range search data structures to video data has been explored. Point-based spatial indexing methods for digital video data have been investigated. Fundamental research investigating the types of keys necessary and useful for searching in video databases has been done experimentally with many short sequences of compressed digital video in MPEG-2 format.

For content-based video indexing, we experimented with a colour histogram indexing data structure with 5706 I-frames of MPEG-2 video clips and a colour space tree indexing data structure with 300 I-frames.

We have shown the process to extract I-frames from MPEG-2 video clips, how to use a 4-d tree data structure to store and index the colour histograms of these I-frames, and allow for quick content-based searching of video data. We have presented a combination of k-d tree and hash table data structures to satisfy 18 possible queries such as "find MPEG-2 video I-frames with $> t$ % pixels in colour range $[R_L, R_H]$". We tested prototype C++ software on UNIX. It took 71 seconds to build the 4-d tree and an average of 37 seconds to perform a histogram range search on a Sun Microsystems ULTRA 5 workstation.

We also have shown another method of content-based video indexing and searching. In that method, the R,G, B values of MPEG-2 I-frames have been used as the keys to index digital video data stored in a combined k-d tree and AVL tree spatial data

structure (that we called the colour space tree) to satisfy the queries for colour space searching. A example query in this method is "Find MPEG-2 video I-frames with $> t$ % pixels in colour range $[R_L, R_H]$, $[G_L, G_H]$, $[B_L, B_H]$, simultaneously". We tested prototype C++ source code on UNIX. It required 5491 seconds to build a colour space tree of 300 I-frames and took 2 seconds to perform one range search on a Sun Microsystems ENTERPRISE 250 workstation.

The core of our solutions is their use of a low-level visual feature (colour) as a representation of content-based indexing for MPEG-2 video data. The experimental results have shown that such solutions are effective and feasible.

The contributions of the author are as follows:

(a) modified the software wrote by the MPEG Software Simulation [MSSG99] to extract only I-frames as Truevision TGA format from MPEG-2 video clips. This required one additional if statement in file store.c.

(b) developed a combination of k-d tree and hash table data structures to satisfy the colour histogram range queries.

(c) developed a combination of k-d tree and AVL data structures to satisfy the colour space range queries.

(d) wrote a program (Kdtree_RGB.cc, 3137 lines of C++ code including 214 lines of comments) to implement a 4-d tree spatial data structure to store, index, and perform range searches on the colour histograms of MPEG-2 video I-frames.

(e) wrote a program (Kdtree_Cspace.cc, 1893 lines of C++ code including 153 lines of comments) to implement a 3-d tree spatial data structure to store, index, and

perform range searches on the R,G, B intensity values of MPEG-2 video I-frames.

(f) tested all implementations in the UNIX (Sun Solaris) environment using the GNU g++ complier.

4.2 Future Work

For content-independent video indexing, we created 30 descriptive files which used text descriptions to describe 30 MPEG-2 video clips (see Appendix A). Further work remains to define an adequate data structure for fast searching of heterogeneous content-independent video keys.

The time required to search the 4-d tree using a histogram-based query is somewhat slow. Why is this? Can the search time be reduced? What are the bottlenecks? Similar future work remains for the colour space tree.

This research has given rise to the following open questions:

- What other types of content-based queries are appropriate for searching large scale video archives?
- What keys are important to be used for non-content based searching of video databases?
- What kind of adequate data structure can be used for fast searching of heterogeneous content-independent video keys?
- How can we combine content and non-content based range search for quick searching of video databases?

REFERENCES

[Adal96] Adali, S., Candan, K. S., Chen, S., Erol, K. and Subrahmanian, V. S. " The Advanced Video Information System: Data Structures and Query Processing", *Multimedia Systems,* Vol. 4, No.4, 1996, pp. 172-186.

[Adje97] Adjeroh, D. A. and Nwosu, K. C. "Multimedia Database Management Systems - Requirements and Issues", *IEEE MultiMedia,* Vol. 4, No. 3, July - September 1997, pp 24-33.

[Ardi96a] Ardizzone, E., Casia, M. La, Gesú, V. Di and Valenti, C. "Content-Based Indexing of Image and Video Databases by Global and Shape Features", Proc. of 13th International Conference on Pattern Recognition, Vienna, Austria, 1996, pp. 140-144.

[Ardi96b] Ardizzone, E., Casia, M. La, and Molinelli, D. "Motion and Colour-Based Video Indexing and Retrieval", Proc. of 13th International Conference on Pattern Recognition, Vienna, Austria, 1996, pp. 135-139.

[CCIR86] "Encoding Parameters of Digital Television for Studios", *CCIR* Recommendation 601-1 XVIth Plenary Assembly Dubrovnik , Vol. XI, Part 1, 1986, pp. 319-328.

[Davi93] Davis, M. "Media Stream: An Iconic Visual Language for Video Annotation", *Telektronikk*, Vol. 89, No. 4, 1993. pp. 59-71.

[Demon99] http://www.gedanken.demon.co.uk or ftp.demon.co.uk:/pub/unix/tools/.

[Dim95] Dimitrova, N. and Golshani, F. "Motion Recovery for Video Content Classification", *ACM Transactions on Information Systems*, Vol. 13, No. 4, October 1995, pp. 408-439.

[Flic95] Flickner, M., Sawhney, H., Niblack, W., Ashley, J., Huang, Q., Dom, B., Gorkani, M., Hafner, J., Lee, D., Petkovic, D., Steele, D. and Yanker, P. "Query by Image and Video Content: the QBIC System", *IEEE Computer*, 28(9):23-32, September 1995.

[Fole95] Foley, J. D., Van Dam, A., Feiner, S. K. and Hughes, J. F., *Computer Graphics Principles and Practice,* Addison-Wesley Publishing Company, Inc. 1995.

[Furh95] Furht, B., Smoliar, S. W., and Zhang, H. *Video and image processing in multimedia systems,* Hingham, MA, Kluwer Academic Publishers, 1995.

[Gon92] Gonzalez, R. C. and Woods, R. E. *Digital Image Processing,* Addison-Wesley Publishing Company, New York, 1999.

[Grin98] Gringeri, S., Khasnabish, B., Lewis, A., Shuaib,K., Egorov, R., and Basch, B. "Transmission of MPEG-2 Video Streams over ATM", *IEEE MultiMedia,* Vol. 5, No. 1, January/March 1998, pp. 58-71.

[Hask97] Haskell, B. G., Puri, A., and Netravili, A. N. *Digital Video: An Intruduction to MPEG-2,* Chapman & Hall, New York, 1997.

[ISO93] "Coding of Moving Pictures and Associated Audio for Digital Storage Media at Up To About 1.5 Mbit/s", International Standard ISO/IEC 11172, International Organization for Standardization, Geneva, Switzerland, 1993.

[ISO94] "Generic Coding of Moving Pictures and Associated Audio (MPEG-2)", International Standard ISO/IEC 13818, International Organization for Standardization, Geneva, Switzerland, 1994.

[Mitc97] Mitchell, Joan L., Pennebaker, William B., Fogg, Char E., and LeGall, Didier J. *MPEG-2 Video Compression Standard,* Chapman & Hall, New York, 1997.

[Mitk98] Mitkas, P. A., Betzos, G. A., and Irakliotis, L. J. " Optical Processing Paradigms for Electronic Computers", *IEEE Computer*, Feb. 1998, pp. 45-51.

[MSSG99] MPEG Software Simulation group (MSSG)
home page http://www.mpeg.org/MSSG/.

[Pent94] Pentland, A., Picard, R. W., and Sclaroff, S. "Photobook: Tools for Content-Based Manipulation of Image databases", Proc. IS and T/SPIE Conf. on Storage and Retrieval of Image and Video Databases II, San Jose, California, 1994, pp. 34-47.

[Prit77] Pritchard, D, H, "U. S. Colour Television Fundamental–A Review", *IEEE Transactions on Consumer Electronic*, CE-23(4), November 1977, pp. 467- 478.

[Rao90] Rao, K. R. and Yip, P. *Discrete Cosine Transform: Algorithms, Advantages, Applications,* Academic Press, New York, 1990.

[Rowe94] Rowe, L. A., Boreczky, J. S. and Eads, C. A. "Indexes for User Access to Large Video Databases", Proc. IS and T/SPIE Conf. on Storage and Retrieval for Image and Video Databases II , San Jose, California, 1994, pp. 150-161.

[Same90a] Samet, H. *The Design and Analysis of Spatial Data Structures,* Addison-Wesley, Reading, MA, 1990.

[Same90b] Samet, H. *Applications of Spatial Data Structures,* Addison-Wesley, Reading, MA, 1990.

[Soml94] Smoliar, S. W. and Zhang, H. "Content-Based Video Indexing and Retrieval", *IEEE Multimedia,* Vol. 1, No 2, 1994, pp. 62-72.

[Vau96] Vaughan, T. *Multimedia: Making It Work*, Osborne McGreew-Hill, 1996.

[Weeks96] Weeks, Jr., Arthur R. *Fundamentals of Electronic Image Processing*, SPIE-The International Society for Optical Engineering, Bellingham, Washington USA, 1996.

[Zhan95a] Zhang, H., Tan, S. Y., Smoliar, S. W. and Yihong, G. "Automatic Parsing and Indexing of News Video", *Multimedia Systems,* Vol. 2, No. 5 , 1995. pp. 256-266.

[Zhan95b] Zhang, H. J., Low, C.Y., Smoliar, S.W. and Wu, J. H. "Video parsing, Retrieval and Browsing: An Integrated and Content-Based Solution", In Proc. of the ACM Multimedia Conference, San Francisco, California, 1995, pp. 15-24.

Appendix A

30 MPEG-2 Video Clip Text Descriptive Files

File Name: 512av.txt
source http://www.bernclare.com/sample.htm#samplelist
date 1998 08 31
duration 00 11
keywords movie
path /home/grads/g01hy/mpeg/512av.mpg

File Name: akl.txt
source http://www.deepsouth.co.nz/~eggplant/MPEG.htm
date 1998 09 02
duration 00 05
keywords scene
path /home/grads/g01hy/mpeg/akl.mpg

File Name: b2future.txt
source http://www.bernclare.com/sample.htm#fun
date 1998 08 27
duration 00 17
keywords movie
path /home/grads/g01hy/mpeg/b2future.mpg

File Name: bird.txt
source http://www.imagemind.com/library_download.htm#mpeg
date 1998 09 01
duration 00 13
keywords movie
path /home/grads/g01hy/mpeg/bird.mpg

File Name: c-saw.txt
source http://www.imagemind.com/library_download.htm#mpeg
date 1998 07 01
duration 00 05
keywords movie
path /home/grads/g01hy/mpeg/c-saw.mpg

File Name: camera.txt
source http://www.imagemind.com/library_download.htm#mpeg
date 1998 09 01
duration 00 07
keywords movie
path /home/grads/g01hy/mpeg/camera.mpg

File Name: creature.txt
source http://www.imagemind.com/library_download.htm#mpeg
date 1998 09 04
duration 00 32
keywords movie
path /home/grads/g01hy/mpeg/creature.mpg

File Name: crissy.txt
source http://www.netkitchen.com/windsurf/crissy.htm
date 1989 08 09
duration 00 04
keywords sailing
path /home/grads/g01hy/mpeg/crissy.mpg

File Name: crissy2.txt
source http://www.netkitchen.com/windsurf/crissy.htm
date 1989 08 09
duration 00 04
keywords sailing
path /home/grads/g01hy/mpeg/crissy2.mpg

File Name: destruct.txt
source http://www.imagemind.com/library_download.htm#mpeg
date 1998 09 10
duration 00 06
keywords movie
path /home/grads/g01hy/mpeg/destruct.mpg

File Name: drill1.txt
source http://www.occuphealth.fi/eng/dept/u/spteam/tools/
date 1998 09 02
duration 00 04
keywords advertisement
path /home/grads/g01hy/mpeg/drill1.mpg

File Name: drill2.txt
source http://www.occuphealth.fi/eng/dept/u/spteam/tools/
date 1998 09 02
duration 00 04
keywords advertisement
path /home/grads/g01hy/mpeg/drill2.mpg

File Name: fantasia.txt
source http://www.bernclare.com/sample.htm#fun
date 1998 08 31
duration 01 04
keywords film
path /home/grads/g01hy/mpeg/fantasia.mpg

File Name: gump.txt
source http://www.bernclare.com/sample.htm#samplelist
date 1998 09 01
duration 00 19
keywords movie
path /home/grads/g01hy/mpeg/gump.mpg

File Name: hammer.txt
source http://www.occuphealth.fi/eng/dept/u/spteam/tools/
date 1998 09 02
duration 00 05
keywords advertisement
path /home/grads/g01hy/mpeg/hammer.mpg

File Name: hell.txt
source http://www.imagemind.com/library_download.htm#mpeg
date 1998 09 01
duration 00 09
keywords movie
path /home/grads/g01hy/mpeg/hell.mpg

File Name: hookip.txt
source http://www.netkitchen.com/windsurf/hokipa.htm
date 1995 08 09
duration 00 07
keywords sailing
path /home/grads/g01hy/mpeg/hookip.mpg

File Name: hookip2.txt
source http://www.netkitchen.com/windsurf/hokipa.htm
date 1995 08 09
duration 00 03
keywords sailing
path /home/grads/g01hy/mpeg/hookip2.mpg

File Name: hur.txt
source http://www.deepsouth.co.nz/~eggplant/MPEG.htm
date 1998 09 02
duration 00 05
keywords scene
path /home/grads/g01hy/mpeg/hur.mpg

File Name: isdndual.txt
source http://www.bernclare.com/sample.htm#samplelist
date 1998 08 31
duration 00 09
keywords movie
path /home/grads/g01hy/mpeg/isdndual.mpeg

File Name: jel_fish.txt
source http://www.imagemind.com/library_download.htm#mpeg
date 1998 09 01
duration 00 14
keywords movie
path /home/grads/g01hy/mpeg/jel_fish.mpg

File Name: jet.txt
source http://www.imagemind.com/library_download.htm#mpeg
date 1998 09 04
duration 00 36
keywords movie
path /home/grads/g01hy/mpeg/jet.mpg

File Name: mvcdsmpl.txt
source http://www.bernclare.com/sample.htm#fun
date 1998 08 01
duration 00 05
keywords movie
path /home/grads/g01hy/mpeg/mvcdsmpl.mpg

File Name: napier.txt
source http://www.deepsouth.co.nz/~eggplant/MPEG.htm
date 1998 09 02
duration 00 05
keywords scene
path /home/grads/g01hy/mpeg/napier.mpg

File Name: port.txt
source http://www.deepsouth.co.nz/~eggplant/MPEG.htm
date 1998 09 02
duration 00 05
keywords scene
path /home/grads/g01hy/mpeg/port.mpg

File Name: producer.txt
source http://www.bernclare.com/sample.htm#showsite
date 1998 08 01
duration 00 08
keywords movie
path /home/grads/g01hy/mpeg/producer.mpg

File Name: ski.txt
source http://www.deepsouth.co.nz/~eggplant/MPEG.htm
date 1998 09 06
duration 00 05
keywords scene
path /home/grads/g01hy/mpeg/ski.mpg

File Name: skybird.txt
source http://www.imagemind.com/library_download.htm#mpeg
date 1998 10 01
duration 00 09
keywords movie
path /home/grads/g01hy/mpeg/skybird.mpg

File Name: wf174.txt

```
source   http://www.mpegcam.net/MPEG/ThatGuy.html
date     1998 08 31
duration 00 06
keywords scene
path     /home/grads/g01hy/mpeg/wf174.mpg
```

File Name: wrench.txt

```
source   http://www.occuphealth.fi/eng/dept/u/spteam/tools/
date     1998 09 02
duration 00 05
keywords advertisement
path     /home/grads/g01hy/mpeg/wrench.mpg
```

VITA

Candidate's full name: Enhai Xie

Universities Attended :

Hebei University, P. R. China, (1978-1982)
B.Sc. (Analytical Chemistry)

Hebei University, P. R. China, (1987-1990)
M.Sc. (Physical Chemistry)

University of Oldenburg, Germany, (1990-1991)
M.Sc. (Physics)

University of New Brunswick, Canada, (1998-2000)
Master of Computer Science
Candidate

Publications:

Conference Presentations:

Enhai Xie and B.G. Nickerson (Supervisor), "Spatial data Structure Indexing for Multimedia Databases", Proceeding of the 8th Atlantic Institute Student Research Annual Conference, 20th-21st February, 1998, University of New Brunswick, Fredericton, New Brunswick, Canada, p.12.

Enhai Xie and B.G. Nickerson (Supervisor), "Content-based Searching of MPEG-2 video", Proceeding of the 9th Atlantic Institute Student Research Annual Conference, 8th-9th June, 1999, Université Laval, Québec city,Québec, p.21.