2004

Real-time video compression using DVQ and suffix trees

Pavan Kumar Vedantam
West Virginia University

Real Time Video Compression using DVQ and Suffix Trees

Pavan Kumar Vedantam

Thesis submitted to the college of Engineering and Mineral Resources at West Virginia University
In Partial fulfillment of the requirements for the degree of

Master of Science
In
Electrical Engineering

Donald Adjeroh, Ph.D., Chair
Hany Ammar, Ph.D.
James Mooney, Ph.D.

Lane Department of Computer Science and Electrical Engineering,

Morgantown, West Virginia

2004

Keywords: Real time video, compression, VQ, DVQ, suffix trees

ABSTRACT

Real Time Video Compression using DVQ and Suffix Trees

Pavan Kumar Vedantam

Video processing is a wide and varied subject area. Video compression is an important but difficult problem in video processing. Several methods and standards exist which address this problem with varying degrees of success depending on the performance measures adopted. The present research work focuses on the real- time aspect of video processing.

In particular we propose a real- time video compression algorithm based on the concept of differential vector quantization and the suffix tree. Differential vector quantization is a relatively new area that focuses on efficient compression of data. The present work integrates the compression provided by Differential vector Quantization and the speed achieved by using the suffix tree data structure to develop a new real-time video compression scheme.

Traditionally Suffix trees are used for string searching. In the present work, we exploit the unique structure of the suffix tree to represent image data on a tree as a DVQ dictionary. To support the special characteristics of natural images and video, the traditional suffix tree is extended to handle *k*-errors in the matching. The result is an orders of magnitude speedup in the matching process, making it possible to compress the video in real-time, without any special hardware.

Experimental results show the performance of the proposed methodology

Dedicated to my parents and philosopher Jiddu Krishnamurthy

Table of Contents

List of Figures

List of Tables

Acknowledgement

I am grateful to Dr Donald Adjeroh for giving me an opportunity to conduct research under his aegis. Without his knowledge, guidance, and above all his perseverance, this thesis could not have been tangible.

I am indebted to my other committee members Dr James Mooney, Dr Hany Ammar for their support and guidance throughout the project.

Without the moral support provided to me by my parents Mr. Shiv Shanker Vedantam, Mrs. Jammalmadaka Samantakamani life would have been an insurmountable task. Their unique perspective of life has been the most valuable lesson in my life.

I would also take this opportunity to thank my friends whose company has been the most cherishable moments in life. Udayabhanu Pattabhi Ram Chamarthy, Sarvesh Makthal who has contributed his valuable time in helping me out during tough phases while conducting research, Murali Mohan Gadde ,Vamsikrishna Nadella and others have been along my side during my stay in Morgantown.

Above all I am thankful to West Virginia University for giving me an opportunity to conduct research. Go Mountaineers !!!

1. Introduction

Video processing is a fast and emerging technology. The need for fast and reliable video has been spawned by the internet. Video is finding applications in several areas such as security surveillance, video over the internet, and video in embedded applications such as cell phones, most of the applications have limited bandwidth and cannot handle the high amounts of data that is common in video. Thus these applications demand high performance video codec's to represent the video data in a reduced data size and to decompress video. Importantly, most of these applications require the encoding or decoding to be performed in real time.

Video is a moving picture accompanied with audio, thus video compression will comprise the compression of both image and audio data. Several methods and standards exist which address this problem with varying degrees of success depending on the performance measures adopted. The present research work focuses on the real- time aspect of video processing. Real time video compression can be defined as compression of video sequences as they appear in real time, i.e. 28 frames per second.

Real time video compression can be accomplished in either of the following two ways,

a. Software implementation
b. Hardware implementation

For software Implementation real time video compression is achieved by a software program on an operating system which implements the video algorithm in real time. The term software is used here in a very broad sense. It implies the implementation of the video codec on a software platform. Examples of software methods are

a. JPEG, b. MPEG, c. H.261

Under hardware Implementation

Compression of video in real time is achieved by employing the video codec over electronic hardware. Because of the inherent speed of hardware, the video codec over hardware perform better than that of software in terms of real time processing. Few of hardware codec's are presented below

a. Real-time MPEG Video Compression using the MVP [25]
b. Real time video codec on a single chip multiprocessor [26]
c. Low Complexity Single-Chip VLSI Implementation [27]

Video codec have been developed in both hardware and software with emphasis on the quality and real time aspect of the codec. Our goal is to develop a software algorithm that achieved real time results on a PC. This thesis presents a real time video compression algorithm based on the concept of differential vector quantization (DVQ) and the suffix tree. The algorithm exploits the unique structure of the suffix tree for real time results. DVQ is an extension of vector quantization to differences whereby difference vectors rather then the original vectors are quantized, hence the term "differential vector quantization." A new concept of temporal DVQ is introduced, whereby the differential vector quantization is performed along the temporal axis. This scheme provides greater compression and speed than the more traditional spatial DVQ.

In the next chapter we introduce video compression techniques, chapter 3 describes suffix trees, real time video compression using our proposed method is described in chapter 4, and in chapter 5 we present our results and conclude the thesis in chapter 6.

2. Video Compression

Video Compression is the technique of compressing the video data for efficient storage and transmission purposes. According to Shannon [1], entropy $H(S)$ of a given sequence S is defined as, $H(S) = \eta = \sum_i p_i \log_2 \frac{1}{p_i}$

p_i is the probability of occurrence of the i^{th} intensity level. Entropy is a measure of randomness or the amount of information in a sequence. It also represents the minimum bits per symbol required to represent the sequence. The aim of video compression therefore is, given an input video sequence, *S,* to represent it as compactly as possible, i.e. to represent it with *L(S)* number of bits per pixel such that *L(S)* is as close to $H(S)$ as possible.

Compression in general is of two types, lossy Compression and lossless Compression.
Lossless and lossy compression are terms that describe whether or not, in the compression of a file, all original data can be recovered when the file is decompressed. With lossless compression, every single bit of data that was originally in the file remains after the file is decompressed. All of the information is completely restored. This is generally the technique of choice for text or spreadsheet files, where losing words or financial data could pose a problem. The Graphics Interchange File (GIF) is an image format used on the Web that provides lossless compression for images.

On the other hand, lossy compression reduces a file by permanently eliminating certain information, especially redundant information. When the file is decompressed, only a part of the original information is still there (although the user may not notice it). Lossy compression is generally used for video and sound, where a certain amount of information loss will not be detected by most users.

2.1 Lossless Compression

Several techniques have been proposed for lossless compression, these include,

a. Entropy coding (ex. Huffman coding, arithmetic coding)
b. Run length encoding
c. Predictive techniques (DPCM)

2.1.1 Huffman Coding

Huffman [4] coding is based on the frequency of occurrence of a data item (pixel in images). The principle is to use a lower number of bits to encode the data that occurs more frequently. Codes are stored in a ***Code Book*** which may be constructed for each image or a set of images. In all cases the code book plus encoded data must be transmitted to enable decoding. The procedure for developing Huffman code is given below,

1. Begin by creating a table for the symbols, order the symbols in decreasing order of probability.
2. Create a binary tree. Build the tree by merging two lowest probability symbols at each level by making the probability of the symbol equal to the sum of the merged symbols probabilities. If more than two symbols share the lowest probabilities choose any two.
3. On every symbol on the tree , label the emerging branches with a binary number. The bit sequence obtained by passing from the root to the symbol location is its Huffman code.

By following the Huffman coding procedure, if we wish to encode the letters A, E, I, O whose probabilities of occurrence are 0.12, 0.42, 0.09, 0.30, 0.07, the Huffman code for the above letters would be,

A - 100, E - 0, I – 1011, O – 11, U - 1010

Using the above code, any given sequence of vowels can be decoded uniquely.

2.1.2 DPCM

Differential pulse code modulation (DPCM) is a procedure of converting an analog into a digital signal in which an analog signal is sampled and then the difference between the actual sample value and its predicted value (predicted value is based on previous sample or samples) is quantized and then encoded forming a digital value. DPCM code words represent differences between samples unlike PCM where code words represented a sample value. The concept of DPCM - coding a difference, is based on the fact that most source signals show significant correlation between successive samples so encoding uses redundancy in sample values which implies a lower bit rate.

Realization of the basic concept (described above) is based on a technique in which we have to predict current sample value based upon previous samples (or sample) and we have to encode the difference between the actual sample value and the predicted value (the difference between samples can be interpreted as prediction error).See Fig 1.0

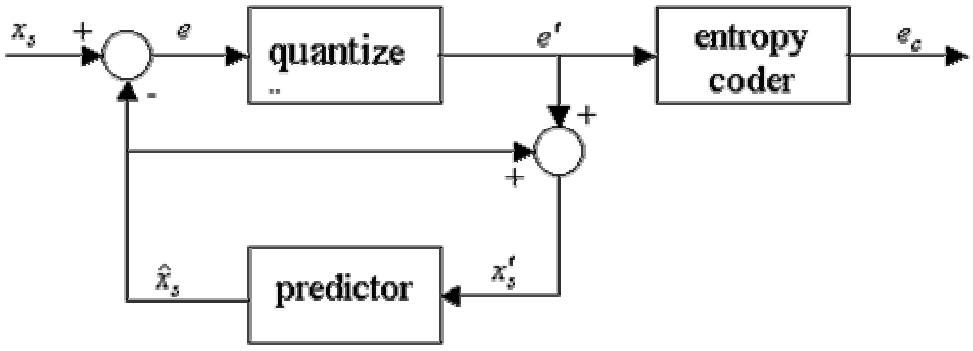


Figure 1 : DPCM Encoder

x_s - Sampled value of input signal $\hat{x}_s$ - predicted value e_c - Value after DPCM coding (input value for DPCM decoding)

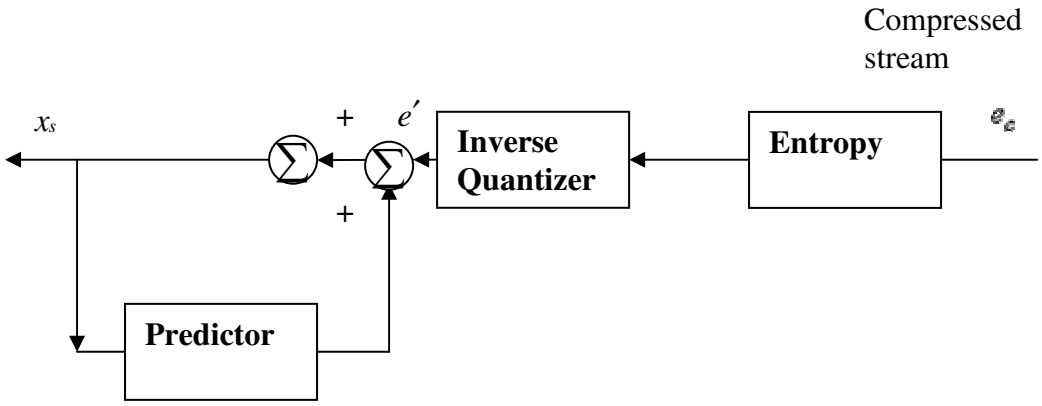


Figure 2: DPCM Decoder

Because it's necessary to predict the sample value, DPCM can be seen as a form of predictive coding. Thus the performance of DPCM compression depends on the prediction technique. An effective prediction technique will lead to a good compression rate. When the predictor is not effective DPCM could lead to data expansion rather than data compression.

In Fig 1.0 and Fig 2, x_s is the current pixel value and $\hat{x}_s$ is formed using p prior pixels $\hat{d}$ is differential image formed as difference between the actual pixel and previous pixels

It is important to point out that in forming a decoder, it has access only to reconstructed pixel values. Since the process of quantization of differential image introduces error, the reconstructed values as expected to diverge from the original values.

The key problem in the DPCM system is optimizing the predictor and quantizer components. Because the quantizer is included in the prediction loop, there is a complex dependency between the prediction error and quantization error. Thus joint optimization should be performed to ensure optimal results. But, modeling such optimizations is very complicated, so optimization of those two components is usually performed separately. It has been shown that under the mean-squared error optimization criterion, separate constructions of quantizer and predictor are good approximations of the joint optimization [1, 2]

2.2 Lossy Compression

The most commonly used methods for lossy compression are, transform coding and, vector quantization. Below we describe the general technique of transform coding. Vector quantization is described in detail in Chapter 4.

2.2.1 Transform Coding

Transform coding is a popular method for data compression. In transform coding of an image, a linear transformation is applied to the image to decompose it into frequency coefficients. The coefficients are quantized and the quantized values are then entropy coded to produce the compressed sequence. This has the advantage that the resulting coefficients have a statistically significant distribution and can be modeled and compressed more easily.

The desired effect is that most of the energy in the image will be contained in a few large transform coefficients. If it is generally the same few coefficients that contain most of the energy in most pictures, then the coefficients may be further coded by lossless entropy coding . In addition, it is likely that the smaller coefficients can be coarsely quantized or deleted (lossy coding) without much degradation in the reproduced image.

2.3 Real Time Video Compression Algorithms

Real time vide compression is a method of video compression that achieves compression and decompression of video in real time. For example MPEG uses 30 frames per second to render real time video, i.e. in order for this video to render real time effects it needs to compress and decompress 30 images per second, failure to do so will result in loss of data and video will be out of synchronization. Thus real time analysis of video is a critical component of a real-time video compression algorithm.

The present video compression algorithms are, a. H.261, b. H.263 ,c. H.263+ ,d. MPEG ,e. MPEG-1 ,f. MPEG-2 ,g. MPEG-4 ,h. MPEG-7 ,i. MPEG -21,j. J.81

We describe these briefly in the following,

2.3. 1 The H Series

H.261

H.261 [9] was developed for transmission of video at a rate of multiples of 64Kbps. Videophone and videoconferencings are some applications. H.261 standard is similar to JPEG still image compression standard. H.261 uses motion-compensated temporal prediction. H.261 coder has a layered structure with 4 layers. The 4 layers are picture layer, group of block (GOB) layer, macro block (MB) layer and block layer. Each block is 8x8, and the layers are multiplexed for transmission in series. Each layer has a header. The Frame format of H.261 is called the common intermediate format (CIF).

H.263

H.263 uses block motion-compensated DCT structure for encoding. H.263 encoding has higher efficiency than h.261 encoding. An encoding specification called test model (TMN) was used for optimization in H.263. There are different versions of test models,

the latest version is called TMN5. H.263 is based on H.261 but it is significantly optimized for coding at low bit rates. Video coding is performed by partitioning each picture into macro blocks. Each macro block consists of 16x16 luminance block and 8x8 chrominance blocks of Cb and Cr. Each macro block can be coded as intra or as inter. spatial redundancy is exploited by DCT coding, temporal redundancy is exploited by motion compensation. H.263 includes motion compensation with half-pixel accuracy and bidirectional coded macro blocks. 8x8 overlapped block motion compensation, unrestricted motion vector range at picture boundary, and arithmetic coding are also used in H.263.

H.263+

H.263+ [11] is an extension of H.263. It has several additional features and negotiable additional modes. It provides SNR scalability as well as spatial and temporal scalability. It has custom source formats. Advanced intra coding is used to improve the compression efficiency for intra macro block encoding by using spatial prediction of DCT coefficient values. Deblocking filter mode reduces the amount of block artifacts in the final image by filtering across the block boundaries using an adaptive filter. Slice structure allows a functional grouping of a number of macro blocks in the picture, enabling improved error resilience, improved transport over packet networks and reduced delay. Reference picture resampling mode allows a resampling of a temporally previous reference picture prior to its use as a reference for encoding, enabling global motion compensation, predictive dynamic resolution conversion, predictive picture area alteration and registration and special-effect warping. Reduced resolution update mode allows an encoder to maintain a high frame rate during heavy motion by encoding a low-resolution update to a higher resolution picture while maintaining high resolution in stationary areas. Independent segment decoding mode enhances error resilience by ensuring that corrupted data from some region of the picture cannot cause propagation of error into other regions. Modified quantization mode improves the bit rate control by changing the method for controlling the quantizer step size on a macro block basis. This also reduces the prevalence of chrominance artifacts by reducing the step size for chrominance quantization and

increases the range of representable coefficient values for use with small quantizer step sizesH.263+ improves error detection performance and reduces decoding complexity by prohibiting certain unreasonable coefficient representations.

2.3.2 The MPEG series

MPEG

MPEG (Moving Picture Experts Group) [12] is an ISO/IEC working group developing international standards for compression, decompression, and representation of moving pictures and audio

MPEG -1

MPEG video compression standard [11, 12] is a layered, DCT-based video compression standard that results in VHS quality compressed video stream that has a bit rate of approximately 1.5Mbps at a resolution of approximately 352x240. At a high level, MPEG video sequences consist of several different layers that provide the ability to randomly access a video sequence as well as provide a barrier against propagation of error.

MPEG -2

MPEG-1 has a bit rate of about 1.5 Mbps, MPEG-2 [13] is designed for diverse applications which require a bit rate of up to 100Mbps. Digital high-definition TV (HDTV), interactive storage media (ISM), cable TV (CATV) are sample applications. Multiple video formats can be used in MPEG-2 coding to support these diverse applications

.

MPEG -4

MPEG-4 uses media objects to represent aural, visual or audiovisual content. Media objects can be synthetic like in interactive graphics applications or natural like in digital television. These media objects can be combined to form compound media objects. Visual part of the MPEG-4 standard describes methods for compression of images and video, compression of textures for texture mapping of 2-D and 3-D meshes, compression of implicit 2-D meshes, and compression of time-varying geometry streams that animate meshes.

MPEG -7

The aim of MPEG-7 is to specify a set of descriptors to describe various forms of multimedia. It will also standardize ways to define other descriptors as well as structures for the descriptors and their relationship. This information will be associated with the content to allow fast and efficient search. MPEG-7 will also standardize a language to specify description schemes.

MPEG -21

MPEG-21 aims at defining a normative open framework for multimedia delivery and consumption for use by all the players in the delivery and consumption chain. The goal of MPEG-21 can thus be rephrased as follows: defining the technology needed to support users to exchange, access, consume, trade and otherwise manipulate Digital Items in an efficient, transparent and interoperable way. MPEG-21 is currently in construction phase.

2.3. 3 J.81

J.81 has bit rates of 34-45Mbps in the format specified by recommendation ITU-R 601. Net video capacity is between 26 and 31Mbps for Europe and depends on the number of optional channels used. J.81 provides very high quality which is suitable for transparent compression necessary for contribution applications.

The H series of standard were primarily developed for video conferencing, video telephony purposes. MPEG -2 for DVD video, MPEG-4 for video over 3G wireless. MPEG requires a higher bit rate than H series for transmission, as such the quality of the H series does not match the MPEG series.

2.4 Vector Quantization

2.4.1 Introduction

The major goal in lossy data compression is to obtain the best possible fidelity for the given rate or, equivalently, to minimize the rate required for a given fidelity. The actual quantization (i.e. conversion of continuous quantities into discrete values) is done on scalars, e.g., on individual real valued samples of waveforms or pixels values. A fundamental result of Shannon's theorems [7] is that better data compression can be achieved by coding of vectors instead of scalars. This is the case whether the data source is memory less, (i.e. Consists of a sequence of independent random variables) or whether the data compression system can have memory (i.e., the action of an encoder at each time is permitted to depend on past encoder inputs or outputs).

Data compression essentially involves the conversion of a stream of analog or very high rate discrete data into a stream of relatively low rate data for communication over a digital communication link or storage in a digital memory. Image storage and communications are a prime example of applications that require data compression where simple schemes require bit rates too large for many communication links or storage devices.

2.4.2 Vector Quantizers

Formally, an L –dimensional memory less quantizer or, simply, a VQ consists of two mappings: an encoder γ which assigns to each input vector $x = (x_0, x_1, x_2, ...x_{L-1})$ a channel symbol $\gamma(x)$ in some symbol set ξ , and a decoder β which assigns to each channel symbol v in ξ a value in a reproduction alphabet $\hat{A}$. The channel symbol set is often assumed to be a space of binary vectors for convenience e.g., ξ may be a set of all 2^R binary R-dimensional vectors. The reproduction alphabet may or may not be the same as the input vector space. In particular, it may consist of real vectors of a different dimension.

If ξ has M elements, then the quantity $R = \log_2 M$ is called the rate of the quantizer in bits per vector, and $r = \frac{R}{L}$ is the rate in bits per symbol or, when the input is a sampled waveform, bits per sample. Unlike scalar quantization, general VQ permits fractional rates in bits per sample. The symbol set ξ is the codebook, i.e. dictionary for the input vector x. In terms of video compression, the performance of VQ depends on the probability of mapping the input vector in the symbol set ξ, i.e. finding a match in the dictionary, a higher number of matches indicates the design of good dictionary and will consequently affect the quality of the image at the decoder. For the VQ algorithm to have real time performance the time required to search for a match should be minimal. This can be achieved by implementing efficient search algorithms.

Figure 3.0 shows the block diagram of the encoder and decoder of a vector quantizer.

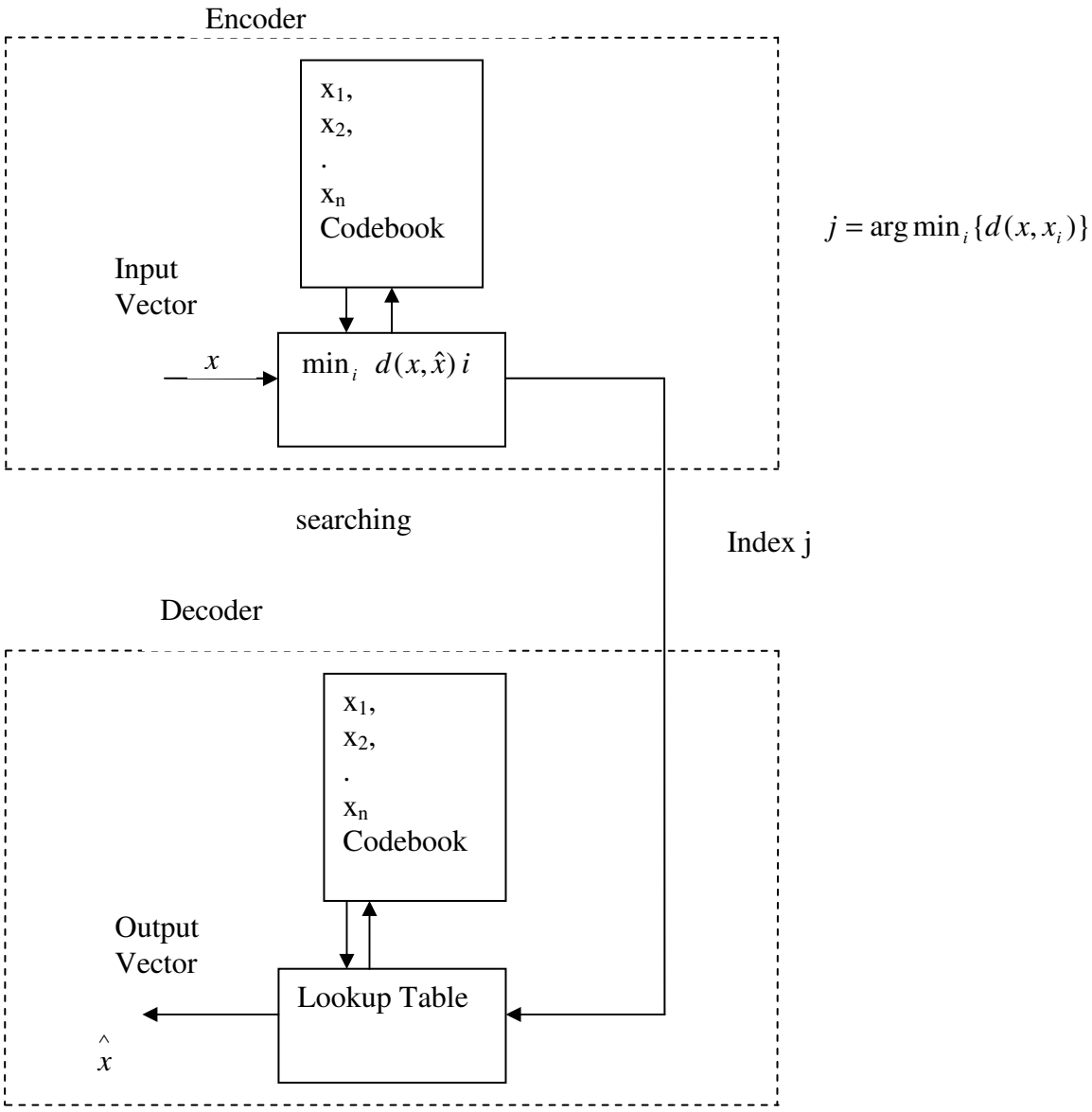


Figure 3: Block Diagram for Vector Quantization

From Figure 3.0 we can observe that the key problems in VQ are the encoding stage. The decoding stage merely involves simple lookup with the transmitted index at the encoding stage. The first problem is the generation of the codebook. That is ,out of all the possible $|\xi|^L$ number of vectors, how do we choose the n $(n << |\xi|^L)$ vectors to represent data set.

Here n is called the vocabulary or dictionary size. The second problem is, given the codebook (i.e. dictionary) $D = \{x_1, x_2, x_3, ...x_n\}$ and an input vector x how do we determine the vector that best represents x. This is a search problem. The amount of distortion between the input vector and the nearest codebook entry determine the quality of the reconstructed image.

2.4.3 Distortion

A distortion measure d is an assignment of a cost $d(x, x')$ of reproducing any input vector x as a reproduction vector x'. Given such a distortion measure, we can quantify the performance of a system by an average distortion $E\{d(X, X')\}$ between the input and final reproduction: A system will be good if it yields a small average distortion. In practice, the important average is the long-term sample average or time average,

$$d(x, x') = \frac{Lim}{n \to \infty} \sum_{i=0}^{n-1} d(x_i, x_i') - \quad (1)$$

If the vector process is stationary and ergodic, then with probability one, the limit exits and equals the expectation $E\{d(X, X')\}$ [1].

A specific example is the squared error distortion measure: Here the input and reproduction spaces are *L*-dimensional Euclidean space

$$d(x, x') = \sqrt{\sum_{j=1}^{k} (x_j - x_j')^2} \quad (2)$$

.

Average distortion is a good measure of performance of a system, the long-term sample average of (I) that is actually measured and which in a good system is expected to be small.

2.4.4 Initial Codebooks

Two approaches for selecting the initial codebooks have been outlined in [1], one can start with some simple codebook of the correct size or one can start with a simple small codebook and recursively construct larger ones. Larger codebooks would imply lesser compression, but a better quality for the reconstructed image.

Random Code: A codebook where the initial training sequence is used as the codebook. An obvious modification would be to select the words from the training sequence that are widely spaced from one another.

2.4.5 Variations of Vector Quantizers

This section focuses at reducing the computation or memory requirements of a full search VQ.

2.4.5.1 Tree searched VQ

Tree searched vectors were first proposed by Buzo et al. [1] and are a natural by product of the splitting algorithm for generating initial code guesses.
A tree searched encoder selects one of the vectors not by an ordinary full search of the codebook, but instead it uses the codebook designed on the whole sequence to select the second code and then it picks the best word in the second code. This encoder can then be used to further subdivide the training sequence and construct even better codebooks for the subsequences.

2.4.6 Feedback Vector Quantizers

Memory can be incorporated into the vector quantizer by using different codebooks for each input vector, and the codebooks can be chosen based on past input vectors. At any point the decoder must know which codebook the encoder is using in order to decode the channel symbols. This kind of codebooks are named as adaptive codebooks where depending on the variations in the input vector the video encoding algorithm can choose between different codebooks that are available. This concept is an excellent idea for rapidly changing videos.

2.5 Differential Vector Quantization

Differential vector quantization is the extension of vector quantization to pixel differences. The term "Differential vector quantization" implies that vector quantization is performed on image differences, rather than the actual pixel values. In our present research, we are concerned with the difference of images. Difference between images is approached in different ways,

a. Spatial difference (using horizontal and vertical spatial difference)
b. Temporal difference(using horizontal and vertical temporal difference)

2.5.1 Spatial Difference

Difference of pixels taken over the spatial domain and later quantized is referred to as spatial differential vector quantization. The difference of pixels is obtained using different scan methods, for example the simple horizontal snake scan will produce a set of horizontal spatial difference. Two different sets of horizontal and vertical scans were

used in obtaining the difference of the image. The following diagram shows an image in spatial domain.

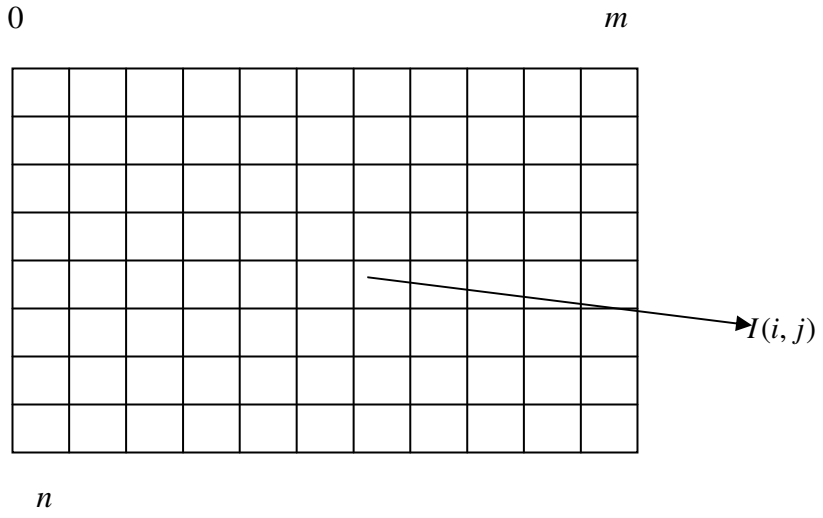


Figure 4: Representation of image in spatial domain

Let m, n the represent the size of the image in the horizontal and vertical axis respectively. Let $I(i, j)$ represent the pixel intensity at location i, j where i represent the horizontal location and j represents the vertical location in the image axis. $\hat{I}(i, j)$ represents the predicted value. For scheme A, we use the following predictor, $\hat{I}_A(i, j) = I(i-1, j)$.

The predicted value is subtracted from the actual value to form the difference of pixels. $e_A = I(i, j) - \hat{I}_A(i, j)$.

For scheme B we use the predictor $\hat{I}_B(i, j) = \frac{I(i-1, j) + I(i-1, j-1) + I(i-1, j-1)}{3}$. This is simply the average of the nearest neighboring pixels. The predicted error $e_B(i, j)$ is calculated as above.

The different scans used for spatial differential vector quantization are illustrated below,

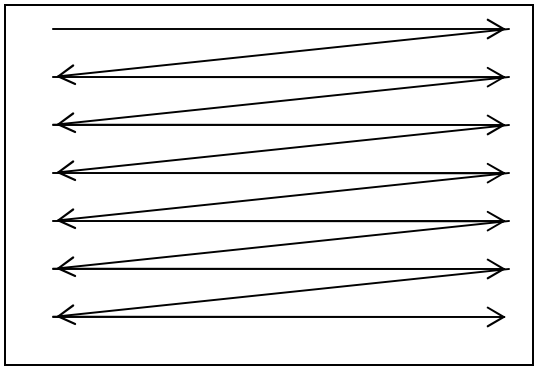


Figure 5: Horizontal scan I

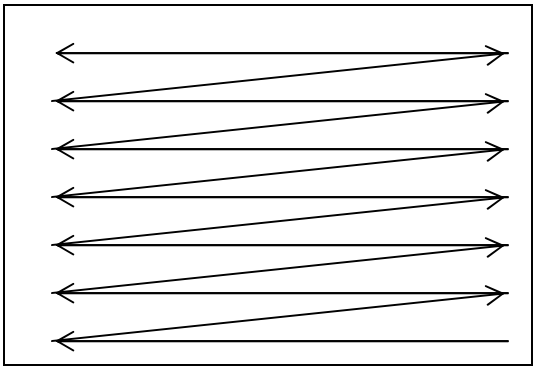


Figure 6: Horizontal scan II

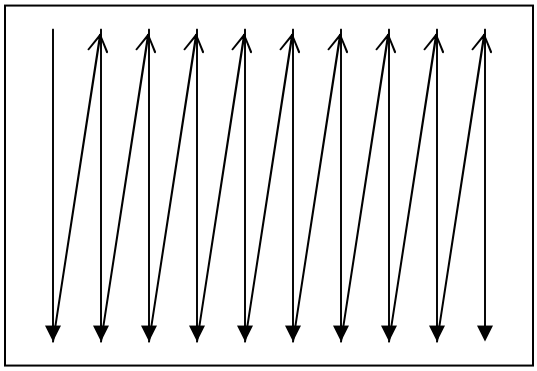


Figure 7: Vertical scan I

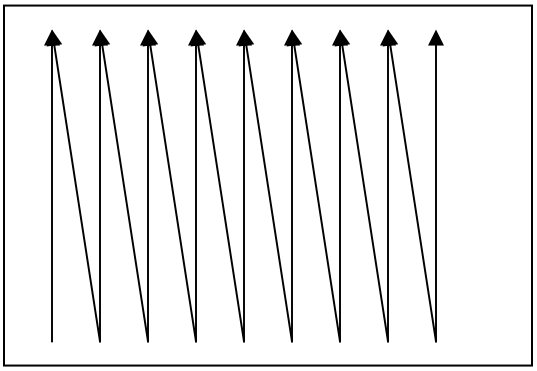


Figure 8: Vertical scan II

The different scans yield different difference images these difference images are then vector quantized and encoded to produce the compressed image.

2.5.2 Temporal Difference

Pixel differences taken over the temporal domain is known as temporal differential vector quantization. Two different schemes of temporal differences were used. The first scheme is a simple temporal difference which included a point by point by difference of pixels values over consecutive frames. The second scheme is spatio temporal differential vector quantization. In this method the spatial difference is taken over the first frame in a sequence of images and is reconstructed at the decoder. The temporal difference is taken

over consecutive images in tandem. We call two temporal differential vector quantization schemes are,

a. Plain temporal differential vector quantization.
b. Spatio temporal differential vector quantization.

The following diagram shows the concept of a plain temporal difference used in temporal differential vector quantization.

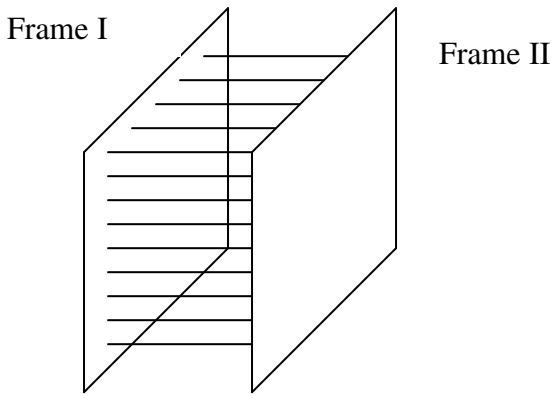


Figure 9 : Temporal difference

Temporal differential vector quantization is obtained by taking temporal difference of images over the temporal domain. A simple temporal predictor would constitute the difference between the pixel location in the current frame and the previous frame, i. e $\hat{I}_t = I_{t-1}(i, y)$.

The second predictor used is similar to the spatial difference predictor except that the difference is taken over the previous frame, i. e

$$\hat{I}_t(i, j) = \frac{I_{t-1}(i-1, j) + I_{t-1}(i-1, j) + I_{t-1}(i-1, j-1)}{3},$$

this is the average of the nearest neighbor pixels from the previous frame. The predictor error is then computed as,

$$e_t(i, j) = I_t(i, j) - \hat{I}_t(i, j) .$$

Spatio temporal differential vector quantization is implemented by using spatial differential vector quantization on the first frame of the video sequence, the first frame is reconstructed at the decoder, and consequently temporal difference is taken over subsequent frames which are quantized and later reconstructed at the decoder. In a given sequence from video, there is a lot of temporal redundancy between consecutive frames, this scheme exploits the temporal redundancy between the frames to achieve better compression and quality of recreated image at the decoder. The process is best described in the figure below,

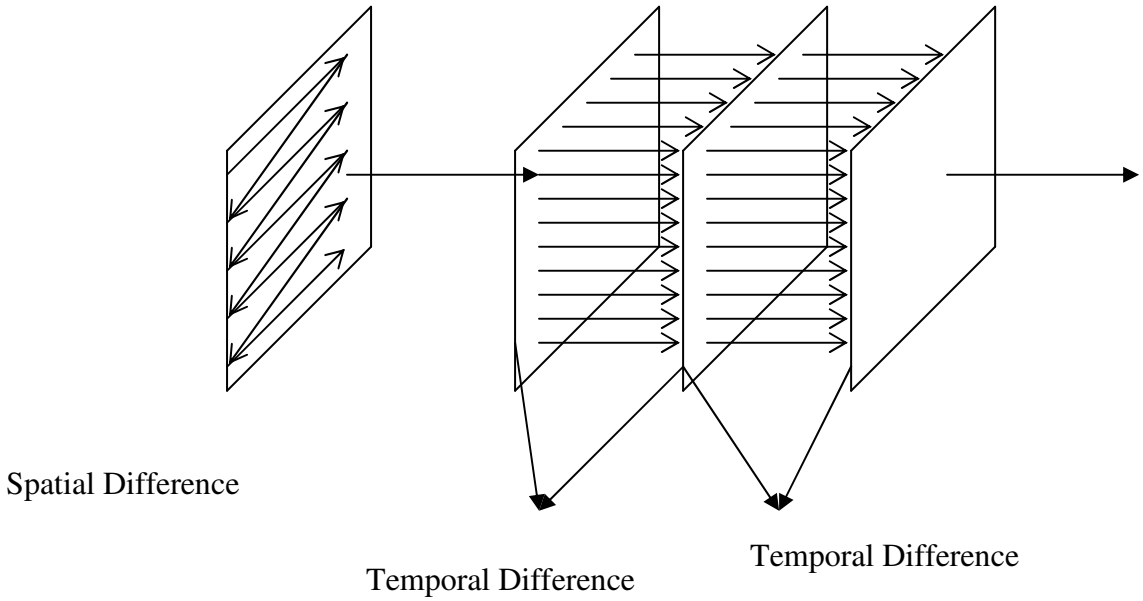


Figure 10: Spatio temporal difference

Spatial and temporal differential vector quantization of video is achieved by encoding the images at the encoder and decoding the compressed stream at the decoder. The encoder and decoder used in this paper are described below.

Overall, the temporal DVQ schemes will perform best if there is an initial segmentation of the video scenes, such that each scene can be handled separately as a temporal block. This will avoid taking differences across a scene change. We also note that scene change detection can be performed in real-time as the video is being encoded.

2.5.3 Encoder

Differential vector quantization is simply the extension of vector quantization on differences. In the present work, horizontal, spatial difference have been identified and quantized. The methodology is straightforward, depending on the type of DVQ scheme, (i.e. horizontal or temporal) difference between two images is taken, and this involves the pixel by pixel difference of the two images. The difference data is vectorized. The vectors from the difference data are then searched over the dictionary for matches. If a match is found, the index of the matching vector in the dictionary is used to represent the input difference vector, thereby quantizing the input vector. This process is repeated for all the input difference vectors.

These indices are then passed into a Huffman coder to generate the Huffman codes for the indices. The compressed bit stream is then transmitted over the network. Temporal DVQ and spatial DVQ may require different quantizers (or codebooks) for best compression.

2.5.4 Decoder

The decoder consists of two stages, the Huffman decoder, and codebook for retrieving the original vectors. Once the decoder receives the compressed bit stream, the Huffman decoder is used to produce the original indices. These index positions are then passed on

to the dictionary to identify the original vectors. These vectors are then arranged into a two dimensional image.

Depending on the scanning (horizontal or vertical) of differences, the original images are then reconstructed at the decoding stage.

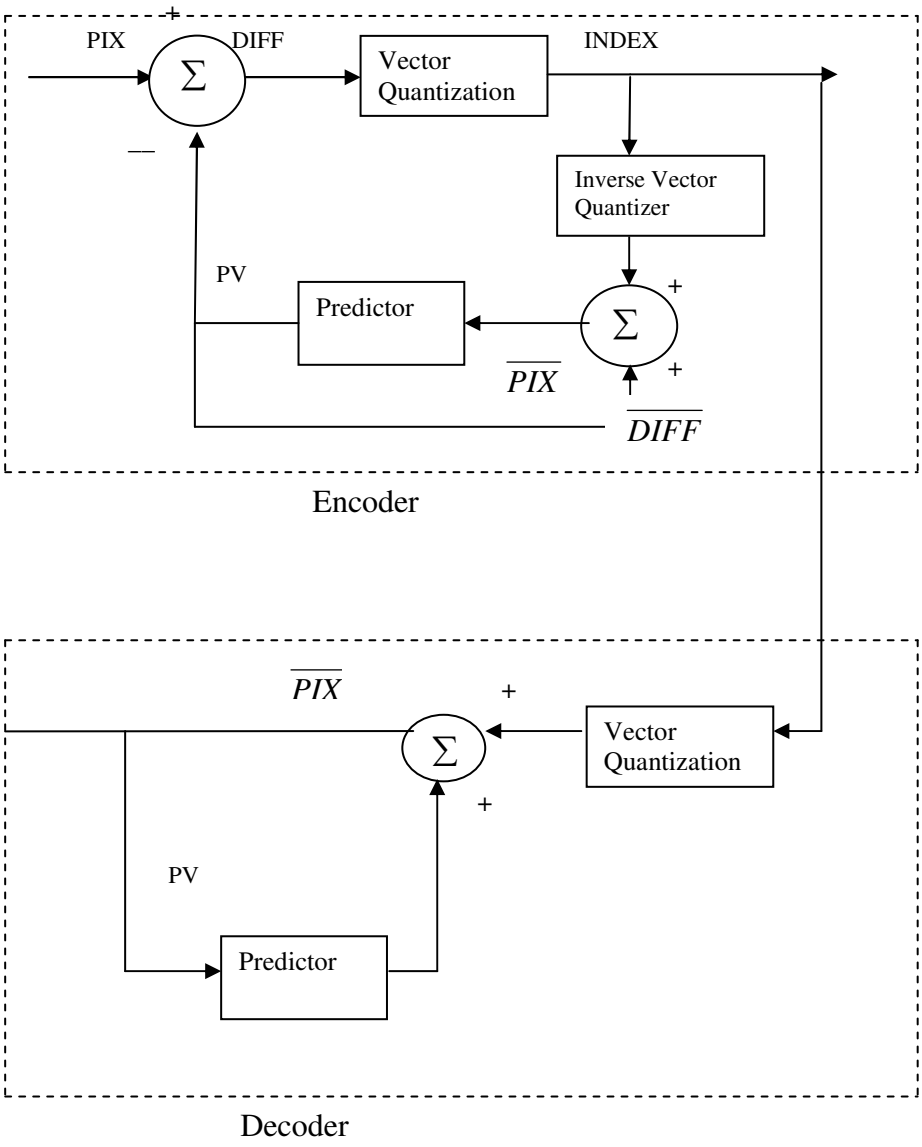


Figure 11: Block diagram of DVQ

The above block diagram shows the encoder and decoder used in differential vector quantization. The input vectors are searched over the codebook for symbol matches to

produce an index, the searching method is the key factor determining the speed of the differential vector quantization algorithm. Sequential search over the codebook will consume lot of time and make the algorithm slower. This is a bottleneck in vector quantization in terms of searching speed.

In order for the vector quantization algorithm to have real time performance, there is a need for an efficient scheme for searching for the vector over the codebook.

We propose a new approach for searching the vectors over the codebook using the suffix tree data structure. The codebook is represented using the suffix tree data structure, the searching speed of suffix tree data structure is exploited for searching the indexes in the codebook thus rendering real time results.

Using suffix tree data structure we introduce the concept of k- error match over the suffix tree. Using this approach an amount of $\pm k$ error is introduced over the suffix tree for increasing the probability of finding indexes over the suffix tree. The approach is described in detail in the following chapter.

3. Suffix Tree

3.1 Introduction

The first linear time algorithm for constructing suffix trees was given by Weiner [24] in 1973, although he called his tree a position tree. Ukkonen [15] developed a conceptually different linear- time algorithm for building suffix trees that has all the advantages of McCreight's [23] algorithm but allows a much simpler implementation.

3.1.1 Basic Definitions

A **suffix tree** $\mathcal{T}$ for an m-character string S is a rooted directed tree with exactly m leaves numbered 1 to m. Each internal node, other than the root, has at least two children and each edge is labeled with a nonempty substring of S. No two edges out of a node can have edge labels beginning with the same character. The key feature of the suffix tree is that for any leaf i, the concatenation of the edge labels in the path from the root to leaf i exactly spell out the suffix of S that starts at position i. That is it spells out $S[i...m]$.

The label of a path from the root that ends at a node is the concatenation, in order, of the substrings, labeling the edges of that path. The label of a node is the label of the path root of $\mathcal{T}$ to the node.

For any node v in a suffix tree, the **string depth** of v is the number of characters in v's label.

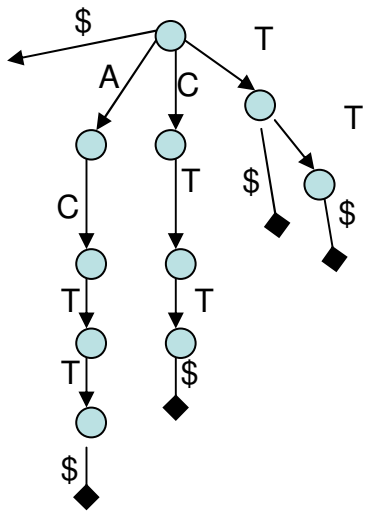
A path hat ends in the middle of an edge (*u, v*) splits the label on (*u, v*) at a designated point. Define the label if such a path as the label of u concatenated with the characters on the edge (*u, v*) down to the designated split point *i.*

3.1.2 Implicit Suffix Trees

Ukkonen's algorithm constructs a sequence of implicit suffix trees, the last of which is converted to a true suffix tree of the string S.

An implicit suffix tree for a string *S* is a tree obtained from the suffix tree for *S*\$ by removing every copy of the terminal symbol \$ from the edge labels of the tree, then removing any edge that has no label , and then removing any node that does not have at least two children. An implicit suffix tree for a prefix *S* [1…*i*] of *S* is similarly defined by taking the suffix tree for *S* [1…*i*] \$ and deleting \$ symbols, edges, and nodes as above.
We use $\mathcal{T}_i$ to denote implicit suffix tree of the string *S* [1…*i*] ,*i*=1,2,3,….*m.*

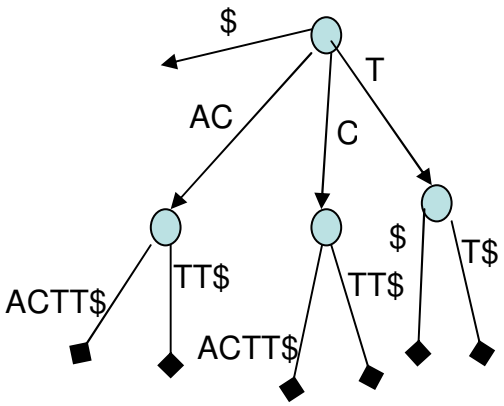
A *trie* is an indexing structure used for indexing sets of key values of varying sizes [17]. A trie is a tree in which the branching at any level is determined by a partial key value (see Figure 12). A suffix tree is a PATRICIA trie [18] built over the set of all suffixes of a given sequence S. Figure 13 illustrates a sample suffix tree of a short sequence. Each path from the root of the suffix tree represents a suffix of the original string. Any individual suffix of the original sequence can be recreated by walking along a path from the root and catenation of the labels of the edges traversed along the way. Nodes with a single outgoing edge can be collapsed, resulting in edges with multi-symbol labels, such as the edge labeled AC coming out of the root of the tree in Figure 13.



List of Suffixes

ACTT$
CTT$
TT$
T$
$

Figure 12: A suffix trie for sequence ACTT

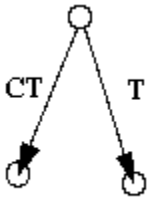


List of Suffixes

ACACTT$
CACTT$
ACTT$
CTT$
TT$
T$
$

Figure 13: Suffix tree for sequence ACACTT. Some edges have multi-symbol labels resulting from collapsing nodes with single children

.



List of
Suffixes
CT$
T$
$

Figure 14: Suffix tree for sequence CT

A suffix tree for sequence S of length *n* can be constructed in O(n) time [19] and the number of nodes in the tree is in general a linear function of *n* [20].

A generalized suffix tree (GST) is an augmented version of the suffix tree allowing for multiple sequences to be stored in the same tree. A GST can be viewed as a suffix tree with additional sequence-identifier leaves added to the leaves of the original suffix tree. For every suffix, its sequence of origin is identified. A GST can be augmented with information about the number of different sequences that contain suffixes expressed by descendants of each node (Figure 15). A GST can be constructed in O (*n*) time, where *n* is the sum of lengths of all sequences stored in the tree.

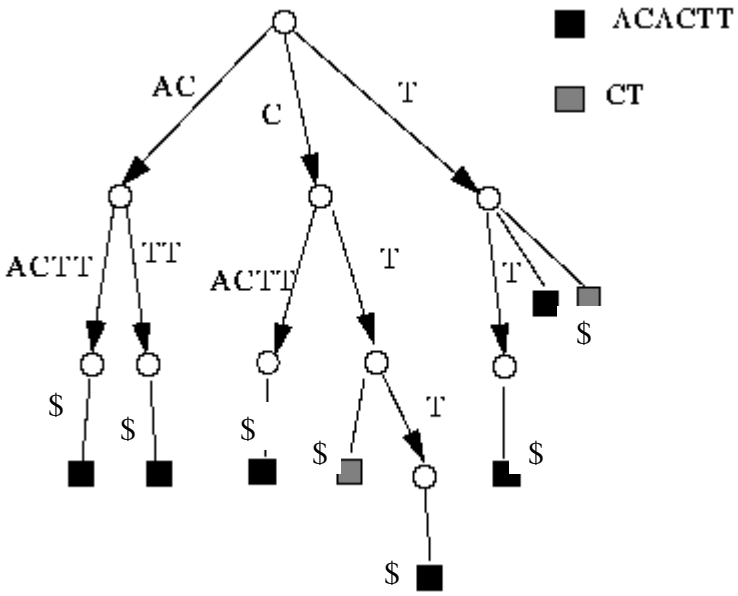
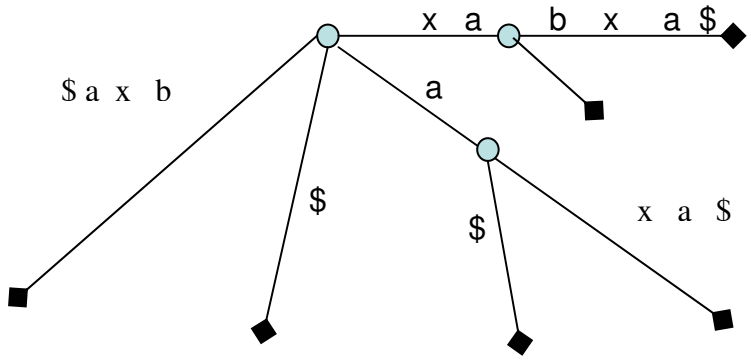


Figure 15: A generalized suffix tree for two sequences, $S_1 = ACACTT$ and $S_2 = CT$. Notice the sequence identifier leaves (squares) denoting the origin of every suffix of every sequence. The number of identifier leaves for every sequence is equal to the number of suffixes of that sequence

3.2 Suffix tree construction

Suffix tree construction algorithms have been described in the literature [19, 21, and 22]. They operate by constructing an initial tree with a single branch corresponding to the entire sequence and incrementally modifying the tree to include all of its suffixes. An important variable in the construction process is the choice of data structures used to represent the tree. Fixed-node size trees have a node access time advantage over child list-based ones since a descendant can be accessed directly, without having to traverse a list. They are particularly useful when the number of descendant pointers is small.

3.2.1 Creating the Suffix Tree



List of Suffixes

Xabxa$
abxa$
bxa$
xa$
a$
$

Figure 16 : Suffix Tree for string xabxa$

Ukkonen's algorithm constructs an implicit suffix tree I_i for each prefix S [1..i] of S, tiling from I_1, and incrementing i by one until I_m is built. The true suffix tree for S is constructed from I_m and the time for *the* entire algorithm is O (m).

3.2.2 Ukkonen's algorithm

For an input sequence S of length m Ukkonen's algorithm is divided into *m phases.* In phase i + 1, tree I_{i+1} is constructed from I_i. Each phase i +1 is further divided into i +1 *extensions*, one for each of the i +1 suffixes of S [1...i+1]. In *extension j* of phase i + 1, the algorithm first finds the end of the path from the root labeled with substring S [j...i]. It then extends the substring by adding the character S [i + 1] to its end, unless S (i + I) already appears there. So in phase i + 1, string S[1..i + 1] is first put in the tree, followed by strings S[2..i +1], S[3..i+1],… (In extensions 1, 2, 3, , respectively) Extension i+ 1 of phase i + 1 extends the empty suffix of S[1..i], that is, it puts the single character string $S(i+1)$ into the tree(unless it is already there). Tree I_i is just the single edge labeled by character S[1].

A high level description of the Ukkonen algorithm is given below,

Construct tree I_1

For i from 1 to $m-1$ do

begin {phase i + 1}

    For j from 1 to i + 1

    begin {extension j}

        Find the end of the path from the root labeled S[j..i] in the

        current tree, If needed, extend that path by adding character S[i + 1),

        thus assuring that string S[j. .i + 1] is in the tree.

    end;

end;

The suffix tree is constructed in an efficient manner by implementing the following two algorithms, single extension algorithm and single phase algorithm.

The final implicit tree $\mathcal{T}$ can be converted to a true suffix tree in O (m) time. First, add a string terminal symbol $ to the end of S and let Ukkonen's algorithm continue with this character. The effect is that no suffix is now a prefix of any other suffix, so the execution of Ukkonen's algorithm results in an implicit suffix tree in which suffix ends at a leaf and so is explicitly represented. The only other change needed is to replace each index e on every leaf edge with the number m. This is achieved by and O (m) – time traversal of the tree visiting each leaf edge. When these modifications have been made, the resulting tree is a true suffix tree.

After the suffix tree is created, the suffix tree is used to search for a substring within the tree structure. In this work, a suffix tree is created for pixel differences which range from -255 to +255. Thus rather than character symbols, the path labels will be made up of numbers.

4. Real Time Video Compression

4.1 Overview

In order to achieve real time video using DVQ, the searching speed of the DVQ had to be made real time. As seen from the previous chapter, suffix tree search proved to be an efficient way to search for a codeword on a codebook, by utilizing the unique structure of the suffix tree ,a suffix tree dictionary was created for the vector quantized dictionary. The searching speed on a suffix tree helps to render video compression and decompression in real time. We segmented a video sequence into frames of MPEG 1 size (240 x 352), which were later vectorized. The vectors were searched for index positions on the suffix tree dictionary, and the indexes were encoded using Huffman coding and transmitted to the decoder which retrieved the code words using the indexes.

4.2 Codebook Generation

Codebooks were created specifically for a particular kind of video, for example surveillance video, the video was parsed into video sequences which were identified from the video based on scene changes. The video sequence was broken down into 30 images per second. Individual images were selected visually from different video sequences and were fed into the vector quantizer one after the other for all the scene changes in the video. The vector quantizer produced a vector quantized dictionary for that particular video. Difference of pixels of the vector quantized dictionary is taken to create the DVQ dictionary, the DVQ dictionary was created using two different horizontal snake scans of the vectorized dictionary and was also created using two different vertical snake scans.
The DVQ dictionary was transformed into a suffix tree data structure to create a suffix tree DVQ dictionary, using the suffix tree data structure as the codebook for the DVQ, vectorized difference values of images were searched on the suffix tree for index matches. The index matches are used with prediction at the decoder to recreate the image.

4.3 Binary Search

Binary search was employed for searching the codebook in DVQ for comparison with the suffix tree search. Binary search was used to search for indexes from a DVQ dictionary. A vector quantized dictionary was created using the method described in the previous chapter, the dictionary was searched for index matches using binary search method, using the binary search there can be no more than $\log_2 n$ comparisons, the worst case runtime complexity of the binary search is $O(\log_2 n)$.

The video sequence is broken down to individual frames , each frame is vectorized and these vectors are searched on the dictionary for possible matches using binary search,
In order to perform the binary search on the dictionary, the Vector quantizated dictionary is first sorted before the implementation of the binary search algorithm.

Binary search is a fast algorithm for searching in a sorted array S of n vectors, to search for vector q, we compare the vector q element by element to the middle vector $S[n/2]$. If q appears before $S[n/2]$ (i.e the value of q is less than the middle vector of the dictionary), it must reside in the top half of our dictionary, if not, it must reside in the bottom half of our dictionary. By recursively repeating this process on the correct half, we find the vector in a total of $\log n$ comparisons, which is comparatively much better than the $n/2$ we expect with sequential search. If a match is found the index of the match is encoded using huffman coding and is transmitted to the decoder which decodes the index and retrieves the vector from the index, in certain cases if a match is not found $\pm k$ error is added to the input vector and the vector is searched again for possible matches.

Binary search on the Vector quantized dictionary could not render real time results. The comparison results suggested that suffix tree search outperforms binary search and it gets

more efficient as the size of the dictionary is increased linearly. The results of the comparison are provided in the following section..

4.4 Suffix Tree Search

The suffix tree is created using vector quantized difference of pixel values. The tree serves as the dictionary for the DVQ. Difference vectors are searched on the suffix tree for possible matches. The index (position) of the match is recorded, encoded to form the compressed stream. At the decoder we perform reconstruction of the image. This is done by first decoding the encoded vector indices, and then by replacing each index with its corresponding difference vector. The block diagram of the method is presented below,

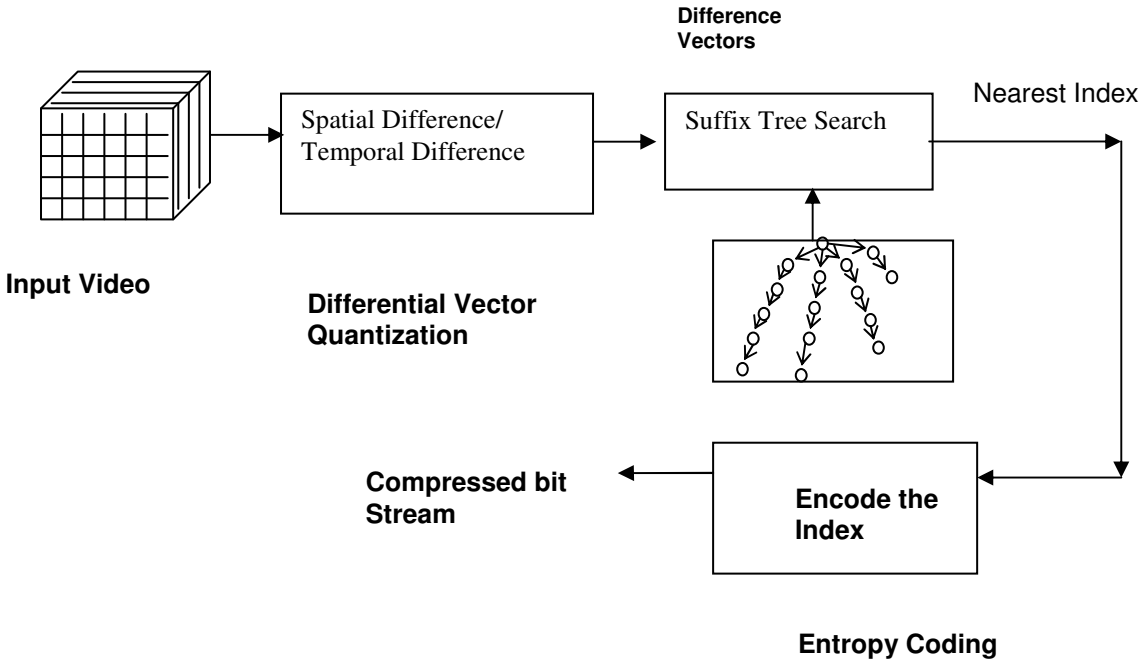


Figure 17 : Encoder for DVQ using suffix trees

Input video is segmented into frames which are then passed into the difference block for difference of images. The difference pixel blocks (2x2) are searched on the suffix tree dictionary for possible matches. The index of the matching block is then passed on to the encoder for entropy coding.

Searching for indices is performed at the encoder by the following schemes,
I. *k*-Error Match
II. Best Match
III. First Match

4.4.1 k- Error Match

Searching the difference pixel block on the suffix tree dictionary will not always yield an index. One of the methods to ensure that an index is generated is to consider a k-error dictionary where an error of *k* is added to the dictionary. The difference pixel blocks are searched with an error of $\pm k$. An error of $\pm k$ is added to the suffix tree dictionary. The vector is first searched over the suffix tree dictionary, if a match is not found then an error of $k = \pm 1$ is added to the vector which is similar to adding an error of ± 1 to the dictionary. The input vector with the error value added is searched over the dictionary for possible matches, the vector is searched over the dictionary element by element of the vector for possible matches with an error of ± 1, this process is repeated till $k = \pm 3$ if a match is not found, if a match is not found even with a $k = \pm 3$, then an escape signal is sent instead of the index, later at the decoder using predictive schemes the escape signal is substituted with pixel values.

It has been found that the viewer cannot discern the difference in an image for $\pm k$ of 3. By adding $\pm k$ error, the probability of finding a match in the codebook increases, as the codebook is built in vectorized differences, by adding $\pm k$ error we are increasing the size of the dictionary by $2k+1$, this is significant increase in the number of vectors that can be matched on the codebook without physically increasing the size of the codebook.

By varying the value of $\pm k$ error we observe there is significant compression achieved at the decoder. This value of $k = 3$ has been used throughout the matching process.

Following is a diagram of $\pm k$ suffix tree dictionary,

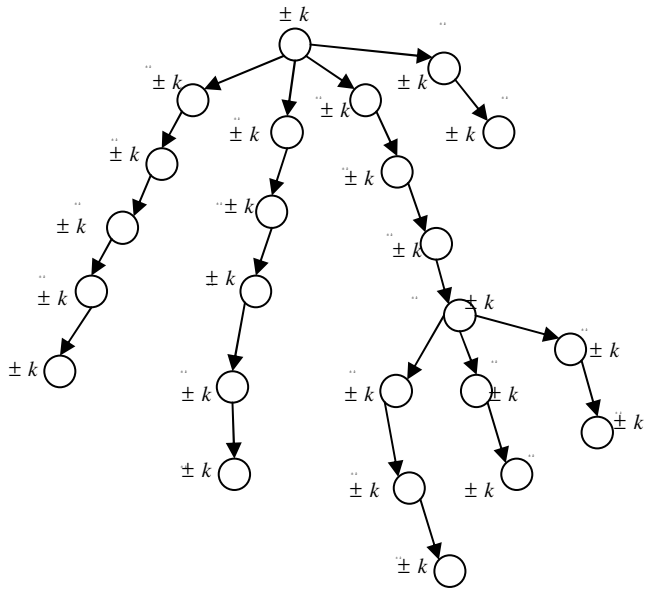


Figure 18: *k* - error matching on a suffix tree Dictionary

4.4.2 Best Match

The pixel block search on the suffix tree dictionary with $\pm k$ error will yield many possible matches to the block being searched on the dictionary. The best match approach is to include the index of the match which has the least $\pm k$ error associated with it. The best match will have the better quality of the recreated image. This matching process may results in slowing down the matching process as extra time is required for finding out the best match.

4.4.3 First Match

The first match approach while searching on an $\pm k$ error dictionary is to choose the match that is found first. This will minimize the search time and make the search faster. This however , may not provide the best quality.

The decoder uses the encoded stream to reconstruct the image. The block diagram for the decoder is presented below,

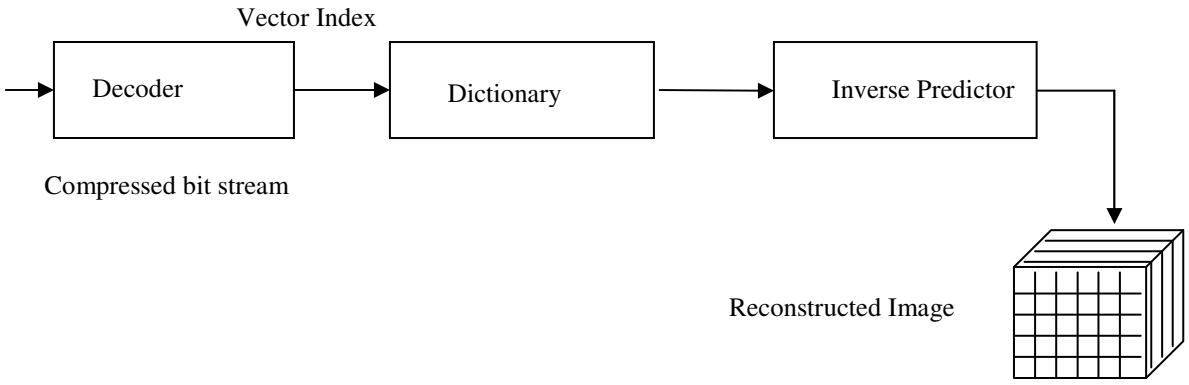


Figure 19 : Decoder for DVQ using suffix trees

The encoded stream is decoded, and the blocks using the indexes from the dictionary along with prediction of pixel blocks are used to recreate the image at the decoder.

5. Results

5.1 Experimental Setup

The purpose of the experiments was to validate our approach that by using suffix trees as the codebook/ dictionary we could achieve real time video using DVQ, in doing so we were also able to validate that suffix tree search was far more superior to binary search. During our experimental analysis we had used five different videos. The videos used in these experiments are in MPEG1 video format. The videos used are, a. Reagan, b. Surveillance, c. Conference, d. Bond , and e. Tennis videos
The videos were provided to us courtesy of Array Microsystems Inc.

The experiments in this thesis were performed on Intel Pentium IV 2.6 GHz Microprocessor with 756 RAM on a windows XP operating system

5.2 Occurrence statistics

The following plots show the frequencies of vectors in the Reagan video. The table shows the vectors and their frequencies. The following is frequencies of vectors from a difference image of two different scenes in the Reagan video, due to the difference in the scenes notice the difference between two images produces vectors with large values.

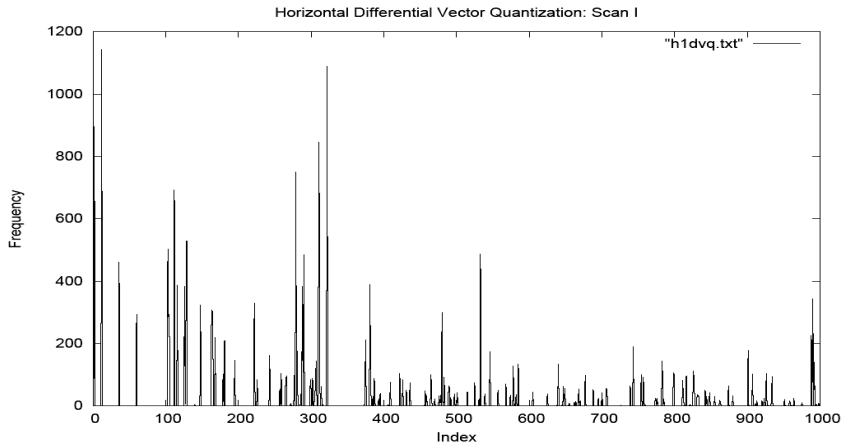
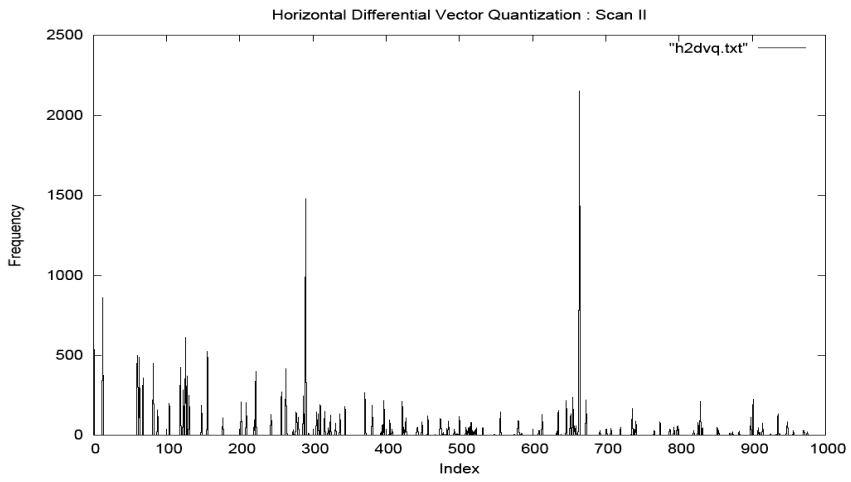


Figure 20 : The frequency of indexes from horizontal scan I (Reagan Video)

Vector	Frequency
-2 -2 -2 -2	243
-1 -1 -1 -1	390
1 1 1 1	1089
11 13 12 13	343
21 14 14 16	154
1 0 0 -1	1167
0 0 0 0	456
5 4 4 3	535
14 15 17 18	343
16 16 12 16	203

The following is a plot of difference of two similar frames in Reagan video using horizontal scan II, notice the high frequency of similar vectors.



Vector	Frequency
-2 -2 -2 -2	243
-1 -1 -1 -1	390
1 1 1 1	489
0 1 0 0	343
1 0 1 0	546
1 0 0 -1	896
0 0 0 0	563
1 -1 0 0	2312
3 -2 1 2	431
0 2 4 1	203

Figure 21 : The frequency of indexes after Horizontal Scan II (Reagan Video)

The following plot is for vertical scan I on surveillance video, the difference was between two images in the video where there was a scene change.

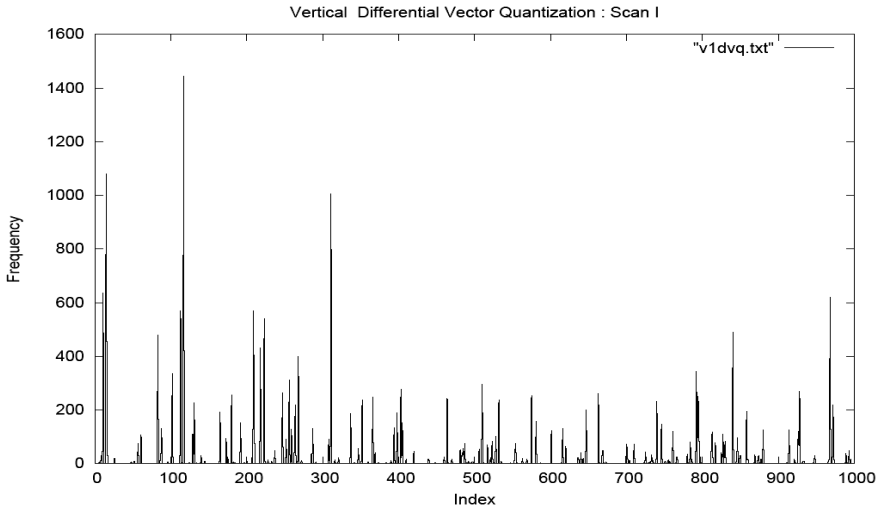


Figure 22: Frequency of Indexes for Vertical Scan I (Surveillance Video)

The following plot is for frequencies of vectors over two similar images in bond video using vertical scan II.

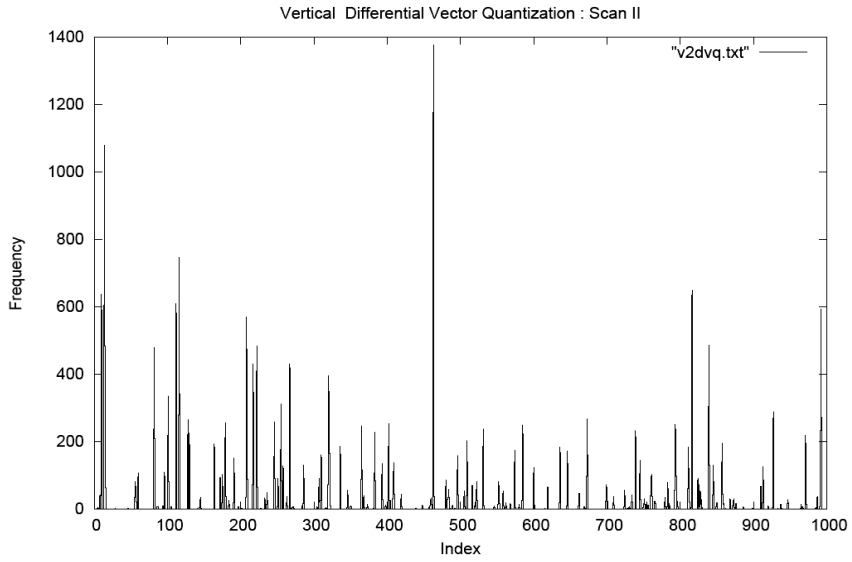


Figure 23: Frequency of Indexes for Vertical Scan II (Bond Video)

The different horizontal and vertical scans yielded similar vectors with high frequencies, this helped in finding the index of the vector in the dictionary and thus compression of the image.

4.1 Real – time performance

The following is a logarithmic graph illustrates the time taken for sequential search and suffix tree search on increasing dictionary sizes, the graph is based on time taken to search for 30 frames.

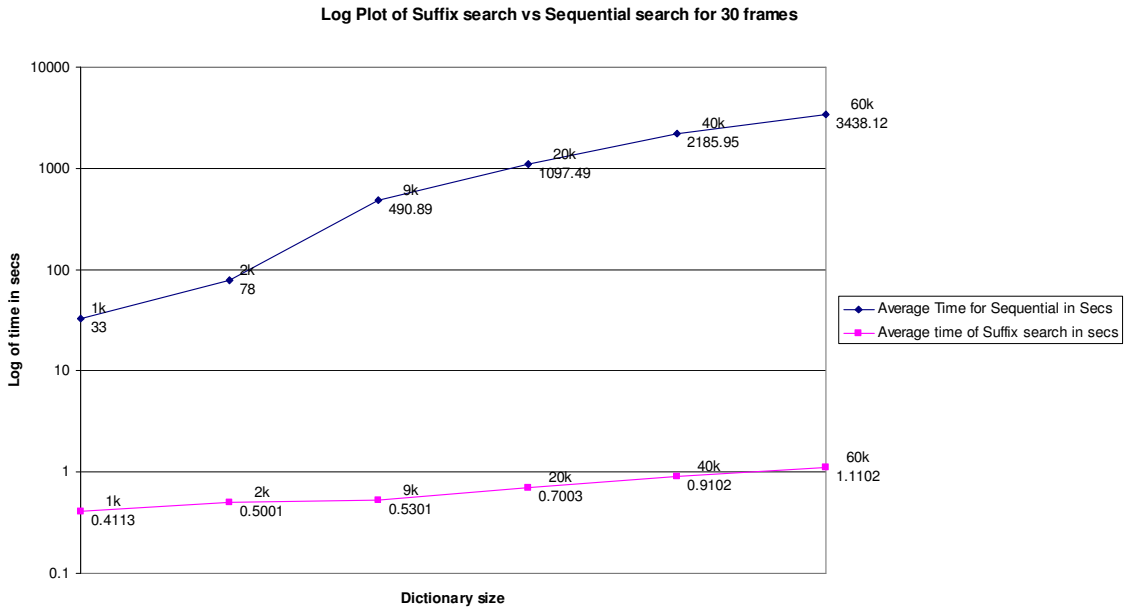
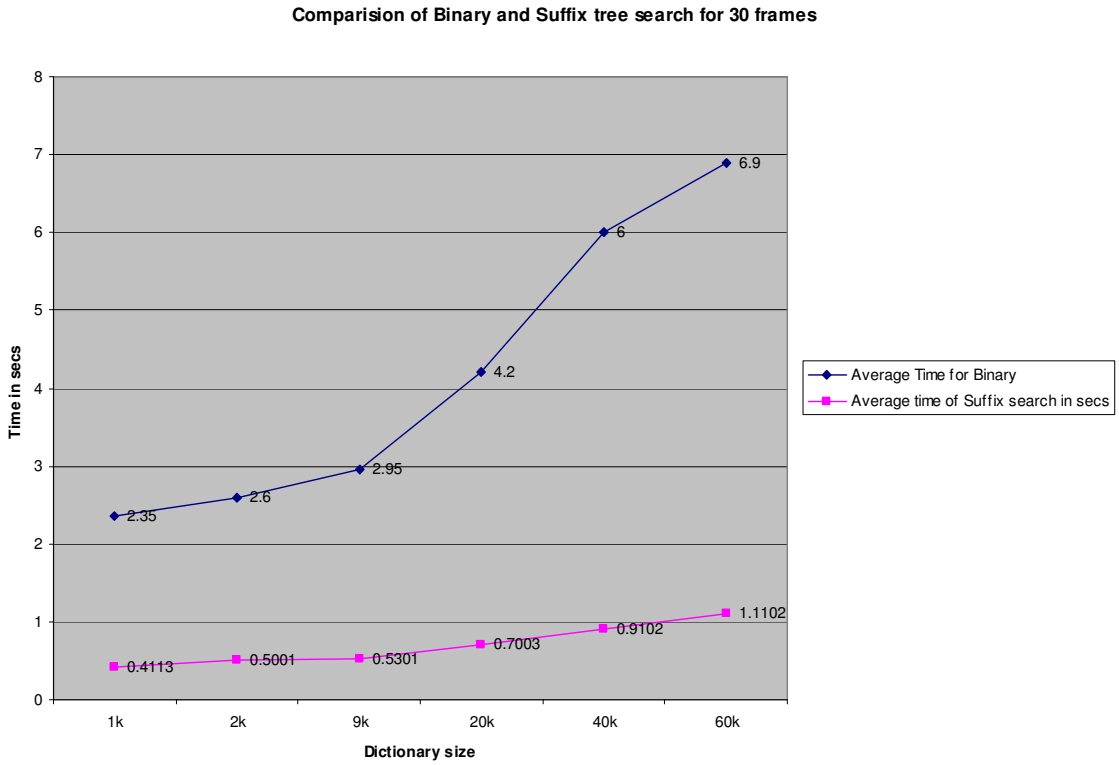


Figure 24: Log plot for average search time in seconds for suffix tree and sequential search

The following shows the times taken for suffix tree search and binary search on increasing dictionary sizes. Notice that suffix tree performs is superior to binary search with increasing dictionary size.

Figure 25: Comparison of suffix tree and binary search time for 30 frames



Using a video sequence of 30 frames suffix tree search was performed over dictionaries of varying sizes, the dictionary size is indicated in terms of the number of vectors in the dictionary. The search time increased linearly with the increase in dictionary size. The recreated image is of higher quality when the suffix tree search is performed on a larger size dictionary. By using dictionary size between 1K and 10K vectors it is evident that suffix tree search is capable of encoding and decoding the images in real time, larger dictionary sizes would increase the quality of the reconstructed image but would make the search slower and cannot render video in real time.

4.4 Compression Results

4.4.1 Spatial Differential Vector Quantization

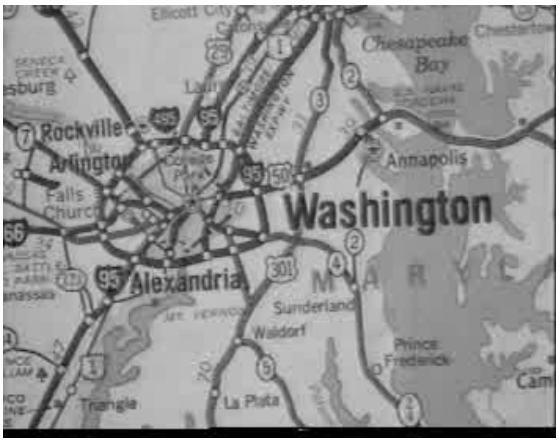
Using the indexes from suffix tree DVQ dictionary along with inverse prediction, the input images were recreated at the decoder. The resulting reconstructed images are shown below. The following images are from Reagan video.

(a)

(b)

(c)

(d)



(e)

(f)

(g)

(h)

Figure 26 : Reconstructed images after real time compression using DVQ and suffix trees.

(a) Original image, (b) Results at $|D| =$ 16K, (c) Results at $|D| = 10$K, (d) Results at $|D| = 4$K, (e) Original image, (f) Results at $|D| = 10$K, (g) Results at $|D| = 10$K, (h) Results at $|D| =$ 16K

The following table lists the compression ratio, mean square error (MSE), peak signal to noise ratio. MSE and PSNR were calculated using the following,

$$MSE = \frac{1}{MN}\sum_{y=1}^{M}\sum_{x=1}^{N}[I(x,y) - I'(x,y)]^2 \ , \ PSNR = 20 * \log 10\left(\frac{255}{\sqrt{MSE}}\right) .$$

Spatial differential vector quantization									
Video	$\lvert D\rvert = 4\text{K}$			$\lvert D\rvert = 10\text{K}$			$\lvert D\rvert = 16\text{K}$		
	CR	**MSE**	**PSNR**	**CR**	**MSE**	**PSNR**	**CR**	**MSE**	**PSNR**
Reagan	5.26	72.22	29.5	5.06	51.896	30.97	4.86	22	34.7
Surveillance	4.42	51.2	31.05	4.16	42.3	31.8	4.05	24.2	34.32
Video Conference	8.14	53.2	30.8	8.07	45.1	31.59	7.91	34.3	32.86
Bond Video	4.02	32.2	33	4.91	30.1	33.5	4.81	21	34.908
Tennis	6.07	64.8	30.06	5.91	44.8	31.7	5.43	32.6	33.07

Table 1: CR, MSE, and PSNR for spatial DVQ on five different videos

4.4.2 Temporal Differential Vector Quantization

Temporal redundancy was exploited in temporal differential vector quantization to obtain better compression ratio. The following table lists the compression, MSE, PSNR for five different videos.

Temporal differential vector quantization									
Video	$\|D\| = 4$K			$\|D\| = 10$K			$\|D\| = 16$K		
	CR	**MSE**	**PSNR**	**CR**	**MSE**	**PSNR**	**CR**	**MSE**	**PSNR**
Reagan	18.32	128	27.05	14.32	101	28.8	14.54	15	36.36
Surveillance	12.51	111	27.6	11.5	95	28.5	10.2	22	34.7
Video Conference	22.61	88	28.6	18.27	64	30.1	17.82	25	34.1
Bond Video	16.41	71	29.61	15	53	30.88	11.24	28	33.1
Tennis	18.21	210	24.9	16.24	110	27.1	12.11	22	34.7

Table 2: CR, MSE, PSNR, for temporal DVQ on five different videos

The following are recreation of surveillance, videoconference, tennis, bond videos that was recreated at the decoder using the approach mentioned in chapter 2.5.2, spatial dictionary of size $|D| = 16$K, and a temporal dictionary of size $|D| = 10$K were used in recreating images at the decoder.

Surveillance video

(a)

(b)

(c) (d)

Figure 27: Reconstructed images in real time using temporal DVQ and suffix trees, (a) original frame 1, (b) original frame 2, (c) reconstructed frame 1, (d) reconstructed frame 2

Conference Video

(a) (b)

(c) (d)

Figure 28: Reconstructed images in real time using temporal DVQ and suffix trees, (a) original frame 1, (b) original frame 2, (c) reconstructed frame 1, (d) reconstructed frame 2

Tennis video

(a) (b)

(c)

(d)

Figure 29: Reconstructed images in real time using temporal DVQ and suffix trees, (a) original frame 1, (b) original frame 2, (c) reconstructed frame 1, (d) reconstructed frame 2

Bond Video

(a)

(b)

(c)

(d)

Figure 30: Reconstructed images in real time using temporal DVQ and suffix trees, (a) original frame 1, (b) original frame 2, (c)original frame 1, (d) original frame 2

Figure 31: Blocky temporal image

Blockiness introduced into the image when using a codebook of $|D| = 1\text{K}$ vectors.

4.5 Effect of the error parameter *k*

A certain amount of error *k* is added to the input vector to increase the number of matches to be found in the dictionary. The following tables show the compression ratio, MSE, PSNR achieved with various *k* values for both spatial and temporal differential vector quantization.

Spatial DVQ

***k*- error at** $\lvert D \rvert = 16\text{K}$	**Reagan video**			**Surveillance video**		
	CR	MSE	PSNR	CR	MSE	PSNR
$k=\pm 1$	5.12	17	35.82	4.05	16.4	36.09
$k=\pm 2$	5.22	22	35.82	4.11	17.2	35.82
$k=\pm 3$	5.26	22	34.7	4.42	24.2	34.32

Table 3 : Effect of *k* error on CR, MSE, and PSNR in spatial DVQ

Temporal DVQ

***k*- error at** $\lvert D \rvert = 16\text{K}$	**Reagan video**			**Surveillance video**		
	CR	MSE	PSNR	CR	MSE	PSNR
$k=\pm 1$	8.53	36.27	32.56	6.01	33.4	32.91
$k=\pm 2$	15.74	25.3	34.13	8.71	28.90	33.65
$k=\pm 3$	18.32	15	36.36	12.51	22	34.7

Table 4 : Effect of *k* error on CR, MSE, and PSNR in temporal DVQ

The following plot shows the peak signal to noise ratio for spatial and temporal DVQ for the Reagan video over a sequence of 30 frames.

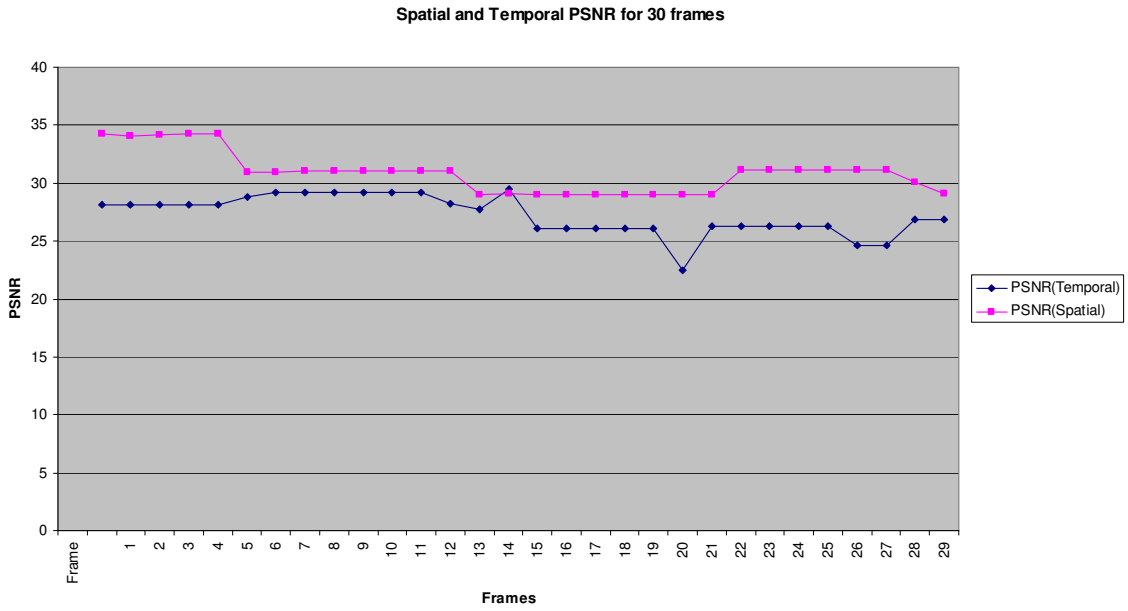


Figure 32: Spatial and temporal PSNR for 30 frames

5. Conclusion and Future Work

Traditionally Differential vector quantization was implemented on hardware to render real time results [2]. The pivotal factor to render real time video using DVQ is the searching speed for the index in the dictionary. Most of the previous software based methods were not able to achieve real time results. Using the suffix tree to represent the dictionary in Differential vector quantization we were able to produce real time results. Introducing the concept of temporal difference provided greater compression in the algorithm by exploiting the temporal redundancy in video.

Compression of image data is achieved by using the difference of images, Suffix tree based DVQ is able to achieve real time compression by integrating the search speed of the suffix tree data structure and the compression achieved by implementing differential vector quantization. By introducing the concept of $\pm k$ error on the suffix tree we were able to greater compression.

The present work can be extended for real time object tracking. Object tracking involves the recognition of a particular object within successive image frames, the indexes of the object can be recorded from the dictionary, and it is possible to track the object within successive image frames by keeping a track of the vector indexes within successive image frames.

List of Acronyms

MPEG: Moving Picture Experts Group

GOP: Group of Pictures

MBVC: Model Based Video Coding

DCT: Discrete Cosine Transform

JPEG: Joint Photographic Experts Group

CR: Compression ratio

DVQ: Differential vector quantization

VQ: Vector quantization

MSE: Mean square error

PSNR: Peak signal to noise ratio

Reference:

[1] Buzo, A., Gray, A.H., Jr., and Gray, R.M., and Markel, J.D., “Speech coding based upon vector quantization,” *IEEE Transactions on Acoustics Speech and Signal Processing* ASSP -28 pp .562-574

[2]. E. Fowler, K. C. Adkins, S. B. Bibyk, and S. C. Ahalt, "Real-Time Video Compression Using Differential Vector Quantization," IEEE Transactions on Circuits and Systems for Video Technology, vol. 5, pp. 14-24, February 1995

[3] J. E. Fowler, M. R. Carbonara, and S. C. Ahalt, "Image Coding Using Differential Vector Quantization," IEEE Transactions on Circuits and Systems for Video Technology, vol. 3, pp. 350-367, October 1993

[4] D. A. Huffman, ``A Method for the Construction of Minimum Redundancy Codes," *Proceedings of the IRE*, Vol. 40, pp. 1098--1101, 1952.

[5] J. Ziv and A. Lempel, “A Universal Algorithm for Sequential Data Compression, " *IEEE Transactions on Information Theory*, Vol. 23, pp. 337--342, 1977

[6] J. Ziv and A. Lempel, ``Compression of Individual Sequences via Variable-Rate Coding," *IEEE Transactions on Information Theory*, Vol. 24, pp. 530--536, 1978

[7] C. E. Shannon, ``A mathematical theory of communication," *Bell System Technical Journal,* vol. 27, pp. 379-423 and 623-656, July and October, 1948.

[8] K. R. Rao, J. J. Hwang, Techniques and standards for image, video and audio coding, 1996 Prentice Hall

[9] Roger J. Clarke, Digital Compression of Still image and video

[10] John Villasenor, Ya-Qin Zhang, Jiangtao Wen, Robust video coding algorithms and systems. Special Isuue of the Proceedings of the IEEE on Wireless Video

[11] Joan Mitchell, William B. Pennebaker, Chad, E. Fogg and Didier J. Legall, MPEG video compression standard, International thompson publishing

[12] Didier LeGall, MPEG: a video compression standard for multimedia applications. Communications of the ACM, April 1991, Vol 34, No 4, 13 pages

[13] John Watkinson, MPEG-2, 1999 Focal Press

[14] E.M. McCreight A space-economical suffix tree construction algorithm *Journal of the ACM*, 23:262-272, 1976.

[15] E. Ukkonen. On-line construction of suffix trees *Algorithmica*, 14(3):249-260, September 1995.

[16] Paul Bieganski, John Riedl, John Carlis, Ernest F. Retzel. Generalized Suffix Trees for Biological Sequence data: Applications and Implementation *University of Minnesota*, 1994

[17] Horowitz, Ellis and Sahni, Sartaj, *Fundamentals of Data Structures*, Computer Science Press, 1987.

[18] Bays, J. C., The Complete PATRICIA, Ph.D. dissertation, University of Oklahoma, 1974.

[19] McCreight, E. M., *A Space Economical Suffix Tree Construction Algorithm*, JACM, 23, 262-272, 1976.

[20] Szpankowski, Wojciech. *Probabilistic Analysis of Generalized Suffix Trees*, Combinatorial Pattern Matching, Third Annual Symposium, Proceedings, Springer Verlag, 1992, 1-14

[21] Needleman, Saul B. and Wunsch, Christian D., *A General method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins*, Journal of Molecular Biology, 48, pp. 443-453, 1970.

[22] Powell, Patrick A., *Using and Constructing the Suffix Tree Index Structure*, U. of Minnesota Computer Science Department, Technical Report TR 89-90

[23] E. M McCreight. A space economical suffix tree construction algorithm.*J.ACM,* 23:262-72, 1976

[24] P. Weiner.Linear Patter Matching algorithms. *Proc of the 14th IEEE Symp, on Switching and Automata Theory*, pp, 1-11, 1973

[25] W. Lee, J. Golston, R.J. Gove, and Y. Kim, ``Real-time MPEG Video Codec on a Single-chip Multiprocessor," Digital Video Compression on Personal Computers: Algorithms and Technologies, Proc. SPIE, Vol. 2187, Feb., 1994.

[26] K. Guttag, R.J. Gove, and J.R. Van Aken, ``A Single-Chip Multiprocessor For Multimedia: The MVP," IEEE Computer Graphics & Applications, pp. 53--64, Nov., 1992.

[27] A.Burg, R.Keller, J. Wassner, N. Felber, W. Fichtner: "A 3D-DCT Real-Time Video Compression System for Low Complexity Single-Chip VLSI Implementation", Proc. of the MoMuC2000, p. 1B-5-1, Tokyo 2000