CONTENT AWARE MULTIMEDIA TRANSCODING FOR RESOURCE CONSTRAINED ENVIRONMENTS

by

SIDDHARTHA CHATTOPADHYAY

(Under the direction of Suchendra M. Bhandarkar)

ABSTRACT

The use of multimedia on mobile devices is fast becoming widespread and popular. Since mobile devices are typically resource constrained in terms of network bandwidth, battery power and available screen resolution, it is often necessary to formulate special encoding techniques in order to optimize for power consumption, and network bandwidth, during multimedia data playback and streaming.

This dissertation reports the design and implementation of several novel content aware algorithms for compact representation, and dissemination, of multimedia data suitable for power -and -network constrained environments. The multimedia sub domains of computer animation data, videos, and images, have been considered.

Content aware data processing is a key theme in all the proposed algorithms. Content information for animation data, represented as Motion Capture (MoCap) data, has been derived from the hierarchical structure of the virtual human associated with the data. For video sequences and images, low level content information, such as gradients, motion, curvature etc. have been detected, and exploited, in the proposed algorithms. Another key theme in the proposed algorithms is the elimination, or reduction, of spatio-temporal redundancy, occurring in MoCap and video sequences. The third key theme is the use of domain specific customization of data, in order to render the multimedia data more suited for resource-constrained environments.

Several novel algorithms, based on these three key concepts, have been proposed for MoCap data compression suitable for power-and-network constrained devices. Several content aware image and video transcoding algorithms have been proposed, which transcode images and video sequences as multi-resolution, multi-layered representations, in order to allow power - and - network bandwidth adaptive video playback and dissemination. Results have shown significant power -and -network bandwidth - adaptive capabilities of the videos, which surpass performance of existing standards of layered video encoding. Further, several caching schemes have been developed in order to disseminate videos created using the proposed technologies to power -and network bandwidth - constrained clients over the Internet, resulting in cache designs with improved performance compared to existing cache designs.

INDEX TERMS: MoCap compression, content aware video encoding, BAP-Sparsing, BAP-Indexing, BAP-Sparse-Indexing, FMOE-MR, MMR, Ligne-Claire video, HLV, QAPD, content aware image encoding, layered video encoding, multi-resolution video, layered video caching, cache replacement policy, LGDS

CONTENT AWARE MULTIMEDIA TRANSCODING FOR RESOURCE CONSTRAINED ENVIRONMENTS

by

SIDDHARTHA CHATTOPADHYAY

M.S., Indian Institute of Technology, Kanpur, India (2002)

A Dissertation Submitted to the Graduate Faculty of the University of Georgia in Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

ATHENS, GEORGIA

2007

CONTENT AWARE MULTIMEDIA TRANSCODING FOR RESOURCE CONSTRAINED ENVIRONMENTS

by

SIDDHARTHA CHATTOPADHYAY

DEDICATION

This dissertation is dedicated to my wife, Debjani, for her constant support, encouragement, and for being a patient listener to all my sudden inspirational ideas and discussions at odd hours. I also dedicate this dissertation to my father, for instilling in me the desire to be creative in my thoughts and ideas.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my major professor, Dr. Suchendra M. Bhandarkar, for his support, guidance, and inputs, throughout the process of the research conducted for this dissertation. I am grateful for his many fundamental scientific insights, and for teaching me how to write scientific articles. I also thank him for the many discussions, pertaining to science, philosophy and general knowledge, that he had with me, which made me grow up as an individual who values dedication and creativity in all aspects of life.

I am grateful to Dr. Kang Li for teaching me fundamentals of computer networks, and for his many technical, and philosophical, inputs to my research projects. I am grateful that he is in my committee. He has taught me how to present scientific articles intelligently by knowing the audience.

I am grateful to Dr. Khaled Rasheed for teaching me the fundamentals of machine learning, and for being in my committee. I am also grateful to Dr. Lakshmish Ramaswamy, for teaching me the fundamentals of distributed computing, and for collaborating with me in my research initiatives in multimedia caching.

Last, but not the least, I would like to thank my lab mates, Ananda Chowdhury, Yong Wei and Xingzhi Luo. I am especially grateful to Yong Wei for collaborating with me in the Video Personalization Server caching project. I want to thank Xingzhi for collaborating with me in my FMOE-MR project.

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Multimedia is a form of digital data that uses computer graphics and animation, images, videos, audio etc to represent information. Specialized information processing and encoding is used to represent various types of multimedia contents. One of the main purposes of multimedia is to inform and/or entertain the (user) audience. Multimedia data is typically voluminous, and requires dedicated computational resources in order to process and use it. As a result, it is required to "transcode" multimedia data in order to make it more suitable for devices which have limited resources in terms of storage and computational power. Transcoding is the digital-to-digital conversion from one lossy encoding format to another, typically with the aim to have more compact representation of the data with acceptable loss in quality of the content. Without loss of generality, throughout this dissertation, other terms similar to transcoding, such as compression and encoding, have been used.

Keeping the above definition in mind, this dissertation investigates various novel content-aware algorithms, developed to *transcode* multimedia content, specifically tailored for network and power constrained devices, such as multimedia enabled mobile phones, PDAs and laptop computers in environments where the computer has to run on a battery. In addition, novel multimedia dissemination schemes have been reported which aid in distribution of the multimedia content over the Internet to end users. An end to end software system has been developed to support various forms of multimedia encoding, and their dissemination.

1.1 Background

The sub-domains of multimedia contents that have been addressed in this report are motion capture (MoCap) data for virtual human animation, and video.

Motion capture (MoCap) data is used for digitally recording movements for entertainment, sports and medical applications. A human subject wears markers near each joint to identify the motion by the positions or angles between the markers. Typically, reflective markers are tracked, and the motion capture computer software records the positions, angles, velocities, accelerations and impulses, providing an accurate digital representation of the motion. For this dissertation, the MoCap data has been obtained from the free archive of MoCap data in *http://mocap.cs.cmu.edu.*

MoCap data is extensively used to animate virtual characters in animated movies, video games and virtual reality applications. MoCap data is somewhat analogous to other spatio temporal data such as video. MoCap encoding, similar to video encoding, involves a series of frames, each frame containing information about the joint angles of the model. There is a frame-rate associated with MoCap data, similar to that in a video sequence (typically 30 frames per second). Typically, each second of MoCap information requires 30 frames, each frame being a multi-dimensional vector.

A video is a series of images, taken at a particular frame rate (which, again, is typically 30 frames per second). Each image is typically a two dimensional array of pixels, each pixel having color values defined by its red, green, blue component, or some other three dimensional components from some color space.

The amount of data required for both video and MoCap is typically voluminous. Over the past decade, various techniques have been proposed to encode both, video and

MoCap data, using various state-of-art techniques [Arikan, 2006] [Glardon 2004] [Safonova, 2004]. A detailed literature study of existing methods has been done at the beginning of each chapter. With the recent advance of multimedia enabled mobile devices, such as mobile phones, PDAs etc., video, as well as MoCap, is used in these mobile devices for various entertainment, or research, purposes. A significant drawback with these multimedia mobile devices is the fact that they run on battery, which is a limited power resource. Video playback, as well MoCap data processing, is typically a computationally resource intensive task [Chandra, 2003], which means that the limited battery power resources in these mobile devices drain out quickly, which is not desirable. Thus, it is required to "customize" MoCap, and video, in order to maximize viewer usage and reducing battery consumption in these mobile devices.

This dissertation reports novel content-aware algorithms to compress MoCap, and video, data, suitable for use in resource constrained mobile devices. Content aware data processing is a key theme in all the proposed algorithms. Content information for animation data, represented as Motion Capture (MoCap) data, has been derived from the hierarchical structure of the virtual human associated with the data. For video sequences and images, low level content information, such as gradients, motion, curvature etc. have been detected, and exploited, in the proposed algorithms. Another key theme in the proposed algorithms is the elimination, or reduction, of spatio-temporal redundancy, occurring in MoCap and video sequences. The third key theme is the use of domain specific customization of data, in order to render the multimedia data more suited for resource-constrained environments.

1.2 Research challenges

The nature of data to be encoded has some interesting characteristics which made the conducted research surprisingly challenging and exciting. The following sub-sections detail some of the challenges.

1.2.1. Distortion due to loss

The novel transcoding techniques reported in this dissertation are *lossy* transcoding/encoding algorithms. This means that data loss occurs while transcoding the original data, thus resulting in a slightly *lossy* data after reconstruction from the transcoded version. The loss is inevitable in order to obtain significant reduction in the data size for storage. Loss incurred in MoCap data leads to distortion of the pose and action of the virtual human, on which the MoCap data is used. Lossy data for images and video lead to visual distortion of the image/video contents. Naturally, one of the most challenging aspects in the conducted research is to reduce loss in data, while increasing the amount of compression, of the multimedia data. A related challenge has been to quantify loss. Data loss, or distortion, when measured directly in terms of quantity, does not suffice to reflect the observed loss in quality directly. For MoCap data, we compared spatial displacement errors of body joints. For erroneous data, this spatial displacement is magnified significantly. For images/video frames, peak signal to noise ratio (PSNR) is a common metric used to compare the original image (or video frame) to the reconstructed image (video frame). However, experience reveals that PSNR is often not an accurate measure of quality, and often undermines the quality of a transcoding algorithm. As a result, subjective evaluations, and other non-standard methods derived from computer vision tasks, have been smartly used in order to compare obtained results.

1.2.2. Spatio-Temporal nature of data

A commonality between MoCap and video data is the fact that both are spatio-temporal data; i.e. information content pertains to spatial attributes, which change across time. For example, in a video, the spatial content is the color elements in the image, and the temporal information is the commonality of the color information across series of frames. For MoCap data, the spatial content is the joint angle corresponding to each joint in the virtual character, whereas the temporal content is the change of joint-angles across each frame. A significant challenge of the conducted research has been the use of temporal coherency between successive spatial data, both for MoCap and video data, in order to improve transcoding efficiency. Measuring, and harnessing the use of, temporal data has proven to be non trivial. A common challenge associated with temporal data is to find correspondence. For example, in two consecutive frames of a video, it is non-trivial to find correspondence between features in one frame, to the next. The associated error observed while trying to determine correspondence between temporally dissociated data points can lead to an increase in data loss and precision, which, as discussed earlier, is not desirable at all. Hence, the temporal aspect of the data makes transcoding/encoding more challenging.

1.2.3. Data redundancy

Data redundancy arises when some portions of the data can be omitted without significant change to the information content of the data. Exploitation of data redundancy has been used in almost all existing video encoding algorithms [Richardson, 2004]. A challenge in the conducted research was to utilize these techniques of exploiting data redundancy in the novel proposed transcoding algorithms. For example, in MoCap data, the change in some joint angles is often not substantial enough in a series of frames. These data points are hence redundant. For video data, pixels corresponding to regions in the scene which do not change can often be considered redundant. A key challenge is the determination of redundancy. Often, due to noise, redundancy in real world is not reflected in the corresponding data. For example, in MoCap data, a body joint angle, even if stationary for all practical purposes in the real scenario, is mathematically shown to change over time in the captured data, due to error/loss while measuring. Hence, it is a challenge to come up with automated techniques to determine whether a subset of data is “redundant enough”, so that its omission does not affect the system performance significantly.

1.2.4. Inherent Structure in data (Content Aware)

This is one of the most important aspects of MoCap and video data which led to the development of the novel innovative technologies for MoCap and video transcoding. MoCap data, for example, depends on the underlying structural hierarchy of the virtual human model for which the data is. Each frame of MoCap corresponds to a pose. Any lossy compression of MoCap data leads to a distortion in the pose of the virtual human for that particular frame. Using this structural correlation between the data points, the techniques developed as a part of this research dissertation display superior compression ratio with reduced error in the virtual human model pose. For image and videos, low level content such as edges in images, motion between images etc., have been used to drive the transcoding process.

1.3. Structure of the Dissertation

Including the introduction, this dissertation report is divided into six chapters. The outlines of the remaining chapters are as follows:

Chapter 2: This chapter reports the various content-aware algorithms developed to compress/transcode MoCap data. After reviewing the state of art of MoCap compression, existing global standard, MPEG, is discussed. Three MoCap compression algorithms, used to create lossy compressed version of MoCap data for virtual humans, are proposed, viz. BAP-Sparsing, BAP-Sparse-Indexing and Weighted PCA. These algorithms exploit the structural hierarchy of the virtual human in order to efficiently compress the MoCap data with reduced distortion due to loss. Experiments and observations reveal that these algorithms perform better compared to existing MPEG algorithms, in terms of standard established metrics used for MoCap data.

Chapter 3: This chapter reports novel techniques for content-aware multi-resolution layered image and video encoding, in order to stream video to environments with dynamically changing network bandwidths, such as the Internet. The state-of-art of layered video encoding has been studied in detail. First, a novel content aware multi-resolution technique to transcode an image into multi-resolution representation has been reported. Next, a novel content-aware multi-resolution layered video encoding scheme, termed as FMOE-MR (Features, Motion and Object Enhanced Multi-Resolution) has been proposed and implemented. FMOE-MR uses content aware techniques in order to improve rate-distortion performance of existing, popular layered video encoding paradigms. Results indicate that the proposed multi-resolution, layered video algorithm is suitable for streaming video in resource constrained environments with fluctuating network resources.

Chapter 4: This chapter reports a novel technology termed as Hybrid Layered Video (HLV). HLV is a content aware layered video transcoding algorithm, which is used to create a combination of sketchy representations of the video, termed as Generative Sketch-based Video (GSV), and approximate texture information driven by FMOE-MR, discussed in the previous chapter. HLV is used for video streaming to, or video playback in, power resource constrained devices, whose available (battery) power decrease with time.

Chapter 5: Novel caching techniques and algorithms, used for efficient, low latency dissemination of layered video content, have been reported in this chapter. Smart, customized cache replacement policies have been proposed in order to improve cache performance. Experiments and results confirm that client observed video download latency is considerably reduced by using these smart caching methods.

Chapter 6: This chapter begins with an overview of my contributions to the field of multimedia encoding. Next, the conclusions to the overall dissertation is

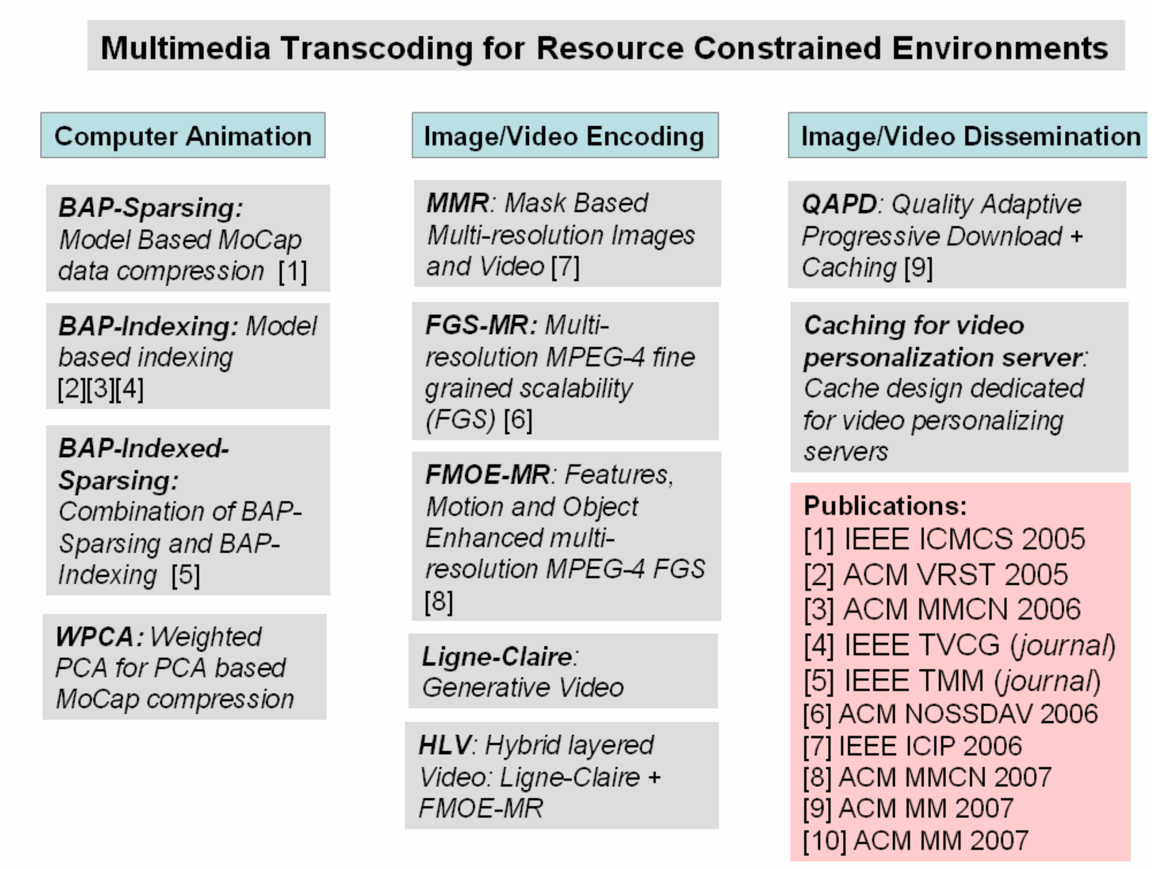

Multimedia Transcoding for Resource Constrained Environments
Computer Animation
Image/Video Encoding
Image/Video Dissemination
BAP-Sparsing: Model Based MoCap data compression [1]
BAP-Indexing: Model based indexing [2][3][4]
BAP-Indexed-Sparsing: Combination of BAP-Sparsing and BAP-Indexing [5]
WPCA: Weighted PCA for PCA based MoCap compression
MMR: Mask Based Multi-resolution Images and Video [7]
FGS-MR: Multi-resolution MPEG-4 fine grained scalability (FGS) [6]
FMOE-MR: Features, Motion and Object Enhanced multi-resolution MPEG-4 FGS [8]
Ligne-Claire: Generative Video
HLV: Hybrid layered Video: Ligne-Claire + FMOE-MR
QAPD: Quality Adaptive Progressive Download + Caching [9]
Caching for video personalization server: Cache design dedicated for video personalizing servers
Publications:
[1] IEEE ICMCS 2005
[2] ACM VRST 2005
[3] ACM MMCN 2006
[4] IEEE TVCG (journal)
[5] IEEE TMM (journal)
[6] ACM NOSSDAV 2006
[7] IEEE ICIP 2006
[8] ACM MMCN 2007
[9] ACM MM 2007
[10] ACM MM 2007

Figure 1.1 The various algorithms developed for this dissertation, and the various corresponding publications.

mentioned, followed by a discussion on possible future work based on the developed algorithms.

1.4 Publication of research articles

Most of the technologies described in this dissertation have been published in highly rated, popular peer-reviewed conferences and journals. **Figure 1.1** shows an overview of the work done, and the various resulting publications, in order to familiarize the reader with the names of the algorithms, and the broad sub-domains to which they belong to.

CHAPTER 2

MOTION CAPTURE DATA COMPRESSION

2.1. Introduction to MoCap

Motion capture (MoCap) data is used for digitally recording movements for entertainment, sports and medical applications. A (human) performer wears markers near each joint to identify the motion by the positions or angles between the markers. Typically, reflective markers are tracked, and the motion capture computer software records the positions, angles, velocities, accelerations and impulses, providing an accurate digital representation of the motion.

2.1.1. Standardizing MoCap data representation - MPEG-4

In order to standardize representation of MoCap data, the MPEG-4 standard seeks efficient representation and encoding of synthetically generated audiovisual information, such as virtual humans [ISO/IEC-Systems, 1999] [ISO/IEC-Visual, 1999]. A virtual human body model is animated using a stream of body animation parameters (BAPs), which are essentially obtained from Motion Capture data. BAP encoding is particularly suited for low-bitrate transmission in dedicated interactive communications and broadcast environments. The BAPs control the various independent degrees of freedom in the skeletal model of the body to produce animation of the body parts. The BAP data enable real time transmission, rendering and manipulation of life-like visual scenes of the human body on a remote device without the need for transmission of the pictorial and video details of the human body in every frame. An overview of body encoding, using MPEG-4 and MPEG-4 compliant avatar control, is given in [Capin, 1999].

2.1.2. MoCap data representation as a matrix

The Motion Capture Data, or MPEG-4 BAPs, are represented as an n x m dimensional matrix **X**, where n is a multiple of the frame rate (expressed as frames per second or fps) and m is the number of degrees of freedom for the avatar (the maximum value of $m = 296$ as defined in MPEG-4 standard). Each row of the matrix represents a pose of the avatar for a small time step. Each column corresponds to either the displacement of the model from a fixed origin, or the *Euler angle* of rotation of a particular joint in the skeletal avatar to achieve the desired pose during the corresponding time interval. Successive rows of **X** depict incremental changes in the pose of the avatar in small time steps, thus animating the avatar.

A 62-dimensional avatar, with a frame rate of 33 fps, has been used in the studies. This means that, for a 10 second motion sequence, the motion matrix **X** is a 330 x 62 array of floating point numbers. The first 3 columns of **X** represent the absolute displacement of the avatar from a fixed origin in the 3-D virtual world. The next three columns represent the absolute orientation of the avatar with respect to the virtual world coordinates. As a first step in the compression process, the matrix **X** is represented by a difference matrix, $\mathbf{d}_{n-1 \text{ x } m}$, and the initial pose vector **I**, where **I** is assigned the first row of **X**, and the rows of **d** are the differences between successive rows of **X**.

$$\mathbf{I}_{1,j} = \mathbf{X}_{1,j} \qquad j = 1, 2, \ldots, m$$

$$\mathbf{d}_{i,j} = \mathbf{X}_{i+1,j} - \mathbf{X}_{i,j} \qquad i = 1 \ldots n\text{-}1;\, j = 1 \ldots m$$

The difference matrix **d,** subsequently termed the motion matrix**,** can be interpreted as successive small angular changes needed by the avatar for each of its degrees of freedom in order to realize the desired animation. Without loss of generality, we will assume that **d** has *n* rows.

Having decided the matrix representation, three methods, viz. BAP-Sparsing, BAP-Sparsed-Indexing and Weighted PCA will be discussed in the subsequent section. Each of these methods is aimed at compressing the BAP data by exploiting the hierarchical structure of the virtual human figure which is being animated, such that the resulting distortion is less.

2.2. BAP-Sparsing

An important application of BAP based animation is the streaming of the BAP data across the Internet to clients, such that real time animation is possible. A practical use of such an animation is in intelligent next-generation distributed human-computer interaction. Issues pertaining to video streaming over the Internet have been well researched [Wu, 2001] [Kang, 2002] [Elsen, 2001] [McNamee, 2000], and a standard compression pipeline for reduced throughput has been established over time. Recent standards in MPEG-4 allow streaming of BAPs over the Internet. Compression of BAP data for efficient network transmission with low bit-rates can be represented by a pipeline of stages that yields a lossless compressed version of the original data [Capin, 2000b]. A lossy adaptation of the existing BAP compression technique includes frame dropping, which results in loss of motion animation quality. A major drawback of the conventional lossy compression pipeline that uses frame dropping is the lack of intelligent, data-and model-dependent compression; i.e. the inherent structure of the underlying model is not utilized. Hence, it is desirable to have a more sophisticated frame dropping algorithm that exploits the structure of the underlying model.

Keeping this philosophy in mind, the first novel technique used for content aware MoCap (or BAP) compression is termed as BAP-Sparsing. In this technique, animation frames are dropped and modified intelligently by exploiting the hierarchical structure of the human skeletal model. The proposed method results in a higher compression ratio compared to standard MPEG-4 BAP compression pipeline, and enables control of the motion animation quality via a single tunable parameter.

As mentioned earlier, the MoCap data is represented by a matrix consisting of a single I-frame, followed by a sequence of P-frames. The given compression algorithm results in a modified version of the matrix with many elements of the matrix reduced to zero (sparsing). The motion resulting from the new matrix is observed to be minimally distorted compared to the motion derived from the original matrix (Section 3). The sparse matrix is the offline emulation of P-frame dropping, and leads to an improved compression ratio, resulting in a lower network throughput requirement for streaming the motion data. The performance comparison is done essentially between the animations resulting from the use of the sparse matrix and the non-sparse matrix.

2.2.1. MPEG-4 BAP Compression

The Face and Body Animation (FBA) object is a collection of nodes in a scene graph, which are animated using two separate FBA object bit-streams, called BIFS (Binary Format for Scenes). The first bitstream contains instances of Body Definition Parameters

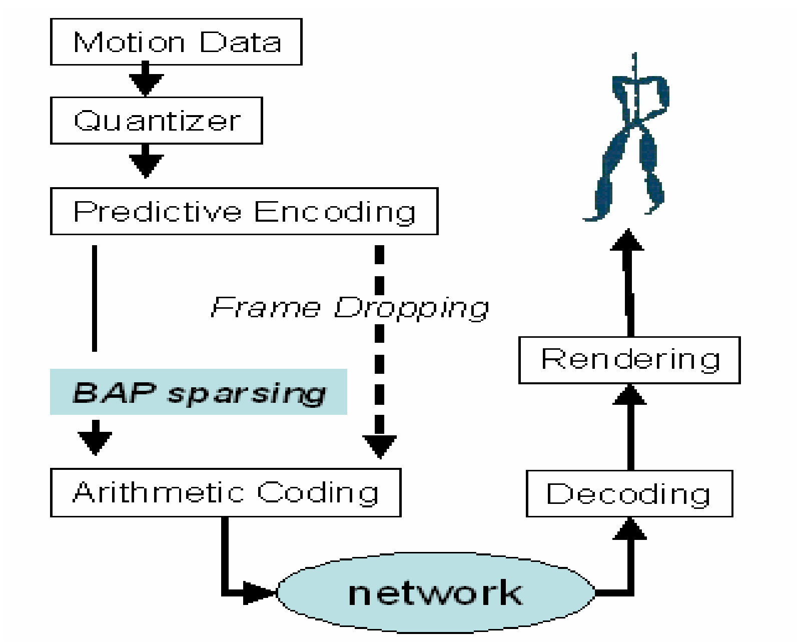


Figure 2.2.1: The standard compression and decompression pipeline for MPEG-4 BAPs, enhanced with a novel alternative stage termed as *BAP sparsing*.

(BDPs) in addition to Facial Definition Parameters (FDPs), and the second bitstream contains Body Facial Animation Parameters (FAPs) [Capin, 2000a].

The BAPs are compressed for efficiency using a standard compression pipeline comprising of quantization taking into account the physical constraints of the joints, representation using prediction errors and finally arithmetic coding for generic bit-level compression. To achieve further compression, frame dropping is used to reduce the number of transmitted frames, consequently reducing the network throughput requirement and hence the bit-rate. Assuming no packet loss, the BAP stream consists of a single I-Frame followed by all the predicted P-Frames necessary to render the complete animation. For a new animation sequence, another set of data, consisting of an I-frame followed by a long sequence of P-frames is streamed.

2.2.2. Error Sparsing Stage in the Compression Pipeline

A novel BAP (error) sparsing stage is introduced between the error encoding and arithmetic coding stages of the standard MPEG-4 BAP compression pipeline (**Figure 2.2.1**). The basic intuition underlying BAP sparsing is to allow the joint corresponding to the BAP parameter to be frozen, for a fraction of a second, and then released all of a sudden. However, once released, the joint corresponding to the BAP parameter will have greater cumulative displacement to make up for the lost displacements when frozen. If performed judiciously, the freeze-release operation should be imperceptible to the human eye.

To implement this idea, each BAP parameter is considered, and consecutive P-frames are dropped, and the values of the dropped P-frames are accumulated until the cumulative value exceeds a predefined threshold. At this point, the current P-frame value is replaced by the cumulative value, and reset (to zero) the variable corresponding to the cumulative value. The same process is repeated for succeeding frames until all the P-frames have been scanned once.

As mentioned previously, the BAP P-frame encoding error data is represented by an n x m matrix $X_{n \times m}$, where n is the number of frames of the animation, and m is the number of BAP parameters to be used for defining a pose of the model (the maximum value for $m = 296$ as defined in the MPEG-4 standard). Initially, the mean m_i and standard deviation s_i of each column (parameter) i is computed. For each instance of each BAP parameter i that passes through the compression pipeline, a variable S_i is used to accumulate the difference of predictive encoding errors, in successive frames. The current encoding error is set to zero (sparsing stage), until the normalized cumulative encoding error ($|S_k$ - m_k $|/$ s_k) exceeds a predefined threshold value Ti. The value of the threshold Ti is determined using information derived from the hierarchical skeletal model of the human, for which the BAPs are being used. The error sparsing algorithm is given in **Figure 2.2.2**. A detailed discussion on the determination of threshold Ti is presented in the next section.

The centered and normalized value of the cumulative predictive encoding ($|S_k$ - m_k $|/$ s_k) is used in order to determine the value of the threshold Ti irrespective of the mean and spread of the data in the columns of **X**. This stage renders the algorithm to be more general, spanning a wide range of motions, and enables the setting of a default value of the control parameter.

```
for  k ← 1 to m
   $S_{\mathbf{k}} = \mathbf{X}_{1,k}$
for  i ← 2 to n
   for  k ← 1 to m
      $S_{\mathbf{k}} \leftarrow S_{\mathbf{k}} + \mathbf{G}_{i-1,k}$
      if  $((|S_{\mathbf{k}} - \mathbf{m_k}|/ S_{\mathbf{k}}) > \mathbf{T_k})$
         $\mathbf{B}_{i,k} = S_{\mathbf{k}}$
         $S_{\mathbf{k}} = \mathbf{X}_{i,k}$
      else
         $\mathbf{B}_{i,k} = 0$
for  i ← 2 to n
   for  k ← 1 to m
      $\mathbf{X}_{i-1,k} = \mathbf{B}_{i,k}$
```

Figure 2.2.2: The algorithm for *BAP Sparsing*, assuming the P-Frames are in a matrix format $X_{n \times m}$. $G_{i,k} = X_{i,k} - X_{i-1,k}$. B is a temporary matrix, whose values are finally assigned to X.

2.2.3. Determination of the Threshold Ti

The threshold value for each BAP parameter is computed by exploiting the hierarchical structure of the underlying human skeletal model. The basic observation that has been

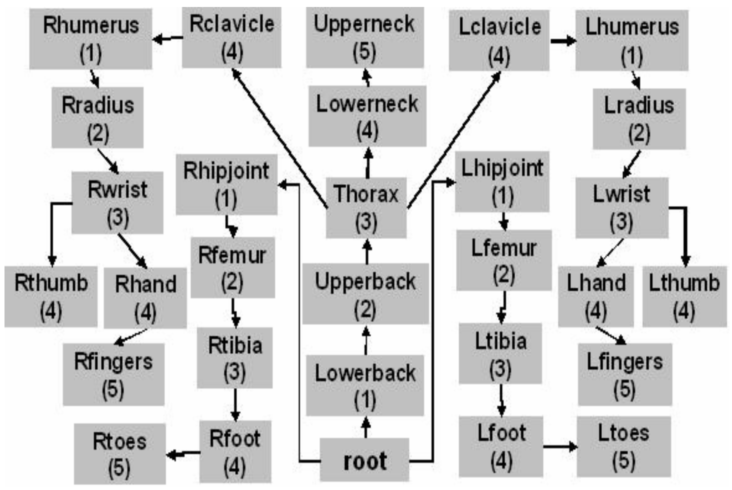


Figure 2.2.3: An example of hierarchical structure of a human consisting of 31 nodes, with a total of 62 degrees of freedom of motion (rotational and translational). For convenience, the root node is drawn at the bottom.

utilized is the fact that the threshold value for a particular BAP parameter should depend on the position of the corresponding body joint in the human skeletal model. For example, consider the hierarchical human skeletal model given in **Figure 2.2.3**. Any joint higher in the hierarchy is allowed less angular error compared to joints lower down in the hierarchy. For example, an angular error at $R_{hipjoint}$ displaces the joint R_{foot} more than the same angular error at the joint R_{tibia}, which is lower down in the hierarchy.

We have represented the hierarchical significance of the various joints in the human body by a level number, as shown in the parentheses in **Figure 2.2.3**. The threshold Ti for a joint *i*, represented by column *i* in matrix $\mathbf{X}_{\mathbf{n\ x\ m}}$, should be such that the joints higher in the hierarchy, with a smaller level value, should be allowed less angular error due to sparsing of the predictor error, when compared to joints with a larger level value. This ensures that the displacement error induced by the sparsing operation is small. For a column *i* of $\mathbf{X}_{\mathbf{n\ x\ m}}$, with level Li, the corresponding threshold Ti is given by

$$\mathrm{Ti} = K\mathrm{Li} \qquad (2.2.1)$$

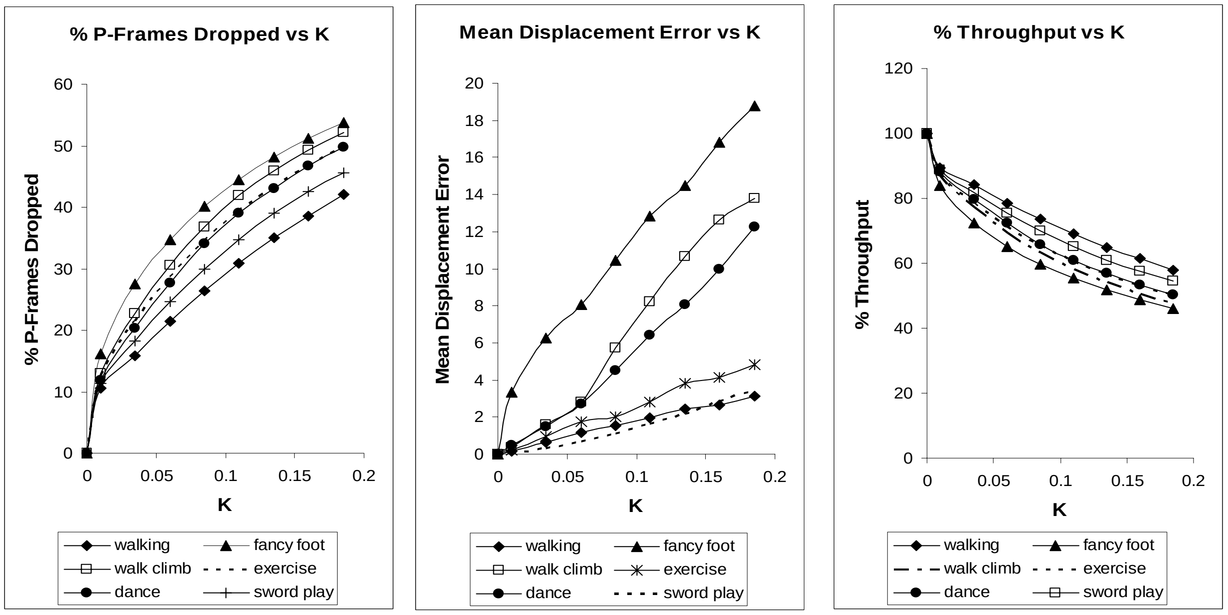


Figure 2.2.4: (Left) Percent P-Frames dropped vs Quality Control Parameter K (Center) Mean Displacement Error vs K (Right) Network throughput requirement vs K. Walking and exercise yield similar throughputs, hence the lines are superimposed.

where K is the quality control parameter (QCP). Typically, the value of *K* is chosen such that the maximum threshold value is less than 1.

2.2.4. Animation quality, network throughput requirement and power consumption

The sparsing algorithm has been tested for different motions of varied duration and complexity. We present our results on the following representative motion examples:

- Walking: Simple periodic motions.
- Sword Play: Simple non-periodic motions.
- Walking-Climbing: Complex but slow motions.
- Exercises: Motions with multiple complexities.
- Dancing: Slow, but ill-correlated motions.

- Fancy Footwork: Fast, erratic and extremely ill-correlated motions, where the subject rolls on the floor and performs complex dance motions very rapidly.

The dependence of the percentage of P-Frames dropped, the mean displacement error and the percentage network throughput requirement for the above mentioned motion examples, as a function of the quality control parameter K, has been given in **Figure 2.2.4**. The percentage network throughput requirement is simply the fraction of the original number of bytes to be transferred over the network. The mean displacement error is the cumulative displacement error induced by sparsing, normalized with respect to the number of frames in the motion sequence.

The mean displacement error depends on the nature of the motion. As shown in **Figure 2.2.4**, complex motions such as a combination of motions (walking and climbing) and erratic motions (fancy footwork) yield more noise/distortion in the motion data, compared to regular motions such as walking. The percentage network throughput requirement is greater for regular motions such as walking, whereas it is less for erratic and sudden motions such as fancy footwork. Although this seems counter-intuitive, it makes sense when the mean displacement error is also considered; for the same value of K, the animated motion results in different mean displacement errors depending on the type of motion. This means that the quality control parameter K is not an absolute universal measure of motion animation quality; the more erratic or complex the motion, the greater is the mean displacement error for a given value of K. Ideally, this should not be the case. However, at this juncture, it should also be noted that K, though not a perfect and universal quality control parameter, is adequate for most types of motions encountered in practice.

Table 2.2.1: Comparison of the degree of compression and number of CPU cycles required to encode and decode motion data, with simple arithmetic coding, and arithmetic coding after *BAP sparsing*. The BAP sparsing step prior to arithmetic coding yields superior results in both cases. QCP K = 0.1 for all the experiments.

Motion	No. of Frames	Arithmetic Coding (KB)	BAP Sparsing + Arithmetic Coding (KB)	% reduction in file size	CPU Giga Cycles Encoding Original	CPU Giga Cycles Encoding Sparsed	CPU Giga Cycles Decoding Original	CPU Giga Cycles Decodin g Sparsed	% reduc-tion in CPU Cycles
Walking	343	83	63	75.9%	1.341	1.031	0.928	0.756	81.5%
Dance	434	104	73	70.2%	1.684	1.169	1.169	0.859	73.5%
Sword Play	2251	515	386	75.0%	8.491	6.394	5.981	4.503	75.3%
Fancy Foot	2555	586	448	76.5%	9.625	6.394	6.841	4.538	66.3%
Exercise	4653	1024	743	72.6%	16.981	12.272	11.928	8.697	72.9%
Walk, Climb	4839	1082	776	71.7%	17.864	12.684	13.166	8.972	68.1%

Since the motion animation quality, throughput requirement, and percentage of P-frame BAPs dropped are all dependent on the quality control parameter *K*, it is desirable to have a default, "safe" value of *K*, which guarantees an acceptable level of displacement error that is not obviously perceptible. Additional experiments and empirical studies with human observers have shown that choosing $K = 0.1$, and hence ensuring that the normalized cumulative predictive error (P-Frame values) is less than 1, yields negligible motion distortion for all the examples we have studied.

Since the MPEG-4 standard uses arithmetic coding as the final stage for compression of the BAP data, we have implemented and tested the amount of compression for various motion data, having processed them through the enhanced pipeline incorporating BAP sparsing, as depicted in **Figure 2.2.1**. We have obtained a significant amount of compression after the final arithmetic coding stage, with no visual distortion of the motion, with $K = 0.1$. The test results for some of the representative motions are given in **Table 2.2.1**.

For a power constrained device, both encoding and decoding of motion data using arithmetic coding are critical sources of power consumption in the case of streaming motion data. To estimate power consumption, the number of CPU cycles needed to encode and decode the motion data for the representative motion examples mentioned above has been computed. A 2.2 GHz Intel Celeron Processor with 128 KB L2 cache and 512 MB RAM for running the arithmetic coding and decoding algorithms has been used. As shown in **Table 2.2.1**, BAP-Sparsing requires fewer CPU cycles for both the encoding and decoding process, from which it is safe to infer that the proposed method saves power during both encoding and decoding of the streaming motion data.

Trading bandwidth and CPU resource consumption with data quality has been a common practice, especially for media streaming over the Internet. The MPEG standard has been designed with the ability of making data quality adaptations by simply performing frame dropping. There exists considerable published research literature on this topic [Rowe, 1994] [Krasic, 2003] [Zhang, 1999]. When compared to data quality adaptation using traditional frame dropping adaptations, BAP sparsing can be considered to be a "smart" data dropping technique. Instead of skipping frames and simply discarding the motion data, BAP sparsing reduces the amount of transmitted data by simplification and aggregation of the motion data to approximate the same underlying motion. BAP sparsing can achieve this by taking advantage of the knowledge of the moving objects and their internal physical representation.

BAP sparsing has a significant advantage over the simple frame dropping technique, especially for animation video. For example, in the case of traditional MPEG random frame dropping, once a P-frame is dropped, all its subsequent frames within the same GOP and even some B-frames, transmitted before the dropped P-Frame that refer to the P-Frame, have to be skipped because of decoding failure. For a normal video clip, because of the relatively small GOP size [Feng, 2003], the quality degradation caused by P-frame dropping is constrained to lie within a very short time interval (determined by the GOP size). The video quality can recover to its original value when the first frame of the next GOP arrives. Unfortunately, the BAPs are organized as a single long GOP with a single I-frame at the beginning of the sequence followed by several P-frames. This makes traditional frame dropping unsuitable for data quality adaptation in the standard BAP compression pipeline, since the quality of BAP data is severely compromised. BAP Sparsing, on the other hand, understands the motion data and the associated skeletal model hierarchy, and keeps track of the accumulated error so that P-frames can be skipped and modified without affecting future frames.

2.2.5. Future Improvements

There is scope for future improvements in the proposed method. Although the current default value for the QCP K = 0.1 yields satisfactory results, it is desirable to obtain an optimal value for the QCP K via statistical analysis of the underlying motion data. A good metric for quantification of the quality of motion is needed, as the current measure based on mean displacement error does not map to any direct measure of the visual quality of motion animation.

2.3. BAP Sparse-indexing

Animation of human-like virtual characters has potential applications in the design of human computer interfaces, computer games and modeling of virtual environments using power-constrained devices such as laptop computers in battery mode, pocket PCs and PDAs [Gutiérrez, 2003] [Kruppa, 2003]. Distributed virtual human (avatar) animation is used in many applications that depict human models interacting with networked virtual environments [Capin, 1999]. Distributed virtual environments (DVEs) either require exchange of motion files between hosts to simulate the avatar motion, or use locally stored motion data [Joslin, 2000] [Cavazza, 2003] [Vacchetti, 2003] [Barakonyi, 2004] [Capin, 1997]. In order to standardize avatar animation, MPEG-4 has proposed H-Anim standards to represent avatars [ISO/IEC-systems, 1999] [ISO/IEC-visual, 1999] [Preda, 2004]. An avatar is animated using a stream of body animation parameters (BAPs) encoded for low-bitrate transmission [Capin, 1999b], using the MPEG-4 compression pipeline (**Figure 2.2.1**), in dedicated interactive communications and broadcast environments [Capin, 2000a] [Preda, 2002] [Preda, 2001] [Capin, 1997]. The BAPs control the various independent degrees of freedom in the skeletal avatar model to produce an animation of the body parts [Capin, 2000b].

The two major issues in MPEG-4 BAP based animation in mobile devices are

(a) Limited bandwidth available for streaming BAP data, and

(b) Limited power available to process the animation data.

Network overload, while streaming, can be reduced by using smart techniques such as dead reckoning [Capin, 1997], and BAP quantization and grouping [Capin, 1999b]. Although these techniques reduce the network load significantly, decompression of the data at the client end entails extra CPU cycles. To reduce the power consumption resulting from the CPU cycles needed for decompression, raw data may be used. However, the direct use of raw data is undesirable as the network may become too overloaded. Hence, it is desirable to have a compression method which reduces the network load significantly, and also requires minimum computation, hence power consumption, at the client side to reconstruct the motion data from the compressed data stream. In addition, the hierarchical structure of the skeletal avatar model needs to be judiciously exploited in the case of lossy compression such that any undesired reduction in motion animation quality is minimized.

A combination of two novel compression algorithms for MPEG-4 BAP data is reported in this section, which (a) intelligently exploit the structural hierarchy of the virtual human avatar to achieve efficient compression which, though lossy, results in reconstructed motion of good quality, (b) use indexing techniques for compression of BAP data, resulting in significant reduction in power consumption required for decompression, and (c) provide quality control parameters for tighter control of the reconstructed motion quality and compression ratio. One of the proposed compression algorithms termed as BAP-Sparsing [Chattopadhyay, 2005], creates a sparsed representation of the original BAP data. This results in improved compression of the BAP data upon using the MPEG-4 compression pipeline (**Figure 2.2.1**). A detailed description of BAP-Sparsing has been given in the previous section. The other proposed algorithm, BAP-Indexing [Chattopadhyay, 2006], creates byte-size indices for representation of the BAP data. BAP-Indexing results in the compression of the BAP data, since the indices can be represented using fewer bits, compared to the original floating point representation of the BAPs. The resulting compression ratio is significantly superior to that obtained under similar conditions using the MPEG-4 compression standard [Chattopadhyay, 2006]. A combination of the two algorithms mentioned above, resulting in a sparse, quantized representation of the original BAP data, has been reported. The resulting hybrid algorithm yields significantly better compression ratio compared to those obtained using the MPEG-4 standard, and requires significantly less client-side power measured in terms of CPU cycles and energy (measured in mJoules) needed to receive and decompress the data. Standard MPEG-4 techniques such as grouping can be used atop the proposed technique to further reduce the bit rate, as can be done in the case of MPEG-4 based BAP compression.

There exist quantization methods for efficient use and distribution of, avatar motion data over the network. Endo et al. [Endo, 2003] propose quantization of the motion type, rather than the motion data itself. Hijiri et al. [Hijiri, 2000] describe a new data packet format which allows flexible scalability of the transmission rate, and a data compression method, termed as SHCM, which maximizes the efficacy of this format by exploiting the 3D scene structure. The reported method uses quantization to achieve data compression in a manner somewhat similar to the above chapter, but incorporates intelligent exploitation of the hierarchical structure of the human skeletal model. Giacomo et al. [Giacomo, 2003] present methods for adapting a virtual human's representation and the resulting animation stream, and provide practical details for the integration of these methods into MPEG-4 and MPEG-21 architectures. Aubel et al. [Aubel, 1998] present a technique for using impostors to improve the display rate of animated characters by acting solely on the geometric and rendering information.

The known techniques mentioned above do not describe any direct impact on the power consumption of the client device on which the animation is being rendered. Also, there is not sufficient quantitative analysis of the quality of the rendered motion upon decompression of the compressed motion data. The algorithm proposed in this chapter not only allows low-bitrate encoding of motion data, but is also suitable for data reception and data reconstruction on power-constrained devices. Henceforth, the proposed algorithm will be termed as *Sparse-indexing* in the remainder of the chapter.

2.3.1. BAP-Indexing: Indexing of BAP data

The basic concept underlying the proposed indexing technique is to be able to index some (perhaps all) of the numbers within the original motion matrix **d**, and generate a corresponding lookup table for the indices. This compression method results in significant data reduction, as each index value can be represented using fewer bits than that the corresponding floating point number.

*2.3.1.1 Indexing Motion Matrix **d***

In order to ensure efficient indexing, we have used the standard equal frequency distribution technique to uniformly assign the $n \bullet m$ numbers in **d** to buckets numbered from 0 to 255. This is done as follows:

Step 1: The floating point numbers in matrix **d** are collected into a single 1-D array **A** of size $n \bullet m$. The array **A** is sorted in ascending order. All the numbers in **A** are multiplied by the *resolution quantization term (RQT)*, *M*. The RQT depends on the number of significant digits used to represent the floating point number. The numbers are rounded to represent integers in the range $[A_{min}\bullet M, A_{max}\bullet M]$.

Step 2: The integers in the range $[A_{min}\bullet M, A_{max}\bullet M]$ are divided into buckets numbered from 0 to 255. It is desirable to allocate each of the 256 buckets an equal share of $n\bullet m$ numbers in **A**. This implies that each bucket should have $freq = (A_{max}\bullet M - A_{min}\bullet M)/256$ numbers allocated to it. This is done by computing the histogram of the integers in **A**, and dividing the histogram into 256 vertical strips such that each strip has the same area, *freq*. After all the numbers in **A** have been allocated to a bucket numbered from 0 to 255, the numbers in **A** are divided by the *RQT* to recover the original numbers.

At the end of this step, we get a set of 256 buckets denoted by **bucket(j)** for $j = 0$ to 255, such that each floating point entry in the motion data matrix **d** is contained in exactly one of the 256 buckets. An index matrix $\mathbf{d_{index}}$ is used to store the bucket number for the corresponding entry in the matrix **d**.

2.3.1.2. Lookup Table for $\mathbf{d_{index}}$

The creation of an appropriate lookup table for the recovery of the original motion data matrix **d** from the index matrix $\mathbf{d_{index}}$ is critical, since recovery of the original data after discretization invariably results in motion distortion. A straightforward method to recover the number associated with a bucket is to compute the arithmetic mean of all the floating point numbers assigned to the bucket. However, this invariably leads to poor approximation of the original motion matrix **d**. We have intelligently exploited the hierarchical structure of the skeletal avatar model to construct the lookup table $\mathbf{T_{lookup}}$ as follows.

Step 1: The avatar is represented by a hierarchical skeletal model. For each *m*-dimensional pose vector (or row in **d**), each dimension, or column in **d**, is assigned a level l_i (**Figure 2.2.3**). The level l_i signifies the importance of the degree of freedom associated with a particular joint, in the overall displacement of the model joints. A joint *i*, at level $l_i = 1$, when given a small angular displacement, affects the model more in terms of the overall displacement, than a joint *j* at level $l_j = 2, 3, 4, 5$ or *6*.

Step 2: After assigning level values to the various joints of the avatar model, these joint level values are used to compute a weighted sum of the numbers belonging to a bucket. The *j*th lookup value in lookup table $\mathbf{T_{lookup}}$ is given by:

$$T^{\eta}_{lookup}[j] = \frac{\sum_{i \in bucket(j)} \frac{1}{(l_i)^{\eta}} A[i]}{\sum_{i \in bucket(j)} \frac{1}{(l_i)^{\eta}}} \qquad j = 0, 1 \ldots m \tag{2.3.1}$$

where η is a constant. Empirical observations have revealed that as η increases, the $\mathbf{T_{lookup}}$ values result in a better approximation to the data, resulting in reduced displacement error. This is due to the fact that the numbers associated with *level = 1* affect the displacements in the body the most. Hence, emphasizing the numbers within a bucket with *level = 1* leads to better approximation of the motion data. As $\eta \rightarrow \infty$, all the weighting terms in **equation (2.3.1)** tend to zero, except for the terms with *level =1*. Hence, when computing the weighted sum of the numbers in a bucket, we consider only those numbers with *level = 1 (selective averaging)*, and compute a simple mean of these numbers. If none of the entries in a bucket have *level =1*, we use the next smallest level to compute the weighted sum.

There is an inherent tradeoff between the size of the motion data that is indexed and the quality of the reconstructed motion. The longer the motion sequence, the lower

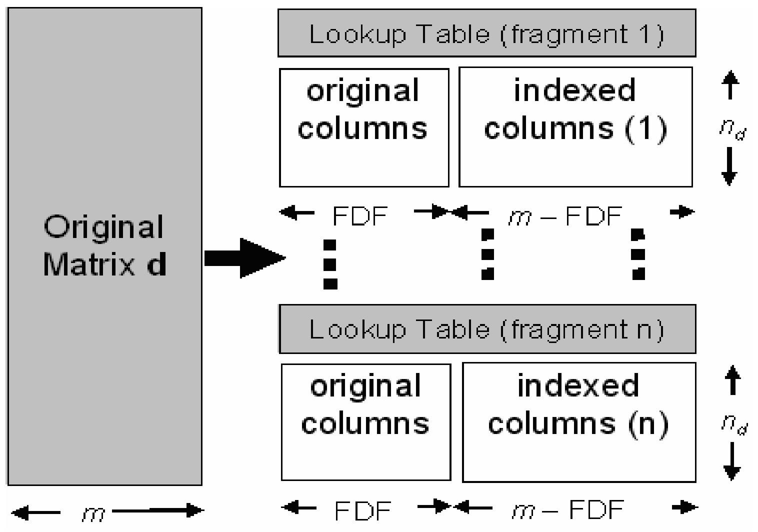


Figure 2.3.1: BAP-Indexing. The motion matrix **d** is decomposed into equal fragments of length n_d each. The first **FDF** rows are untouched; the next m – **FDF** columns are indexed. Each fragment is indexed separately, with its own lookup table.

the reconstructed motion quality after indexing. To overcome this problem, the original difference matrix **d** is fragmented into a succession of smaller matrices $\mathbf{d}_1$, $\mathbf{d}_2$, …, $\mathbf{d}_L$. Each fragmented motion matrix has n_d rows. Each of the fragmented matrices of size n_d x m in turn is discretized using the above method resulting in separate lookup tables, $\mathbf{T}_{\mathbf{lookup}}^{1}$, …, $\mathbf{T}_{\mathbf{lookup}}^{L}$. This simple technique yields a good approximation to the motion data of any arbitrary duration.

2.3.1.3. Combining the original and indexed data

It is often desirable to preserve the original data contained within some of the columns of the original motion matrix **d**, and index the remainder of the columns. This is especially true for the first six columns of **d**, which represent the absolute displacement and orientation of the avatar with respect to a fixed origin. Indexing these columns may lead to undesirable motion distortion, and consequently result in contradictory physical appearances such as the avatar's feet not touching the ground while walking, etc. A combination of the original and indexed columns can be achieved by not indexing the first **FDF** (Fixed Degrees of Freedom) columns, and indexing the remainder of the *m* – **FDF** columns. **Figure 2.3.1** depicts the indexing of the original motion matrix **d**.

2.3.2. BAP-Sparsing: Sparse representation of indexed matrix

Once the difference matrix **d** is indexed, we now use the second algorithm, *BAP-Sparsing*, in order to eliminate several indices from the index matrix. The basic intuition underlying *BAP sparsing* is to allow the joint corresponding to the BAP parameter to be *frozen*, for a fraction of a second, and then *released* all of a sudden. However, once released, the joint corresponding to the BAP parameter will have greater cumulative displacement to make up for the lost displacements when frozen. If performed judiciously, the *freeze-release* operation should be imperceptible to the human eye.

2.3.2.1. Implementation of Sparsing Algorithm

To implement BAP-Sparsing, for each column *i* of the motion matrix **d**, we consider each floating point number corresponding to the BAP index, and drop (reduce to zero) consecutive indices along column *i*, and accumulate the floating point values corresponding to the dropped indices until the cumulative value exceeds a predefined *threshold* $\mathbf{T_i}$. At this point, we replace the current index by the corresponding index of the cumulative value, and reset (to zero) the variable corresponding to the cumulative value. This process is repeated until all the rows in the column are either reduced to zero or replaced by a corresponding index for the cumulative value (**Figure 2.2.2**). A detailed discussion on the determination of the threshold value $\mathbf{T_i}$ is presented in the next section.

The centered and normalized value of the cumulative predictive encoding, $|\mathbf{S_k} - \mathbf{m_k}| / \mathbf{s_k}$ (see **Figure 2.2.2**) is used so that the value of the threshold $\mathbf{T_i}$ can be determined irrespective of the mean ($\mathbf{m}_k$) and spread ($\mathbf{S}_k$) of the data in the columns of **d.** This stage renders the algorithm more general, enabling it to span a wide range of motions, and also enables the setting of a default value of the control parameter.

2.3.2.2. Determination of the threshold T_i

We hypothesize that the threshold value $\mathbf{T_i}$ for a particular BAP parameter should depend on the position of the corresponding body joint in the hierarchical human skeletal model. We have represented the hierarchical significance of the various joints in the human body by a level number, as shown in the parentheses in **Figure 2.2.3**. The threshold $\mathbf{T_i}$ for a joint *i*, represented by column *i* in matrix $\mathbf{d}_{n \times m}$, should be such that the joints higher in the hierarchy, with a smaller level value, should be allowed *less angular error* due to sparsing of the prediction error, when compared to joints with a larger level value. This ensures that the *displacement error* induced by the sparsing operation is small. For a column *i* of **d**, with level L_i, the corresponding threshold $\mathbf{T_i}$ is given by:

$$\mathbf{T_i} = KL_i \qquad (2.3.2)$$

where K is another quality control parameter (QCP). Typically, the value of $\boldsymbol{K}$ is chosen such that the maximum threshold value is less than 1.

2.3.3. Effect of QCP on compression ratio and animation quality

The two algorithms, *BAP-Indexing* and *BAP-Sparsing* when combined, as mentioned above, have four associated quality control parameters (QCPs). The bounds for the QCPs are as follows: $1 \leq \mathbf{FDF} \leq m$ (where m = number of columns in motion matrix $\mathbf{d}$), $8 \leq b \leq 32$ (assuming byte boundary for the indices), $\mathrm{n_d}^{\mathrm{min}} \leq n_d \leq n$ and $0 < K < \infty$.

*2.3.3.1. Compressed file size as a function of **FDF**, b, n_d and K*

In practice, the BAPs in motion matrix $\mathbf{d}$ are represented by floating point numbers. Assuming that floating point numbers take 4 bytes each, the original raw file size for $\mathbf{d}_{\mathrm{nxm}}$ is $4 \bullet m \bullet n$. We first compute the file size assuming no sparsing, and then introduce sparsing to compute the file size as a function of all four QCPs (**FDF**, n_d, b and K).

1) File size assuming no sparsing ($K = 0$): Assuming no sparsing (i.e. sparsing parameter $K = 0$), the resulting file size is obtained simply by expressing the number of bytes as a function of the three indexing parameters **FDF**, n_d and b. In the case of a motion data file that has been decomposed into smaller segments, each segment contains a header and a data section. The header file for each segment (**Figure 2.3.1**) takes $4 \bullet 2^b$ bytes for the lookup table, and a few extra c_1 bytes to store the numbers n_d and b. Hence, the total number of bytes taken by the header for a motion segment is given by:

$$\text{seg\_head} = 4 \bullet 2^b + c_1 \quad (2.3.3)$$

For the actual motion data, the first **FDF** columns ($0 \leq \mathbf{FDF} \leq m$) are floating point numbers taken directly from the motion matrix $\mathbf{d}$, and the next $m - \mathbf{FDF}$ columns are indices for the lookup table, where each index takes a maximum of b bits ($\lceil b/8 \rceil$ bytes). Hence, the total number of bytes needed for the motion data segment is given by:

$$\text{seg\_mot} = n_d \bullet (4 \bullet \mathbf{FDF} + \lceil b/8 \rceil \bullet (m - \mathbf{FDF})) \quad (2.3.4)$$

For a motion data sequence spanning n frames, the number of blocks, or motion segments, each consisting of n_d frames, is $\lceil n/n_d \rceil$. Hence, the total number of bytes, $\mathbf{T}(\mathbf{FDF}, b, n_d)$ of the semi-indexed matrix is obtained by combining **equations (2.3.3)** and **(2.3.4)**:

$$\mathbf{T}(\mathbf{FDF}, b, n_d) = \lceil n/n_d \rceil \bullet (\text{seg\_ head} + \text{seg\_mot}) + 4m \qquad (2.3.5)$$

The minimum file size $\mathbf{T_{min}}$ with sparsing parameter $K = 0$ is obtained by assigning the values $\mathbf{FDF} = 0$, $n_d = n$ and $b = 8$ (assuming a byte boundary for the indexed data) in **equation (2.3.5)**. The maximum file size, $\mathbf{T_{max}}$, should not exceed the original file size $4 \bullet m \bullet n$. For the parameter values mentioned above, the minimum file size $\mathbf{T_{min}}$ (including the lookup table and initial pose vector) is given by:

$$\mathbf{T_{min}} = (n\text{-}1)m + 4m + c_2 \qquad (2.3.6)$$

where c_2 is the number of bytes needed to store 256 buckets, each containing a 4-byte floating point number; i.e. $c_2 = 1024$.

Since the motion quality decreases with increase in compression ratio, it is desirable to obtain the maximum file size permitted by the network, as a fraction of the original file size. For a desired fraction of the original motion data, the corresponding quality control parameters can be obtained by exhaustively searching the space of all possible parameter values such that the following constraint holds:

$$(n\text{-}1)m + 4m + 1024 \leq \mathbf{T}(\mathbf{FDF}, b, n_d) \leq f \bullet 4mn \qquad (2.3.7)$$

where f (henceforth termed as the *minimum-compression-ratio*) is the fraction of the maximum file size of the original motion data. The parameter values that satisfy **equation (2.3.7)** can be termed as *valid* points in the parameter space.

2) File size with sparsing and indexing (K > 0): In order to obtain a *sparsed-indexed* representation of the motion matrix **d**, it is essential to determine a suitable value for the sparsing parameter K such that the resulting motion quality is not too distorted. A basic heuristic is to maintain the normalized threshold value $\mathbf{T_i} \leq 1$ for each column (joint degree of freedom). This is possible by ensuring that $K \leq 0.2$, since the maximum level value can be 5. Having assigned K an empirically selected value less than 0.2, say $K = 0.1$, the degree of sparsing in the resulting indexed matrix depends on the type of motion. We will present experimental results obtained for $K = 0.1$ for various motion examples to demonstrate how sparsing helps to reduce the file size in **Section 2.3.6**. The total file size is now given by **T**(**FDF**, b, n_d, K), which is a certain function of **T**(**FDF**, b, n_d).

*2.3.3.2. Quality of reconstructed motion vs **FDF**, b, n_d and K*

To quantify the reconstructed motion quality, we use the metric **DEF** (Displacement Error per Frame). Let $\mathbf{C}_{nx3m}$ be the original coordinates of the joints of the human model, and $\mathbf{C'}_{nx3m}$ be the coordinates of the joints of the human model after reconstruction of the motion from the indexed matrix. We define the error function, Δ:I $\rightarrow$ R as follows:

$$\Delta(\mathrm{i}) = \sum_{j=1}^{n} \left\| P_{ji} - P'_{ji} \right\|, \qquad i = 1\,, 2, ..., m \qquad (2.3.8)$$

where $P_{ji} = (C_{j,3i-2}\,, C_{j,3i-1}\,, C_{j,3i})$ and $P'_{ji} = (C'_{j,3i-2}\,, C'_{j,3i-1}\,, C'_{j,3i})$, $i = 1,2, .., m$, and $\|.\|$ is the Euclidean norm defined as $\|(x, y, z)\| = \sqrt{(x^2 + y^2 + z^2)}$. This error metric represents the error at each joint position.

The **DEF** is defined as the sum of the errors for all the joints, normalized by the total number of rows (i.e. frames) in the motion data matrix.

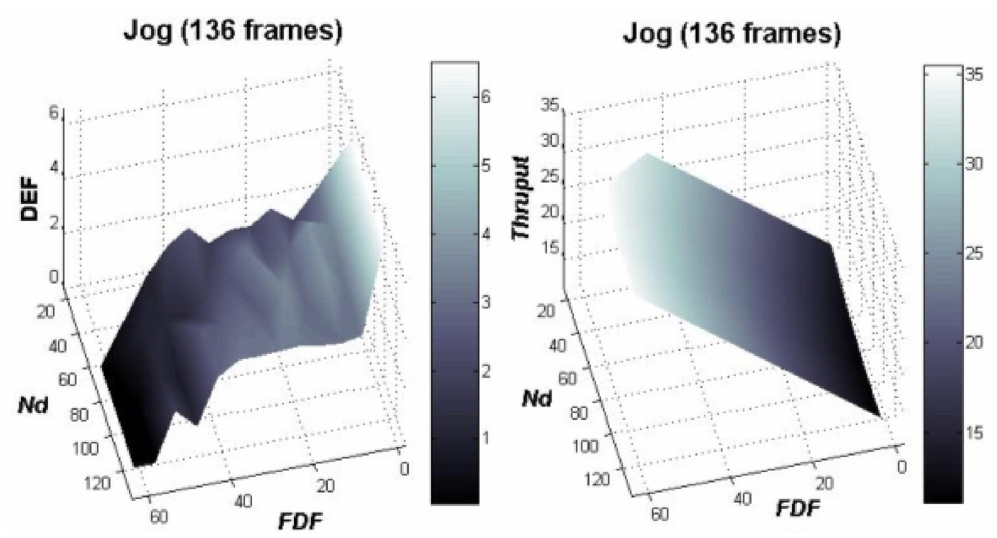


Figure 2.3.2: (**Left**) Displacement error per frame (**DEF**) for $30 \leq n_d \leq 120$ rows and $1 \leq$ **FDF** ≤ 62 columns. (**Right**) Obtained throughput for $30 \leq n_d \leq 120$ rows and $1 \leq$ **FDF** ≤ 62 columns. ($b = 8$, $K = 0.1$).

$$DEF(FDF, b, n_d, K) = \frac{\sum_{i=1}^{n} \Delta(i)}{n} \qquad (2.3.9)$$

Although the **DEF** value does not represent the motion quality in an absolute sense, comparing the **DEF** values for various values of the quality control parameters (QCPs) provides an adequate measure of relative motion quality.

*2.3.3.3. Determining optimal values of **FDF**, b, n_d and K*

In order to compress the BAP data matrix **d** in practice, it is essential to obtain the values for the four QCPs (**FDF**, n_d, b and K). Recall that the value of K is fixed at 0.1 as described in subsection 2.3.5.1**.** Empirical studies reveal that for $b > 8$, the size of the file explodes exponentially, thus violating the upper bound in **equation (2.3.7)** even for $f = 1$ (100%). Hence, we fix $b = 8$, which means that a motion segment will have a 256-bucket lookup table, and that each index value is represented by a single byte. The surface plots of **T**(**FDF**, b, n_d, K) and **DEF**(**FDF**, b, n_d, K), for a *jogging* motion example, for the various possible values of **FDF** and n_d, while fixing the other parameters as $b = 8$ and $K = 0.1$, are presented in **Figure 2.3.2**. As can be observed, the lower the **DEF**, the better is the motion quality. Hence, the set of parameters amongst the valid parameters which minimize both **DEF**(**FDF**, b, n_d, K), and **T**(**FDF**, b, n_d, K), are the optimal quality control parameters (QCPs). Since we have already fixed $b = 8$ and $K = 0.1$, we now need to determine the optimal values for the quality control parameters **FDF** and n_d. These are the values which minimize the following figure of merit function:

$$\mathbf{M}(\mathbf{FDF},b,n_d,K) = \mathbf{T}(\mathbf{FDF},b,n_d,K)\bullet\mathbf{DEF}(\mathbf{FDF},b,n_d,K) \qquad (2.3.10)$$

A simple exhaustive search reveals that **FDF** = 41 and $n_d = 60$ (with $b = 8$ and $K = 0.1$ as fixed values) minimizes the figure of merit function **M** for the *jogging* example. Although the indexing method is inherently lossy, the visual degradation in the reconstructed motion is imperceptible. The functions **T** and **DEF** are inversely proportional to each other. A relationship plot can be used to select the desired throughput requirement **T** based on an acceptable value of **DEF**.

The QCPs can also depend on the display size of the screen on which the animation is being rendered. For small QVGA resolution displays (320 X 240), the amount of error allowed for the joints of the avatar in each frame can be higher, compared to that in the case of higher resolutions such as 640 X 480. This screen size information can be used to *weigh* the **DEF** value appropriately (perhaps empirically) in **equation (2.3.10)** to allow greater quantization (smaller **FDF** and larger n_d), and hence more compact compressed file sizes in the case of lower-resolution displays.

2.3.4. Experimental results for different motion types

In order to test the usefulness of the proposed *Sparse-indexing* technique, we have experimented with various types of motions. The motion types we have selected range from periodic motions (jogging, walking, jumping, long jump etc) to extremely ill-correlated and complex motions such as dancing. We have considered b = 8 (number of bits used to encode an index) and K = 0.1 (the sparsing coefficient) for all of our analysis. The motion examples have been created using motion capture data from real human actors. The motion data files are obtained from the website *mocap.cs.cmu.edu*.

2.3.4.1. Compatibility of Sparse-indexing with existing BAP encoding technologies:

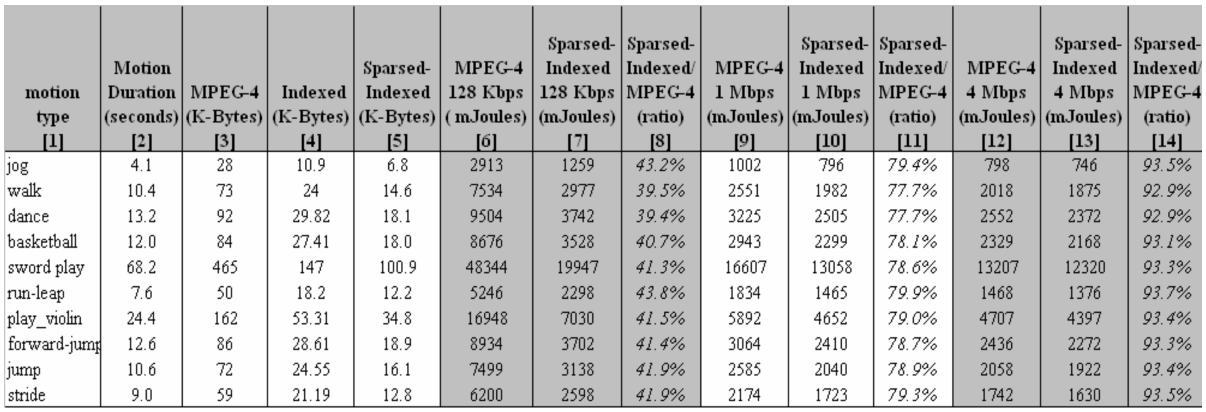
Sparse-indexing is partially compatible with existing MPEG-4 quantization and grouping techniques [Capin, 1999b]. The quantization portion of *Sparse-indexing* has several advantages over MPEG-4 quantization. The former uses a byte to represent each of the derived indices, whereas MPEG-4 quantization may require indices up to 4 bytes long, depending on the precision [Capin, 1999b]. MPEG-4 spatial grouping groups together joint types, in order to encode only certain portions of the joints. A similar approach can be easily incorporated in *Sparse-indexing*; i.e., for the 62 degrees of freedom avatar, a 62 bit mask (rounded up to 8 bytes) can be used as a mask to tell the decoder which joints are encoded in the BAP data. Another popular technique, called *dead-reckoning* [Capin, 1997] can be used easily in conjunction with *Sparse-indexing*. In *joint-level dead reckoning*, joint angles of the avatar are the only information that is required. Since *Sparse-indexing* is another way of encoding the joint angles, dead-reckoning techniques require trivial modifications to be made compatible with *Sparse-indexing*.

In the next three subsections, various comparisons are made with MPEG-4. We have not used grouping for the MPEG-4 encoded file; i.e. the entire 62 degrees of freedom of the BAP data are completely encoded in each row/frame. Quantization, followed by arithmetic coding, is used for MPEG-4 encoding. To ensure a fair comparison, the *Sparse-indexing* does not group the data either.

2.3.4.2. Minimum BAP data file size obtained by Sparse-indexing compared to MPEG-4

Table 2.3.1 gives the minimum throughput **T** obtainable using *Sparse-indexing*, and the corresponding displacement error **DEF**. As expected, the throughput **T** is significantly less than that obtained by MPEG-4-based compression.

Table 2.3.1: Comparison of compression ratio obtained via sparse-indexing and power consumption in m-joules by the wireless network interface card (WNIC) for various motion examples. Column [3] gives the compression file size after MPEG-4 arithmetic coding based compression. Column [5] gives the compressed file size obtained after sparse-indexing. Columns [6] and [7] give the energy consumption in milli-Joules for the reception of streaming MPEG-4 and Sparse-indexing data for a 128 Kbps network. Column [8] gives the ratio of power consumption by the WNIC using the sparse-indexing, compared to MPEG-4. Columns [9] – [11] give comparisons for network bandwidth of 1 Mbps, and columns [12] – [14] for 4 Mbps. $\boldsymbol{E_S}$ = 177 mJ/s and $\boldsymbol{E_R}$ = 1425 mJ/s.



motion type [1]	Motion Duration (seconds) [2]	MPEG-4 (K-Bytes) [3]	Indexed (K-Bytes) [4]	Sparsed-Indexed (K-Bytes) [5]	MPEG-4 128 Kbps (mJoules) [6]	Sparsed-Indexed 128 Kbps (mJoules) [7]	Sparsed-Indexed/ MPEG-4 (ratio) [8]	MPEG-4 1 Mbps (mJoules) [9]	Sparsed-Indexed 1 Mbps (mJoules) [10]	Sparsed-Indexed/ MPEG-4 (ratio) [11]	MPEG-4 4 Mbps (mJoules) [12]	Sparsed-Indexed 4 Mbps (mJoules) [13]	Sparsed-Indexed/ MPEG-4 (ratio) [14]
jog	4.1	28	10.9	6.8	2913	1259	*43.2%*	1002	796	*79.4%*	798	746	*93.5%*
walk	10.4	73	24	14.6	7534	2977	*39.5%*	2551	1982	*77.7%*	2018	1875	*92.9%*
dance	13.2	92	29.82	18.1	9504	3742	*39.4%*	3225	2505	*77.7%*	2552	2372	*92.9%*
basketball	12.0	84	27.41	18.0	8676	3528	*40.7%*	2943	2299	*78.1%*	2329	2168	*93.1%*
sword play	68.2	465	147	100.9	48344	19947	*41.3%*	16607	13058	*78.6%*	13207	12320	*93.3%*
run-leap	7.6	50	18.2	12.2	5246	2298	*43.8%*	1834	1465	*79.9%*	1468	1376	*93.7%*
play_violin	24.4	162	53.31	34.8	16948	7030	*41.5%*	5892	4652	*79.0%*	4707	4397	*93.4%*
forward-jump	12.6	86	28.61	18.9	8934	3702	*41.4%*	3064	2410	*78.7%*	2436	2272	*93.3%*
jump	10.6	72	24.55	16.1	7499	3138	*41.9%*	2585	2040	*78.9%*	2058	1922	*93.4%*
stride	9.0	59	21.19	12.8	6200	2598	*41.9%*	2174	1723	*79.3%*	1742	1630	*93.5%*

2.3.4.3. Power consumption by Sparse-indexing compared to MPEG-4

The process of data reception, decoding and rendering for BAP based animation can be broken into four steps:

(1) The Wireless Network Interface Card (WNIC) periodically receives the encoded BAP data and stores it in the WNIC buffer

(2) The CPU periodically transfers the data from the WNIC buffer to MEMORY

(3) The CPU reads the encoded data from MEMORY, and stores the reconstructed floating point numbers corresponding to the various joint angles in MEMORY.

(4) The Graphics Processing Unit (GPU) renders avatar animation using reconstructed data.

Power consumption in each of the above four high level steps can be attributed to the CPU (we will ignore the power consumption by the MEMORY) and also by the three main buses; i.e. the buses connecting CPU and MEMORY (BUS_{cm}), MEMORY and WNIC (BUS_{mw}), and MEMORY and GPU (BUS_{mg}). Power is consumed for each bus clock cycle, and the number of bus clock cycles is directly proportional to the number of bytes transferred over the bus. Hence, in order to estimate power consumption by the buses, it is adequate to quantify the amount of data transferred.

The above four steps consume power as follows: In *step 1*, power is consumed by the WNIC while receiving data from the server. In *step 2*, power is consumed by the BUS_{mw} while transferring data from the WNIC buffer to the MEMORY (the power consumed by the CPU during this data transfer is considered relatively small and hence ignored). In *step 3*, power is consumed by both the CPU while decoding the data, and the BUS_{cm} while transferring data between the CPU and the MEMORY. In *step 4*, power is consumed data by both BUS_{mg} (while transferring data from MEMORY to GPU), and the GPU while rendering the avatar for each frame.

Power consumption by BUS_{mg} is the same for both the MPEG-4 coded BAP data (henceforth termed as M-data) and *Sparse-indexing* coded BAP data (henceforth termed as SI-data), since the amount of data upon decompression is the same in both cases. Decoding M-data requires a substantial number of CPU cycles, compared to decoding of SI-data, since decoding of M-data requires a pipeline of processes such as inverse-arithmetic coding, and inverse quantization, whereas decoding of SI-data entails only an access to the lookup table. Hence, the power consumption by BUS_{cm} is greater in the case of decoding of M-data compared to SI-data. Finally, the BUS_{mw} transfers less data in the case of SI-data compared to M-data. This is evident from the results presented in subsection 2.3.6.2, which shows that the amount of SI-data generated is much less compared to the amount of M-data generated, in comparable settings.

Power consumption by the GPU in order to render the frames is same for both M-data and SI-data, since after decompression, the data required to animate the actual avatar model, are the same for both M-data and SI-data. The GPU is a major source of power consumption, along with data decoding and the bus data transfer. The significance of reducing power consumption for data reception, data transfer and data decoding, increases when simpler graphics models are used, such as in mobile devices [Chandra, 2003].

In order to ensure a fair comparison of power consumption during decompression of M-data and SI-data, we encode the M-data and SI-data to yield identical data file sizes. To achieve this, we first generate the M-data using MPEG-4 quantization and arithmetic coding (without grouping). We then compute the *minimum-compression-ratio f* of the obtained M-data file size to the original motion file size, and use this *minimum-compression-ratio f* in **equation (2.3.7)** to compute an upper bound on the network throughput requirement (file size) for the motion example. Next, we exhaustively enumerate all the possible values of **FDF** and n_d which satisfy the constraint in **equation (2.3.10)** (note that b = 8 and K = 0.1 have been fixed previously). We select the parameter values **FDF** and n_d which yield the minimum reconstruction error (**DEF**). The final parameters for the indexed motion file are chosen to be these parameter values. The results of the experiment are presented in **Table 2.3.2**. As evident from the table, decoding of the M-data requires a significant number of CPU cycles. On the other hand, a simple table lookup is required for decoding of the SI-data to obtain the actual data for joint angles, which, logically, entails fewer CPU cycles.

Finally, an analytical comparison of power consumption for receiving M-data and

Table 2.3.2: Comparison of MPEG-4 BAP compression vs *Sparse-indexing* (b = 8 and K = 0.1). The file sizes are in Kbytes. To compute the CPU cycles needed for decoding, we have used a 2.2 GHz Celeron CPU with 128 KB L2 cache and 512 MB RAM.

Motion Type	Org size	MPEG-4	G-cycles decoding MPEG-4	Comp. Ratio f	Sparse-indexing	DEF
Jog	33	28	0.65	85%	18.99	1.49
Run Leap	61	50	0.93	82%	28.55	0.45
Stride	72	59	1.23	82%	29.67	1.97
Walk	83	73	1.34	88%	45.91	0.17
Jump	85	72	1.35	85%	41.95	0.25
Basketball	96	84	1.44	88%	53.84	0.42
Forward jump	101	86	1.56	86%	55.71	0.34
Dance	106	92	1.68	87%	58.93	14.09
Play violin	195	162	2.89	83%	104.81	1.23
Sword play	545	465	8.49	85%	315.58	2.49

SI-data at the WNIC is made. For a motion of time duration T, data size S and given available bandwidth B, the energy used by the WNIC is given by

$$E_{network} = E_R \bullet S/B + E_S \bullet (T - S/B) \qquad (2.3.11)$$

E_R is the energy used by the WNIC during data reception and E_S is the energy used by the WNIC when it is sleeping and not receiving data. Using **equation (2.3.11)**, we compute the WNIC energy utilization for reception of M-data and SI-data (**Table 2.3.1**). We have used energy usage data from [Stemm, 1996] [Havinga, 2000] to obtain the energy usage for data reception in m-joules. Again, reception of SI-data resulted in significantly less energy consumption by the WNIC when compared to reception of M-data.

We conclude that BAP data compressed using *Sparse-indexing* leads to less power consumption for decompression, and much smaller compressed file sizes, compared to MPEG-4 compressed BAP data.

2.3.5. Conclusions and Future Work

A novel *Sparse-indexing* technique to compress the BAP data used for MPEG-4 compliant character animation has been reported. The proposed *Sparse-indexing* method leads to reduction in both, the throughput requirement for networked applications requiring motion data exchange, and client power consumption for data reception and data decompression. The resulting quality of the reconstructed motion is improved considerably by intelligent exploitation of the hierarchical structure of the skeletal avatar model during the process of creation of optimal lookup tables for reconstruction of the quantized motion. The quality and throughput requirements of the motion data are controlled via four quality control parameters. We have proposed a simple systematic search procedure to obtain the optimum combination of these parameters depending on the required compression ratio.

A limitation of the proposed technique is that the optimal values of two of the parameters, **FDF** and n_d, are obtained via exhaustive search, and values for the sparseness coefficient K and indexing-bits b are obtained via empirical observations. It may be possible to obtain the optimal values for these parameters more efficiently. Another drawback with any animation research is that there is no perfect quantitative measure for the quality of the reconstructed motion. Finally, the intelligent use of the hierarchical structure of the model yields good results for full body motions of the avatar; for small delicate motions such as movement of the fingers, or for facial animation, the proposed technique offers considerable scope for future improvement.

2.4. Weighted PCA

As seen in previous sections, MoCap data is typically represented as a matrix of dimensions $n \times m$, where n is the number of frames of the captured motion, and m is the number of degrees of freedom which can be independently manipulated. The number of columns m is referred to as the dimensionality of the motion data. MoCap data in this form is widely used for motion editing, and creation of new motions from a database of MoCap data [Brand, 2000], [Glardon, 2004], [Jenkins, 2002], [Lim, 2001], [Safonova, 2000]. A common preprocessing step is to alternatively project the MoCap data onto an appropriately determined k-dimensional sub-space, where $k < m$. This projection would result in a lower-dimensional representation of the motion data, making it easier to analyze, manipulate and edit the motion data. Hence, dimensionality reduction is a fundamental step in most applications of MoCap data.

A popular method for dimensionality reduction (DR) of MoCap data is Principal Component Analysis (PCA) [Bowden, 2000], [Jenkins, 2002]. PCA attempts to derive an efficient low-dimensional representation of the MoCap data by first, determining a set of mutually orthonormal axes which maximally de-correlate the input data, and second, by projecting the input data onto a lower dimensional subspace spanned by a judiciously chosen subset of these orthonormal axes. The projections of the input data on the above subset of the orthonormal axes are termed as the principal components (PCs) of the input data [Alexa, 2000]. However, a major drawback of the standard PCA technique is that the reconstructed motion from the lower-dimensional representation invariably exhibits visual distortion, irrespective of the error metric used. To reduce the distortion error, researchers have used various ad-hoc schemes for assigning weights to the columns of the original MoCap data matrix (termed as axis stretching) to improve the visual quality of the reconstructed motion [Gleicher, 1998], [Grochow, 2000]. These axis-stretching methods use various ad-hoc optimization techniques to determine a locally optimal solution (i.e. set of weights). However, none of the axis stretching methods explicitly exploit the hierarchical structural information derived from the human skeletal model. The use of PCA as a compression methodology, or as a pipeline component in the general scheme of MoCap compression, has been made explicit in the recent chapter by Arikan [Arikan, 2006]. In Arikan's approach, short clips of motion are first approximated by Bezier curves and compressed using standard PCA. Since the proposed WPCA is an improvement over standard PCA, it represents an important and significant step towards MoCap data compression in general.

Our empirical observations have shown that when a node displacement-based metric is used to measure the reconstruction error, lossy PCA, obtained via omission of eigenvectors corresponding to small eigenvalues from the basis set, does not necessarily yield optimal rate-distortion performance. A node displacement metric essentially considers the error metric as the amount of relative displacement that the nodes or bone joints of the skeletal model have undergone upon reconstruction from the reduced dimensional representation of the MoCap data matrix. In this article, we first propose a simple weighing scheme, which exploits the hierarchical structural information from the skeletal model to compute these weights. Although these weights result in improved rate-distortion performance, they are not necessarily optimal. To resolve this issue, we propose a Genetic Algorithm (GA) to efficiently learn the optimal weights. The initial heuristic weighing scheme is used to create the initial population for the GA. Results show that, for a given dimensionality of representation, the reconstruction error is significantly less for the proposed WPCA when compared to standard PCA. GAs are known to give near-globally optimal solutions, thus enabling the determination of a near-optimal set of weights which yield a near-global minimum of the reconstruction error.

2.4.1. Principal Component Analysis (PCA)

PCA attempts to efficiently represent the weighted data in $\boldsymbol{\delta}$ by first determining a set of mutually orthonormal axes, which maximally de-correlate the data. The input data is then projected on a lower-dimensional subspace spanned by a judiciously selected subset of the mutually orthonormal axes termed as the principal components. Let the covariance matrix of $\boldsymbol{\delta}$ be $\boldsymbol{\Sigma}$ and the eigenvalues and the eigenvectors of $\boldsymbol{\Sigma}$ be λ_i and $\boldsymbol{\Lambda}_i$ respectively, where $1 \leq i \leq m$. We can now write $\boldsymbol{\delta}$ as

$$\boldsymbol{\delta} = \mathbf{USV}^T \tag{2.4.1}$$

where $\mathbf{U}$ is an n x m principle component (PC) matrix. Each of the m columns of $\mathbf{U}$ represents a single PC curve. $\mathbf{S}$ is an m x m diagonal matrix, with the diagonal containing the eigenvalues λ_i, computed above, arranged in a non-increasing order, from top to bottom. The columns of matrix $\mathbf{V}$ are the eigenvectors computed above, arranged in the same order as the corresponding eigenvalues in $\mathbf{S}$, i.e., the column $\boldsymbol{\Lambda}_j = (\Lambda_{1j}, \Lambda_{2j}, \ldots, \Lambda_{mj})^T$ of the matrix $\mathbf{V}$ denotes the jth eigenvector corresponding to eigenvalue λ_j.

The desired dimensionality reduction is achieved by retaining the first k of the m eigenvalues in **S**, and their corresponding columns in the **U** and $\mathbf{V}^T$ matrices, and discarding the rest, such that, for a given fraction f of motion data that is to be retained, the following equation holds:

$$\frac{\sum_{i=1}^{k} |\lambda_i|}{\sum_{i=1}^{m} |\lambda_i|} = f \qquad (2.4.2)$$

Each of the three truncated matrices **U**, **S** and $\mathbf{V}^T$ is represented by a $k < m$ dimensional matrix, which requires much less data than the original matrix $\boldsymbol{\delta}$. The dimensionality of the compressed motion representation is $k < m$. This procedure is termed as dimensionality reduction via PCA.

2.4.2. Weighted Principal Component Analysis (WPCA)

In the proposed WPCA technique, the matrix $\boldsymbol{\delta}$ is reduced to a lower-dimensional representation, by first normalizing the matrix, then multiplying each matrix column by a weight, and finally, performing PCA on the resulting matrix, as described in Section 2.4.3. Let the mean and standard deviation of the columns of $\boldsymbol{\delta}$ be $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, where μ_i and σ_i are the mean and standard deviation of the i^{th} column, respectively. We *normalize* the data by re-computing $\delta_{ij} \leftarrow (\boldsymbol{\delta}_{ij} - \mu_i)/\max(\sigma_i)$, $1 \le i \le n,\ 1 \le j \le m.$ The elements of each row of the normalized matrix $\boldsymbol{\delta}$ are then multiplied by weights w_j, $j = 1, 2...m.$ This process is termed as *axis stretching*, as all axes in the m-dimensional space to which the rows of $\boldsymbol{\delta}$ belong, are stretched along mutually orthogonal directions. The stretching is generally asymmetrical, with some axes being stretched more, and some less. The objective of this chapter is to determine an optimal set of weights w_i, $j = 1, .., m,$ which results in minimum reconstruction error. After performing PCA on the axis-stretched matrix $\boldsymbol{\delta}$, the motion data is recovered from the low-dimensional representation, and each column is multiplied by the inverse of the weights to recover the original motion data.

2.4.2.1. The Reconstruction Error Metric

The reconstruction error is essentially the degree of error perceived in the motion capture data after the data is reconstructed from the low dimensional representation. Consider the motion matrix **δ**, whose columns represent the various degrees of freedom of the human skeletal model. For each of the *m* joint positions representing the degrees of freedom in the human skeletal model, the reconstruction error at a specific joint position is defined as the sum of *Euclidean* distances between the joint position in the original motion matrix and the joint positions in the reconstructed motion matrix over all the motion frames. Let $\mathbf{C}_{nx3p}$ be the original coordinates of the joints of the human skeletal model corresponding to the original motion matrix, and $\mathbf{C'}_{nx3p}$ be the coordinates of the joints of the human skeletal model corresponding to the reconstructed motion matrix. We define the error function at a joint position, Δ:I → R as follows:

$$\Delta(i) = \sum_{j=1}^{n} \| \mathbf{P}_{ji} - \mathbf{P'}_{ji} \|, \; i = 1, 2, ..., m \qquad (2.4.3)$$

where $\mathbf{P}_{ji} = (C_{j,3i-2}, C_{j,3i-1}, C_{j,3i})$, $\mathbf{P'}_{ji} = (C'_{j,3i-2}, C'_{j,3i-1}, C'_{j,3i})$, $i = 1,2, .., m$, and $\| . \|$ is the conventional Euclidean norm defined as$\|(x, y, z)\| = \sqrt{(x^2 + y^2 + z^2)}$. The error metric Δ(i) represents the error at each joint position. To get an estimate of the overall error for the motion sequence, the errors at each joint position (**equation (2.4.3))** are summed, and divided by the number of frames *n*, in the motion sequence (the number of rows in the motion matrix) to get the average error per frame Є as follows.

$$\text{Є} = \frac{\sum_{i=1}^{m} \Delta(i)}{n} \qquad (2.4.4)$$

Thus, Є is the reconstruction error corresponding to the motion matrix after performing PCA.

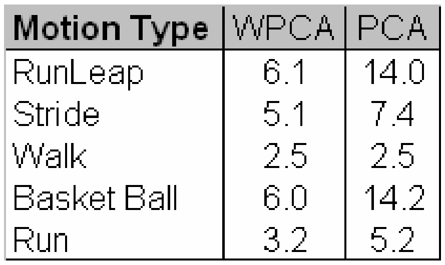
2.4.2.2. Heuristic Weight Assignment

Once the error metric is defined, it is now required to determine an appropriate set of weights which can further reduce the error Є in **equation (2.4.4)**. We first determine a heuristic weighing scheme which exploits the hierarchical structural encoding of the human model. The human model has a hierarchical representation, as shown in **Figure 2.2.3**. An important observation is that the manner in, and extent to, which the error at a joint position is propagated through the human body depends largely on the position of the joint in the skeletal hierarchy. We have exploited this fact to compute the heuristic weights. Each node representing a joint in the skeletal hierarchy has an associated level value, which can be interpreted as the tree depth of the node from the root. Consequently, each column i of the matrix **δ** has a level value l_i associated with it (**Figure 2.2.3**). We hypothesize that columns (nodes) with a higher level value should be allowed a lower contribution to the weighted PCA, (and vice versa) since the overall displacement error caused by a joint is inversely proportional to the level value of its corresponding node in the skeletal human model. Such a weight assignment scheme would be expected to limit the overall error. Thus, we compute weight w_i corresponding to column i as:

$$w_i = K\frac{r}{l_i} \qquad (2.4.5)$$

where K is a constant scaling factor and r is a real number between 0 and 1. We use a predetermined empirical value $r = 1$ in the case of the heuristically determined weights. Later, we assign a random value in the range [0, 1] to r when determining the initial population for the GA.

Table 2.4.1. Comparison of Reconstruction error for WPCA and standard PCA.



Motion Type	WPCA	PCA
RunLeap	6.1	14.0
Stride	5.1	7.4
Walk	2.5	2.5
Basket Ball	6.0	14.2
Run	3.2	5.2

We have tested impact of the heuristically chosen weights on some typical motion data. **Table 2.4.1** shows that the reconstructed error ϵ is significantly lower for most motion types when the MoCap matrix is pre-multiplied by the heuristically determined weights before performing PCA. In the next section, we describe a GA-based algorithm to determine the optimal weights. As is evident from the results, the reconstruction error ϵ obtained using the near-optimal weights determined by the GA is significantly lower compared to the reconstruction error obtained by using standard PCA.

2.4.3. Determining the near-optimal weights w_i using a Genetic Algorithm

In this section, we demonstrate how a near-optimal weight vector $\mathbf{w} = <w_1, w_2 \dots w_m>$ can be determined using a Genetic Algorithm (GA) [Goldberg, 1989]. The GA is particularly suited for this problem since the solution space is high-dimensional and contains several local optima. In addition, the objective function depends on the chosen motion example, and as such, does not have a functional representation. As a result, conventional derivative-based optimization algorithms cannot be readily used.

2.4.3.1. Representation of a solution

A potential solution to the axis stretching or weight determination problem is represented by an m-dimensional vector $\mathbf{w} = <w_1, w_2 \dots w_m>$. The elements of each row of the motion matrix $\boldsymbol{\delta}$ are multiplied by the weight vector $\mathbf{w}$ for the purpose of axis stretching. The optimal set of weights is expected to result in minimum reconstruction error. In this chapter, we have used a *steady state* model for the GA. The population size for the GA is varied between 10 and 40. The dimensionality (the number of columns) of the original motion matrix $\boldsymbol{\delta}$ is 62; i.e. $m = 62$.

The fitness function value for a given weight vector is given by:

$$F(\mathbf{w}) = C/(1 + \epsilon) \qquad (2.4.6)$$

where C is a constant, and ϵ is obtained from **equation (2.4.4)** computed after reconstruction of the motion from the weighted PCA, weighted using the corresponding set of weights. The goal therefore is to determine the optimal weight vector $\mathbf{w} = <w_1, w_2 \ldots w_m>$ such that the fitness function F(**w**) is maximized.

2.4.3.2. Initialization Strategy

The initialization of the weights is done by using **equation (2.4.5)** as in the case of heuristic weight assignment. The only difference here is that the parameter r is assigned a random value between 0 and 1. Since it is possible that standard PCA may well prove to be more optimal than the WPCA, a special member, all of whose weights are 1, is inserted into the population.

2.4.3.3. Selection Strategy

We have used the roulette wheel procedure [Goldberg, 1989] to select a subset of the potential solutions for the purpose of reproduction. Hence, the selection probability of a member in the population is given by the ratio of its fitness to the sum of fitness values of all the members in the population. This is a good choice for a selection operator in our case, as the fitness values of the various members of the population are comparable.

2.4.3.4. Crossover Operators

We have investigated three different types of crossover (X-over) operators; single point crossover, random crossover and arithmetic crossover. For the point crossover, we create the offspring by randomly selecting a crossover point, such that weights to the left of that point come from one parent, and the weights to the right come from the other parent. For random crossover, each weight value in the offspring is chosen randomly from either of the parents. Arithmetic crossover is performed by computing each weight of the offspring as the arithmetic mean of the corresponding weights of the two parents.

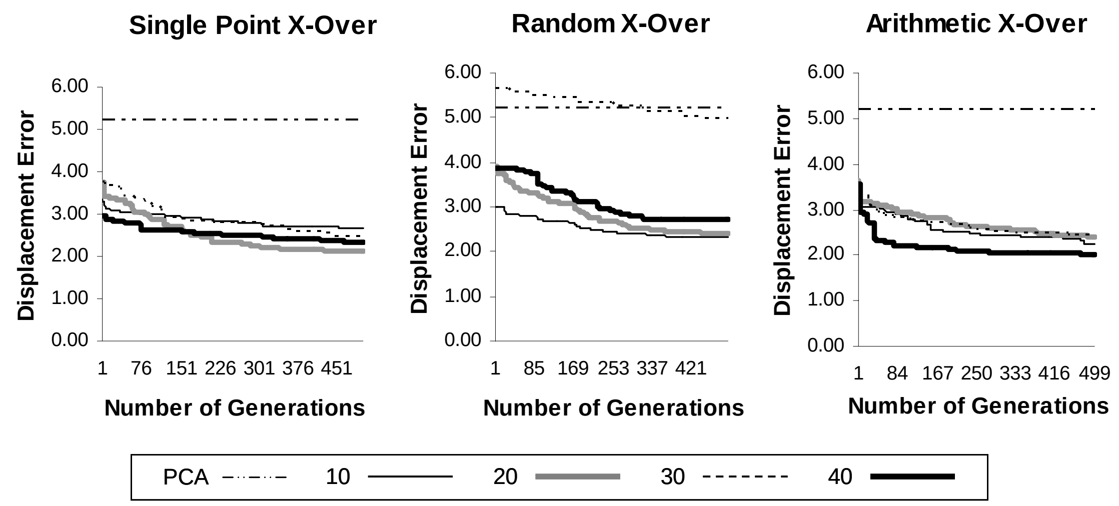


Figure 2.4.1: Behavior of Mean Reconstruction Error for Weighted PCA over successive generations for different population sizes (10, 20, 30, and 40), different X-Over operators (Single Point, Random and Arithmetic) for a *running* example (136 Frames). The mean reconstruction error for standard PCA without axis stretching is 5.2.

2.4.3.5. Mutation Operator

We have implemented a uniform mutation operator by changing a single component w_i in the weight vector of the offspring. This is effectively accomplished by selecting a random number i between 1 and m, and using **equation (2.4.5)** to re-compute w_i.

2.4.3.6. Replacement Strategy

We have implemented a simple replacement policy by removing the worst member from the population after the new offspring is added to the population. Note that the worst member is defined by the weight vector that results in maximum reconstruction error Є (**equation** (**2.4.4**)) after having performed the weighted PCA procedure.

2.4.4. Analysis

For the purpose of analysis, we have considered the popular motion example of *running*, which is commonly used in many computer games, war simulations and other animation

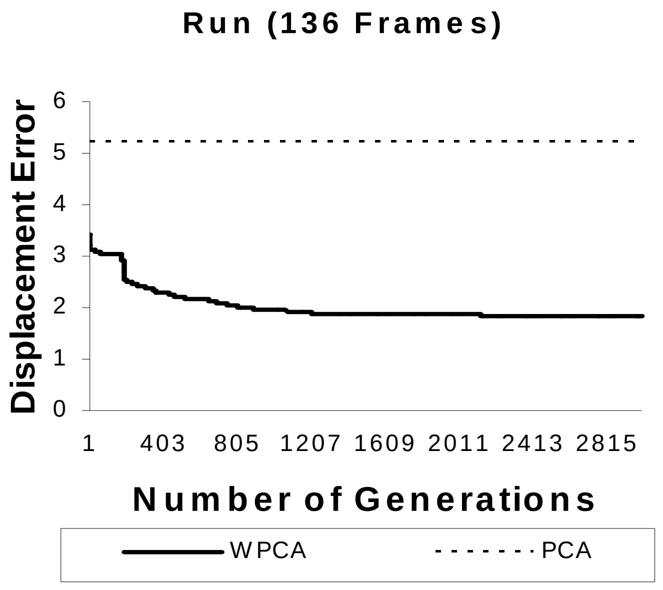


Figure 2.4.2 : Convergence of reconstruction error for the *running* example over 4000 generations. The GA is seen to converge in 2000 generations. The reconstruction error is almost a third of that obtained using standard PCA (dotted line)

applications. We also provide examples of various other types of motions that are commonly encountered in computer animation. The *running* motion example has 136 frames of motion, sampled at 33 frames per second. Furthermore, we require that 90% of the information in the original data needs to be preserved using PCA (equivalently, f = 0.9 in **equation (2.4.2)**). Empirical results show that a choice of f = 0.9 results in reconstructed motion of acceptable visual quality. The original motion data is 62-dimensional, whereas standard PCA results in a 14-dimensional representation. In order to ensure a fair comparison, the number of reduced dimensions in the case of WPCA is kept same as that in the case of PCA obtained by retaining 90% of the information content in the input data. A plot of the behavior of the Mean Reconstruction Error for the weighted PCA (WPCA) with a 14-dimensional representation of the original 62-dimensional motion vector, over 500 successive generations for different population sizes

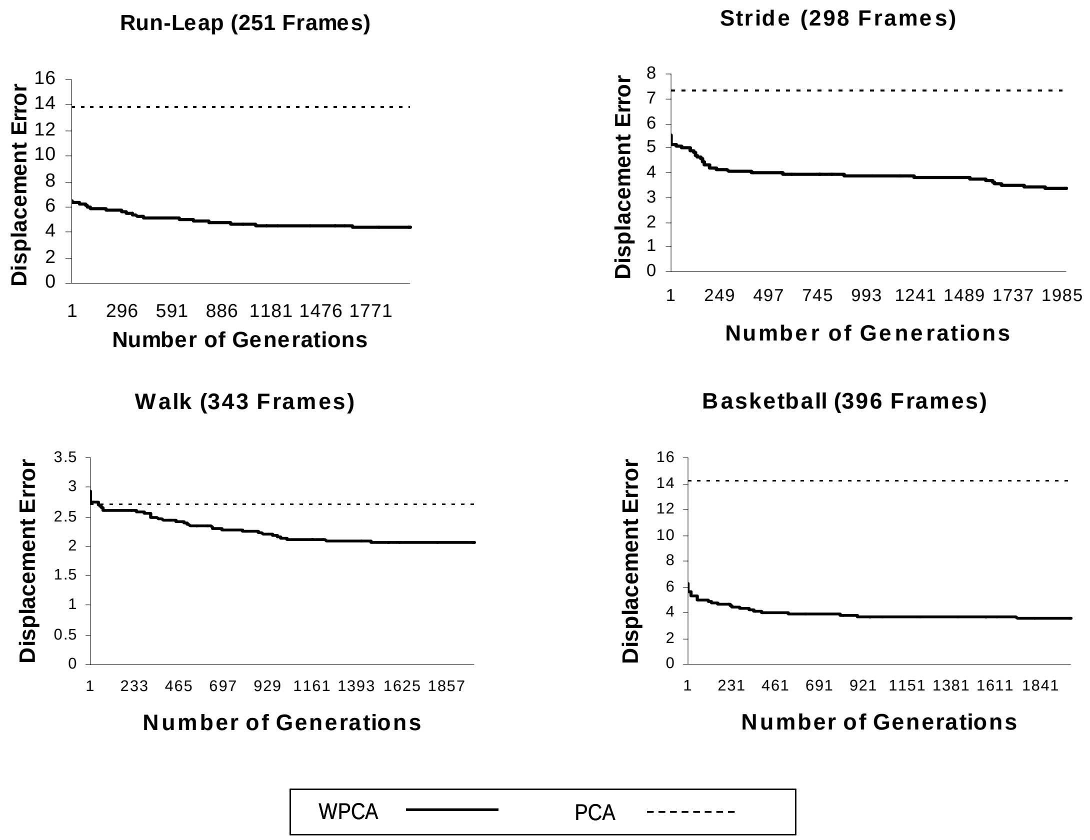


Figure 2.4.3: Displacement errors obtained for 2000 generations of the steady state GA for four motion examples. The dotted line represents the displacement error for standard PCA with 90% of the information in the input data retained. The solid line represents the displacement error for the weighted PCA, where the weights are determined using a steady state GA.

(10, 20, 30, and 40), and different crossover operators (single point, random and arithmetic) for the *running* example is given in **Figure 2.4.1**.

One immediate observation from the plot is that the initial population itself has a much lower reconstruction error than that obtained by standard PCA. Surprising as this may seem, note that we have intelligently exploited the hierarchical structure of the human skeletal model to generate the initial population. The fact that this is tantamount to a smart way of initial axis stretching is apparent from this observation. It now remains to use a good combination of the initial population and the selection, crossover and mutation operators to rapidly converge to a good set of weights. A brief study of the three plots in

Figure 2.4.1 reveals that that the GA with a population size of 40, and with the arithmetic crossover as the crossover operator of choice is seen to converge faster for the *running* example, based on the first 500 generations that are plotted. The GA with a population size of 10 exhibits the weakest performance. The GA with a population size of 20 is, in general, good for all the three crossover operators, and is also computationally less intensive. However, for motion examples of short duration, the GA generally takes less time to converge; hence, using a population size of 40 appears to be the proper choice. The convergence plot of the reconstruction error for the *running* example over a span of 3000 generations is shown in **Figure 2.4.2**. The GA is seen to converge in around 1250 generations.

Using the best combinations of GA parameters discussed above (population size = 40 and arithmetic crossover), we have tested the weighted PCA method for various motion examples. The PCA is performed such that reconstruction preserves 90% of the information in the original data. The WPCA is performed by retaining the same number of reduced dimensions as obtained via standard PCA, in order to ensure a fair comparison between PCA and WPCA. The results are shown in **Figure 2.4.3**.

A natural concern is whether this technique works universally for all types of motions. The proposed WPCA technique appears to be especially good for regular motions such as running, walking or striding, that are typical of regular human activity, or for slightly more complex motions such as run and leap, or basketball dribbling. However, this method may not perform well in situations where PCA itself performs badly, such ill-correlated motions resulting from arbitrary human actions. Another concern is that the proposed WPCA may result in worse reconstruction errors compared to PCA. To address this issue, one of the members of the initial population is chosen such that all of its weights are unity. If standard PCA (i.e. WPCA with unit weights) does indeed result in the best solution, then in the version of the GA algorithm presented in this chapter, the unit weight vector will always survive in the population to result in the least reconstruction error. This ensures that WPCA is at least as good as standard PCA, if not better. However, experimental results have shown that the proposed WPCA technique outperforms the standard PCA technique in terms of overall reconstruction error in all of the animation examples considered in this chapter.

Some videos comparing the results obtained via PCA and WPCA are presented at the web-address given in [Project-WebPage]. The very low-dimensional representation of the motions obtained using standard PCA show marked distortion in the motion quality, and are visually unacceptable. The motions corresponding to the same number of (reduced) dimensions, but obtained using WPCA instead, show marked improvement in quality, and are, in fact, acceptable.

Finally, with regard to the run-time for determining the weights, the GA-based optimal weight determination cannot be accomplished in real time. The run-time of the GA depends primarily on the size of the motion data and the chosen size of the GA population. Our typical running times were around 3 minutes for a 15 second motion sequence on a 2.2 GHz Intel processor with 512 KB L2 cache and 512 MB RAM. The heuristic determination of weights via exploitation of the hierarchical human skeletal model, however, can be computed easily in real time. The GA-based scheme is thus better suited for determination of the optimal weights for off-line compression of the MoCap data whereas the heuristic scheme could be potentially used for on-line compression of the MoCap data if some degree of sub-optimality is tolerable.

2.4.5. Conclusion and Future Works

In this chapter, we propose a weighted PCA (WPCA) technique as an improvement over the standard PCA technique commonly used for low-dimensional lossy representation of Motion Capture Data. The proposed WPCA scheme weighs the columns of the motion matrices prior to using standard PCA. We describe two schemes for weight determination. The first scheme is heuristic, and is based on exploitation of the hierarchical model of the skeletal humanoid. The second scheme, based on the Genetic Algorithm, yields near-optimal weights that result in minimum overall reconstruction error (i.e., the sum of node displacement errors). The heuristically determined weights are computationally efficient, and are well suited for quick computation. The determination of near-optimal weights using the GA involves computation which, typically, cannot be performed in real time, but which yields significantly lower reconstruction error.

We believe that this work will generate a fair amount of interesting discussion and open avenues for future research. First, since GAs are computationally expensive, direct gradient-based methods or other more sophisticated optimization methods may be used in the future to determine the optimal weights. Second, the current visual quality metric is based on the sum of node joint displacement errors in the human skeletal model after motion reconstruction. Although this is a reasonable metric, there exist many other metrics for visual quality assessment, which need to be tested. Finally, the basic GA-based scheme used in this article can be further improved to increase efficiency and speed of convergence.

2.4.6 Acknowledgement

The motion capture data used in this project was obtained from mocap.cs.cmu.edu. The database was created with funding from NSF EIA-0196217.

CHAPTER 3

MULTI-RESOLUTION LAYERED VIDEO ENCODING

3.1. Introduction

Typically, images and video as encoded in a single spatial resolution. The measure of how closely lines can be resolved in an image is called spatial resolution. It depends on properties of the system creating the image, not just the pixel resolution in pixels per inch (ppi). For practical purposes the clarity of the image is decided by its spatial resolution, not the number of pixels in an image. Thus, typically images and videos frames have uniform clarity throughout the image.

A fundamental property with low clarity image is that, with proper transcoding, low resolution/clarity image leads to more compact representation of the image or video. The theme of this chapter is to achieve compactness in order to display multimedia data in mobile devices which require compact multimedia representation to preserve power. However, overall low clarity of the multimedia content is not generally desirable.

Apropos, multi-resolution image/video encoding is essentially the technique of imposing good clarity in “visually important” regions of the image, and low clarity in so called unimportant regions. This representation results in a more compact representation compared to the original image/video with uniform clarity, yet the user is more satisfied than having to view a uniform low clarity image/video.

In this chapter, various novel techniques to create multi-resolution images and videos have been discussed and explored. Further, methods to create layered representation of multi-resolution videos has been reported, which enables the viewer to view video quality with clarity ranging from the original, best quality, to multi-resolution videos with general low clarity, but with good clarity in selected regions.

3.2. MMR: mask based multi-resolution images and videos

A multi-resolution (MR) image representation [Wilson, 1990] is useful for image encoding at multiple bit-rates. MR video, being a sequence of images, uses MR images to encode video at variable bit rates [Finkelstein, 1996]. MR images also have other applications in content-based image retrieval [Jacobs, 1995], medical imaging [Liu, 2001], storing and manipulating remotely sensed satellite data [Benz, 2004] among others. MR frames of an MR video yield better quality video compared to uniform resolution (UR) frames of the same encoded size, since each MR frame is encoded at multiple resolutions (within the same image), with *average* resolution comparable to that of the UR image, yet with better resulting image quality.

The standard algorithms used to create MR images exploit spatial redundancy within the image to group certain pixels with similar color values to form pixel blocks of uniform color. However, this approach is not sensitive to important spatial image features such as edges, object shapes, and other semantically or visually important objects such as human faces. As a result, MR often leads to sub-optimal image/video representations, where interesting regions of the image/video are represented at low resolution, and/or uninteresting regions of the image/video are represented at high resolution. In addition, since the definition of *interesting regions* of an image/video is subjective and application dependent, the generalized color difference-based schemes used by existing MR algorithms are not adequate to obtain MR representations of images/videos that are best suited for a specific application.

In this chapter, a novel mask-based MR (MMR) image/video representation technique is proposed. Rather than strict color based segmentation, a mask is used instead to define regions of interest within the image, where the best resolution is desired. Such a mask, when used appropriately, can be used to encode the desired features of the image at the highest resolution, and encode the rest of the image at the lowest resolution. The introduction of such a mask results in three significant benefits; (a) the quality of the multi-resolution image is completely controlled by the mask; (b) the mask-based multi-resolution image encoding algorithm is parametric in the mask-parameter space, instead of the original MR image space. This is desirable, since the mask is visually more intuitive to control; and finally (c) since the image contents and quality are now completely controlled by the mask, many existing algorithms for low-level feature extraction and semantic feature detection can be used to create the mask.

3.2.1. Quad-Tree for MR Images

In this section, we give an overview of the methods used for creating MR images. Note that multi-resolution (MR) videos [Finkelstein, 1996] use the same spatial encoding techniques used for multi-resolution (MR) images. Hence, the techniques for generation of MR images are automatically applicable to MR videos frames.

3.2.1.1. Creating quad-trees for multi-resolution images

Quad-trees are, by far, the most popular data structures to represent MR images [Wilson, 1990] [Samet, 1990]. Quad-tree decomposition is a simple technique for representing images at multiple resolutions. In this technique, the image is recursively divided into four equal-size square regions depending on the contents of the blocks. For example, a 2^nx2^n image is represented as a tree of depth n. The root of the tree represents the original image at resolution level zero, and the four equally-sized squares represent its children at resolution level one. Each node at every resolution level encodes its own color values (RGB); the parent node color is the mean of the colors of its child nodes. The details of the quad-tree decomposition algorithm can be found in [Wilson, 1990] and [Samet, 1990].

3.2.1.2. Color difference-based quad-tree pruning

In most standard MR algorithms, at each node of the quad-tree corresponding to an image, a decision is made as to whether to decompose the corresponding block into four equal-size squares or to halt the decomposition process. In order to arrive at a decision, the most widely used measure is a difference measure of color values [Samet, 1990]. At each node, the value of the difference measure is compared with a *threshold value*; if the absolute difference is smaller than the *threshold value*, the recursive decomposition procedure at that node is halted. Otherwise, the node is further decomposed into four equal-size squares. The data structure for each node of the quad-tree can be represented by:

```
struct QuadNode
        ENUM type: NODE | LEAF
        QuadNode child[0, 1][0, 1]
        COLOR: RGBA
        REDUNDANCY: 0 or 1
end struct
```

If the node type is LEAF, then each child is NULL, and the COLOR is the color of the corresponding pixel. Otherwise, the node type is NODE, and each child is a pointer to the four equal sections of the block. The COLOR attribute in this case is the average color of the four children. The attribute REDUNDANCY, which is a binary number, is used to describe the degree of redundancy. REDUNDANCY = 1 in a node means that this node (and its children) can be eliminated, whereas REDUNDANCY = 0 means that this node (and consequently its children) cannot be eliminated at all.

3.2.2. Using Masks For Quad-Tree Pruning

The quad-tree pruning mask is defined as follows:

$$\text{Mask-Bin}(i, j) = \begin{cases} 1 & \text{if that pixel should not change at all} \\ 0 & \text{otherwise} \end{cases}$$

Assuming that the quad-tree and the desired mask, *Mask-Bin*, exist, the quad-tree branches can be pruned using the given mask by the following recursive pseudo-code:

```
function Mask_Prune(QuadNode p) returns 0 or 1
    if p. type = LEAF then
        i, j ← pixel corresponding to this leaf
        return (1 - Mask-Bin(i, j))
    end if
    redundancy = 1 // assume this node is redundant
    for each i, j = 0 and 1
        flag = Mask_Prune(p.child[i][j])
        **if** flag = 0 **then** redundancy = 0
    **end for**
    p.REDUNDANCY = redundancy
    **return** redundancy
**end function**
```

The root of the quad-tree is passed to the Mask_Prune function. The function Mask_Prune results in the best resolution (by preserving the original pixels) in the smallest-size image blocks corresponding to non-zero mask values. It also results in the maximum number of low resolution blocks in the other regions of the image. In the next section, we discuss some plausible masks for MMR images and MMR videos.

3.2.3. Masks for MMR Images and Videos

In this section, we present three broad categories of masks, which can be used for most applications requiring MR images and videos. Note that the advantage of mask based multi-resolution (MMR) image is that the average resolution of the image is much lower than that of the original image. Hence, the MMR image when encoded using a standard encoding technique for quad-tree based image representation, results in much less data compared to the original image, thus proving to be an enhanced image/video compression method.

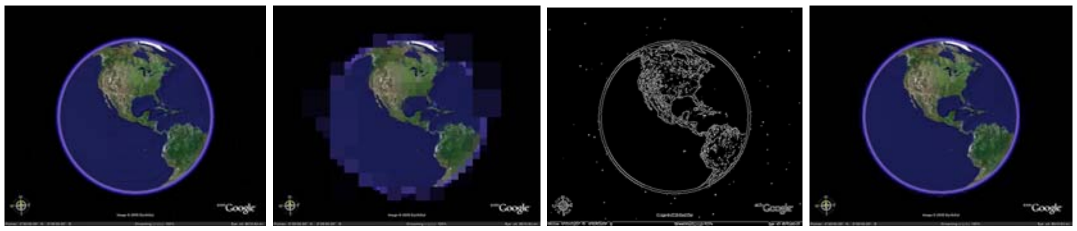


Figure 3.2.1. MR vs MMR for a google earth image. (a) The original image (b) Standard MR image; δ_{MR} = 76.1% (c) MMR *edge mask* (d) MMR image; δ_{MMR} = 78.8% i.e. δ_{MMR} and δ_{MR} are comparable; which means that the MMR image has approximately the same average resolution as that of MR image, but significantly better image quality.

3.2.3.1. Feature masks

Features masks are useful in preserving low level spatial image features such as edges, boundaries, texture etc. For example, edges can be preserved in MMR using *edge masks*. An interesting application of *edge masks* is in multi-resolution streaming video applications, such as *Google-Earth*, where the basic edge features are maintained while zooming in on a country or region. We used an *edge mask* on a *Google-Earth* image (**Figure 3.2.1**), obtained by using the standard Canny edge detector [Canny, 1986]. The value of the Canny edge detector parameter, σ, is obtained semi-empirically; a detailed discussion on this issue is beyond the scope of this chapter. Note that the MMR image quality is significantly superior compared to the MR image quality. Detailed quantitative analysis on the comparison of MMR and MR image qualities is given in section 3.2.5.

3.2.3.2. Object masks

Object masks exploit the basic human psychology of focusing on semantically important objects while ignoring other artifacts in an image/video. For example, in a video requiring attention to be drawn to human faces, a *face mask* can be used for each frame in the video stream. A *face mask* can be obtained using a face-detection algorithm [Rowley,

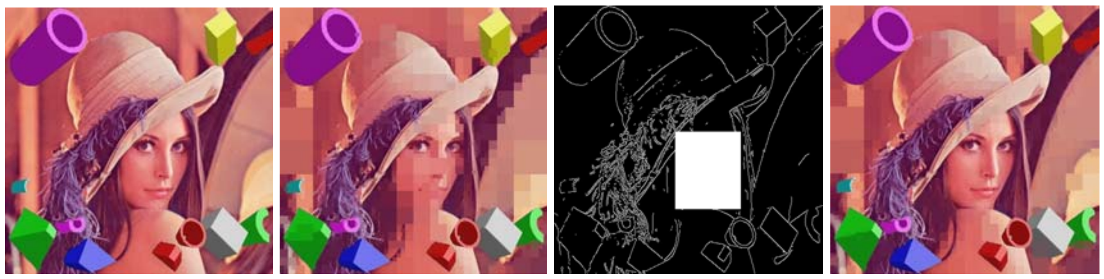


Figure 3.2.2. MR vs MMR for the Lena image with embedded 3D objects. (a) The original image (b) MR image; δ_{MR} = 72.4% (c) MMR combination of *edge mask* and *face object mask* (d) MMR image; δ_{MMR} = 78.4%.

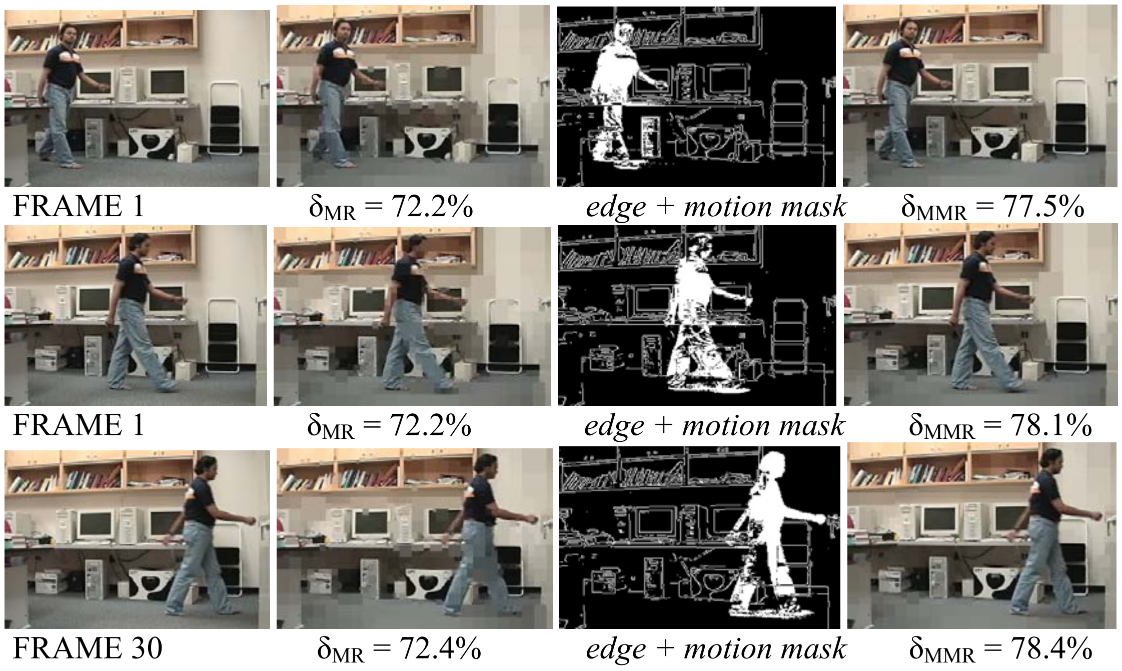


Figure 3.2.3. MMR on frames from a video sequence. (*Column 1*) The original frames (*Column 2*): Standard MR frames (*Column 3*): Mask for MMR: *edge + motion mask*; *motion mask* obtained by background separation (*Column 4*): MMR frames.

1998]. Other *object masks*, such as *shape masks*, emphasize certain shapes in an image, which can be useful for robotic applications. Such a mask may be obtained using pattern recognition algorithms [Jain, 2000]. **Figure 3.2.2** illustrates a combination of an *edge mask* and a *face mask*. Note that *visually interesting regions*, such as the face, hair, and the shapes of the 3D objects, are preserved at their original resolution, whereas other areas are displayed at the lowest local resolution possible.

3.2.3.3. Motion Masks

Motion masks can be used to render temporally changing objects in a video at higher resolution compared to the rest of the video. An example, using a combination of an *edge mask* and a *motion mask*, obtained by simple background subtraction, is illustrated in **Figure 3.2.3**. The MMR frames show a significant improvement in quality of the *interesting video regions* such as the moving object and background edges; these specific enhancements are absent in the general MR frames.

3.2.4. MMR VS MR

A comparison between MMR and MR involves two fundamental attributes; the visual quality obtained and the size in bytes of the encoded image/video. The image/video quality resulting from both MMR-based and MR-based rendering can only be assessed subjectively by the user, based on the requirements of a particular application. To have a numerical estimation of the image/video size obtained using MR or MMR, we have used a percentage metric δ such that:

$$\delta = 100 * n / N \qquad (3.2.1)$$

where n is the number of pixels in the multi-resolution image, obtained by treating each block as a single pixel, and N is the total number of pixels in the original image. To ensure a fair comparison between MR and MMR, we compute δ_{MR} and δ_{MMR} for MR and MMR respectively, and report the resulting image quality where δ_{MR} is comparable to

δ_{MMR}. **Figures 3.2.1**, **Figure 3.2.2** and **Figure 3.2.3** illustrate the difference in the image/video quality obtained by using MR and MMR, where δ_{MR} and δ_{MMR} are comparable. The MMR images/videos retain all the desired visually and semantically meaningful features of the images/video-frames. The MR images/videos, on the other hand, are not subjected to the same quality control, resulting in visual distortion in some critical regions.

3.2.5. Conclusions and Future Work

A mask-based multi-resolution (MMR) image/video representation technique has been discussed. The MMR representation uses a mask to control the quad-tree pruning for generation of multi-resolution (MR) quad-tree image/videos. The masks are defined such that a value of 1 indicates that the pixel is totally redundant (i.e. pixel color can be changed at will) whereas a value of 0 indicates that the pixel has to remain unaltered in the image. Three groups of masks i.e. *feature masks*, *object masks* and *motion masks* have been suggested. Examples and results have shown the considerable advantage of using MMR compared to MR in terms of image/video quality control.

Using masks provides enormous opportunity for future work. Various masks suitable for different types of images/videos can be proposed. Application-specific masks can be developed in future to aid in compression, representation and querying of images and videos.

3.3. FMOE-MR: Features, Motion and Object Enhanced Multi-Resolution Layered Video Encoding

Streaming video across the Internet has become one of the most important means of distributed information sharing. Since the Internet bandwidth availability is dynamic, it is essential to dynamically adapt the bit rate of streaming video, in order to ensure uninterrupted video streaming. Although H.264 (also called MPEG-4 *part 10*) is currently one of the most popular encoding standards for standard Video encoding, MPEG-4 Fine Grained Scalability (FGS) profile [ISO/IEC-visual, 2004] [Li, 2001] [Radha, 2001] is still one of the most popular standards to achieve adaptive video streaming. MPEG-4 FGS encodes the video into a Base Layer and an Enhancement Layer. The Base Layer bit rate is the minimum bit rate at which the video can be streamed. However, this bit rate may not be sufficient for low bandwidth networks which cannot support even the low bit rate of the Base Layer. Theoretically, the Base Layer may be encoded at even lower bit rates to allow streaming to these low bandwidth networks; however, this inevitably leads to a drastic reduction in video quality to the point that the visual information is almost useless. Hence, the rate distortion performance of standard MPEG-4 Base Layer encoding for FGS calls for improvement, in order to allow transmission at even lower bit rates at a reasonable video quality.

One way of improving the rate distortion performance of the MPEG-4 encoding scheme for the low bit rate Base Layer is by filtering out semantically and visually "*uninteresting*" information from the video frames. This can be achieved by re-encoding each frame as a multi-resolution frame, with the visually and semantically "*interesting*" information at high resolution, and the rest in low resolution. The multi-resolution scheme is implemented such that it fits within the scheme of the standard MPEG-4 compression pipeline, consisting of predictive encoding, quantization and variable length encoding of the DCT space of the video frames.

In this section, we have described and implemented a *mask*-based multi-resolution (MR) step in the standard MPEG-4 FGS Base Layer encoding pipeline, which can achieve acceptable video quality in visually important regions of the video at very low overall bit rates. Each frame of the FGS Base Layer video is re-encoded, using the proposed Features, Motion and Objects Encoded-Multi-Resolution (FMOE-MR) scheme, such that the regions defined by the *mask* are at high resolution, whereas the remaining regions in the frame are at low resolution. The multi-resolution scheme is implemented in a manner such that when the MPEG-4 video encoding pipeline converts the color-space of frames to their corresponding DCT space, the DCT coefficients require a very low number of bits for encoding. This leads to low bit rate Base Layer MPEG-4 video encoding schemes. We demonstrate unsupervised and semi-supervised methods to create *Features, Motion and Objects* (*FMO)-masks* based on the presence of features, motions and objects detected in the video sequence. The *FMO-Mask* essentially defines the importance of a pixel in the image by assigning it a weight that lies between 0 and 1. The *FMO-Mask* is obtained computationally via content analysis of the video sequences using appropriate computer vision algorithms. The combination of the *FMO-Mask* and Base Layer selective video enhancement using multi-resolution (MR) techniques is what we term as the proposed FMOE-MR scheme.

It must be noted that there exist technologies [Richardson, 2004] to selectively enhance the quality of spatial regions in the video frame while streaming within a constrained bandwidth. MPEG-4 selective enhancement [Van der Schaar, 2001] is employed in the Enhancement Layer of MPEG-4 FGS in order to stream higher quality video within selected image regions. However, MPEG-4 selective enhancement does not provide quality improvement for the Base Layer. Improving video quality of the Base Layer is essential, because for low network bandwidths since the Base Layer is often the only layer which can be streamed. In order to improve the quality of the Base Layer, MPEG-4 FGS uses adaptive quantization (FGS-AQ) [Van der Schaar, 2001]. FGS-AQ quantizes each 8×8 DCT block differently based on its relevance in improving the overall video quality.

The proposed FMOE-MR technique has several advantages over the existing MPEG-4 FGS-AQ based Base Layer encoding; (a) FMOE-MR results in significantly better rate distortion performance compared to FGS-AQ, by using a pixel-level multi-resolution video frame representation (b) FMOE-MR is transparent to the decoder; FGS-AQ, on the other hand, requires additional parameters and components for the decoder to decode each frame (c) FMOE-MR requires no additional changes to the existing MPEG-4 codecs, thus making the overall scheme very portable.

The rest of the section is organized as follows: In Section 3.3.2, an overview of the existing technologies for MPEG-4-based layered coding of video (FGS), and FGS-AQ, is provided. Section 3.3.3 describes the proposed FMOE-MR scheme in detail. Detailed quantitative and qualitative comparisons of the proposed FMOE-MR Base Layer video encoding scheme with the existing MPEG-4 FGS-AQ scheme are presented in Section 3.3.4. Finally, the conclusions and potential future work are presented in Section 3.3.5.

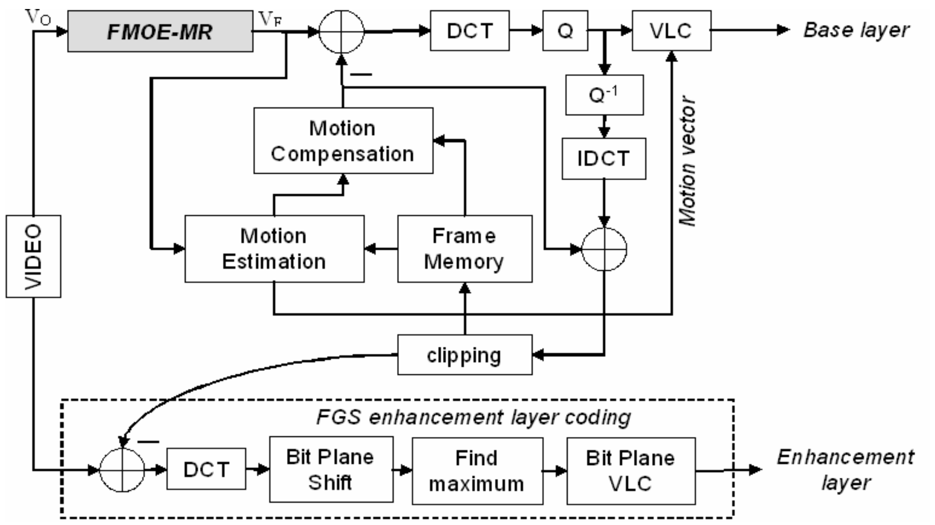


Figure 3.3.1: The MPEG-4 FGS scalable video encoding pipeline, with the proposed FMOE-MR step added. (grey box).

3.3.1. Background

MPEG-4 fine grained scalability profile, FGS, separates the video frames into two layers, which are referred to as the Base Layer and the Enhancement Layer (**Figure 3.3.1**, minus the shaded box). The Base Layer is encoded at the minimum bit rate available to the video streaming network. The Enhancement Layer is obtained by encoding the difference between the original DCT coefficients and the coarsely quantized Base Layer coefficients in a bit-plane manner [Radha, 2001] [Richardson, 2004]. The Enhancement Layer can be truncated at any bit position and can provide fine granularity of control of the reconstructed video quality which is proportional to the number of bits actually decoded.

The Base Layer, since it is encoded at the minimum bit rate, is often the most significant layer that can be streamed across to the client when the bandwidth drops down to a certain minimum value. Thus it is necessary that the Base Layer retain the highest quality in the semantically and visually important regions of the video for a specified bit rate. MPEG-4 uses adaptive quantization [Richardson, 2004] [Van der Schaar, 2001] in its Fine Grained Scalability Base Layer encoding (FGS-AQ) scheme to assign more bits to the DCT coefficients of the blocks that correspond to the desired regions that need to be enhanced. FGS-AQ is achieved via a quantization matrix that defines different quantization step sizes for the different transform coefficients within a block (prior to performing entropy coding on these coefficients). These adaptive quantization tools have been employed successfully in the MPEG-2 and MPEG-4 (base-layer) standards [Van der Schaar, 1999]. However, the aim of FGS-AQ is not to improve the rate distortion performance, but rather to improve the visual quality of the resulting video. As a result, the rate distortion performance of an FGS encoder that uses FGS-AQ may actually degrade due to the overhead entailed in the transmission of the FGS-AQ parameters.

One of the main aims of the proposed FMOE-MR technique is to actually obtain better rate distortion performance for the Base Layer encoding compared to the FGS-AQ technique. A crucial by-product of using the proposed FMOE-MR scheme is its transparency to the existing MPEG-4 codecs; thus not requiring any additional codecs for decoding multi-resolution video frames. In the next section, we describe the proposed FMOE-MR technique in detail.

3.3.2. Proposed approach: FMOE-MR

The proposed FMOE-MR scheme is based on the fundamental observation that applying a low pass filter in the color space of an image is equivalent to DCT coefficient truncation in the corresponding DCT space of the image [Geusebroek, 2001] [Gonzalez, 1992]. The FMOE-MR scheme is a two step process:

(i) *Creating the FMO-Mask*: Features, Motion and Objects (FMOs) are detected in the video sequence using state-of-the-art computer vision algorithms. A corresponding mask (FMO-Mask) is created to mark the regions corresponding to the presence of the FMOs. The mask has floating point values between (and inclusive of) 0 and 1, where 0 represents a completely uninteresting region and 1 represents a vital region for visual and semantic understanding of the image.

(ii) *Multi-Resolution (MR) re-encoding*: The original frame is re-encoded as a multi-resolution representation, guided by the FMO-mask such that regions corresponding to mask values near 1 are at higher resolution compared to regions corresponding to mask values near 0.

3.3.2.1. Creating the FMO-Mask

The FMO-Mask is essentially a combination of one or more of the following three individual masks; the *Feature*-Mask (F-Mask), *Motion*-Mask (M-Mask) and the *Object*-Mask (O-Mask). As discussed in Section 3.2.3.1, the masks can be the following:

Feature mask (F-mask)

The F-Mask captures the low-level spatial features of the video frame. Edges are one of the most important low-level features of an image (or video frame) because human perception tends to first detect edges for object recognition and general scene analysis.

Edges can be detected automatically in a given image. There are many ways to perform edge detection. However, the majority of different methods may be grouped into two broad categories: gradient-based and Laplacian-based. The gradient-based methods detect the edges by seeking the maximum and minimum in the first derivative of the image intensity (color) function. The Laplacian-based methods, on the other hand, search for zero crossings in the second derivative of the image intensity (color) function to find edges. We have used a gradient-based edge detection algorithm, known as the Canny edge detector [Canny, 1986], to find edges in a video frame.

Once the edges in the video frame are determined by using the Canny edge detector, the F-mask is created by assigning the value of 1 to regions in and around the edges, and the value of 0 elsewhere. Note that the mask is actually a weighting matrix, and as such, each pixel may be assigned any value between (and inclusive of), 0 and 1.

Motion mask (M-mask)

Motion within a video sequence constitutes a very important visual phenomenon. The human eye tends to follow the moving objects to note their activities. Therefore, in situations which demand reduction in quality of the video, the regions with motion in the video can be rendered at high resolution and the rest of the video at low resolution. Detection of motion in video sequences is summarized in two major steps:

(i) *Background Subtraction*: The background of a video sequence is either the stationary backdrop, or backdrops which change as a result of camera motion such as panning and translation. Background subtraction is required in order to extract foreground objects which are moving relative to the camera, or had been moving recently. Background subtraction is done typically by first creating background models [Luo, 2005a]. The video sequences are then compared with the background model to detect regions which are not part of background. These regions are classified as belonging to the foreground object.

(ii) *Foreground Object Tracking*: Once the foreground objects are detected, they are tracked over the video sequence. Note that a simpler modification would be to just detect the foreground in each frame, as the mask is dependent on the foreground objects only. However, this approach fails when the moving object stops temporarily; in such a case, just the foreground becomes a part of the background. Tracking, on the other hand, will still detect the still object as part of the foreground.

Motion tracking by itself is an extremely well-researched area, and a detailed discussion on the same is clearly beyond the scope of this chapter. We have implemented a novel tracking algorithm based on optical flow-based multi-scale elastic matching [Luo, 2005b]. The algorithm can detect and track multiple objects moving in a video sequence. The tracked objects constitute an important part of the M-mask.

Object mask (O-mask)

All the foreground objects in a video sequence may not be equally important visually or semantically. For example, in a video sequence containing a news reader, with a rotating background logo, the face of the news reader is more important than the moving logo (which too is a part of the foreground). In this case, the face is deemed an object of interest amongst the foreground regions.

Face recognition and tracking is typically done by extracting feature points that characterize a face, and tracking these feature points in the video sequence. A detailed description of the face recognition and tracking algorithm is again beyond the scope of this chapter. Faces in a video sequence can be detected using the algorithm described in [Zarit, 1999] and tracked using the algorithm described in [Luo, 2005b] to create an O-mask based on human faces automatically.

3.3.2.2 Mask Combinations

The three masks mentioned above are implemented such that each element of the mask value represents a weight in the range [0, 1]. The higher the weight, the more significant the corresponding pixel in the overall visual and semantic content of the image frames. A combination of these masks may be more appropriate for a particular application. The three masks, F-Mask, M-Mask and O-Mask, can be combined as follows:

(i) *Optimistic Combination*: An arithmetic MAX operation at each pixel would assign the maximum of the three mask values at each pixel location. This is an optimistic combination, as this would mean that a non-interesting pixel would not affect an interesting pixel.

(ii) *Pessimistic Combination*: An arithmetic MIN operation on each pixel would set the minimum mask of the three mask values at each pixel location. This is a pessimistic combination, as an uninteresting pixel would render the combined value to be that of the uninteresting pixel.

(iii)*Arithmetic Combination*: Each pixel of the combined FMO-mask is the arithmetic mean of the three mask values at each pixel location. This gives the maximum weight to pixels where all the masks values are 1, and the minimum weight at pixels where all the values are 0.

Since the M-Mask requires identification of objects in the video scene, one may argue that a natural method of encoding would be to use MPEG-4 object-based coding [Bertini, 2004][Wang, 2005]. However, MPEG-4 FGS does not support object based coding. In addition, object-based coding requires additional components at both the encoder and the decoder end. FMOE-MR, on the other hand, does not require any modifications either at the encoding side, or at the decoding side.

The visual performance of FMOE-MR can be improved by using a proper combination of the above mentioned masks. The combination depends on the application on hand. For example, in order to encode a surveillance video, combinations of motion mask (to render moving objects at good resolution) and face mask (to recognize faces of the people moving around) might be required. For generic movies, where contents are not known, a combination of motion mask and edge masks can be used, as these two features are generally the most robust recognition mechanisms for human perception. An example of a combination of all the three masks to create an FMO-Mask, using *optimistic combination*, is given in **Figure 3.3.2**. A detailed study of the effect of mask combinations is present in Section 3.3.4.4. It must be noted that the FMO-Mask is used only at the encoding side, and, as such, is transparent to the decoding side.

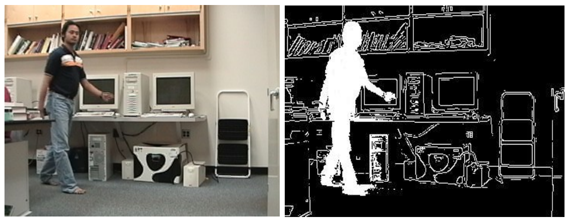


Figure 3.3.2: An *optimistic combination* of F-Mask, M-Mask and O-Mask to form the FMO-mask.

3.3.2.3. Multi-resolution (MR) Re-encoding using the FMO-Mask

Once the FMO-Mask is created, it remains now to re-encode the original frame in a multi-resolution manner, guided by the FMO-Mask. The goal of the MR step is to take the original video frame, V_O, re-encode it at multiple resolutions using the FMO-Mask, and produce the final video frame, V_F. The final video frame, V_F, is in turn fed to the MPEG-4 FGS Base Layer encoding pipeline (**Figure 3.3.1**) to obtain the Base Layer for FGS.

The simplest method of creating the MR video frame V_F is to render a weighted combination of two video frames at different resolutions. The original video frame, V_O, is used to render two video frames, V_H and V_L, such that V_H is a high resolution rendering and V_L is a low resolution rendering of the same video frame. We assume that a Gaussian filter [Davies, 1990], denoted by $G(\sigma)$, with parameter σ representing the standard deviation, is used as a representative low pass filter. V_L can be obtained by convolving V_O with a Gaussian filter $G(\sigma_L)$; similarly, V_H can be obtained by convolving V_O with a Gaussian filter $G(\sigma_H)$. Maintaining $\sigma_L > \sigma_H$ ensures that V_L is *smoother* than V_H; in other words, V_L is rendered at a lower resolution compared to V_H. In order to combine the video frames at the two resolutions, the mask weight matrix, **W** (matrix version of the FMO-Mask), is created which describes, in terms of normalized weights (between 0 and 1), the regions in a frame which need to be rendered at good resolution. An intermediate video frame, V_I, is created from the two video frames, V_H and V_L, and the weight matrix **W**, and is given by

$$V_I = (\mathbf{I} - \mathbf{W})V_L + \mathbf{W}V_H \qquad (3.3.1)$$

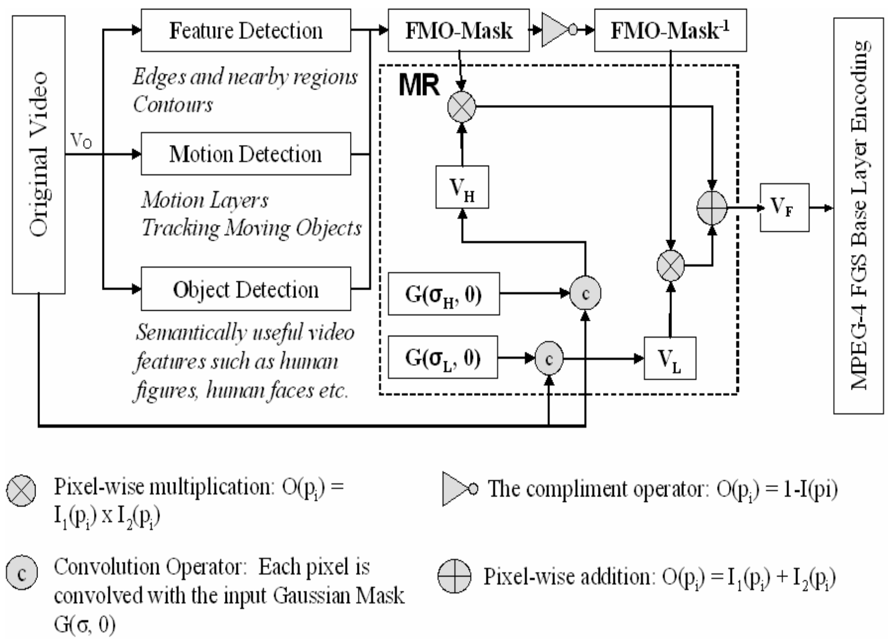

Original Video
V_O
Feature Detection
Edges and nearby regions
Contours
Motion Detection
Motion Layers
Tracking Moving Objects
Object Detection
Semantically useful video features such as human figures, human faces etc.
FMO-Mask
FMO-Mask^{-1}
MR
V_H
$G(\sigma_H, 0)$
$G(\sigma_L, 0)$
V_L
V_F
MPEG-4 FGS Base Layer Encoding
Pixel-wise multiplication: $O(p_i) = I_1(p_i) \times I_2(p_i)$
The compliment operator: $O(p_i) = 1\text{-}I(pi)$
Convolution Operator: Each pixel is convolved with the input Gaussian Mask $G(\sigma, 0)$
Pixel-wise addition: $O(p_i) = I_1(p_i) + I_2(p_i)$

Figure 3.3.3: The proposed FMOE-MR enhancement step. The steps in the dotted box create the Multi-Resolution (MR) rendering of the original video frame.

where **I** is the matrix with all of its entries as 1. The intermediate video frame, V_I, represents a multi-resolution re-encoding of the original video frame, V_O. However, V_I contains abrupt changes in resolution, which is not pleasing to the eye. Another smoothing operation is performed on the intermediate frame V_I with a Gaussian filter $G(\sigma_I)$, to yield the final frame V_F as a multi-resolution version of V_I. The detailed description of the FMOE-MR step is depicted pictorially in **Figure 3.3.3**. Note that the final video frame V_F is fed to the standard MPEG-4 Base Layer encoding pipeline, as shown in **Figure 3.3.1**.

The Base Layer video quality and encoded bit rate, after FMOE-MR and MPEG-4 video encoding, depend on the MR-Parameters σ_L, σ_H, σ_I and the FMO-Mask, depicted by the matrix **W**. The parameters σ_L, σ_H, and σ_I are bounded scalar quantities which control the bit rate; the weight matrix **W** controls the quality of the encoded video frame (it also controls, to a some extent, the encoded bit rate). Detailed discussions on the analysis, evaluation and performance of FMOE-MR are presented in the next section.

3.3.3. Analysis, Evaluation and Discussion

In this section, we first describe our evaluation methodology for the performance of the FMOE-MR scheme, followed by an objective comparison between the FMOE-MR and FGS-AQ schemes. Next, we analyze the rate distortion performance of FMOE-MR as a function of the three MR-Parameters σ_L, σ_H and σ_I. Finally, we analyze the rate distortion performance of FMOE-MR with respect to the FMO-Mask.

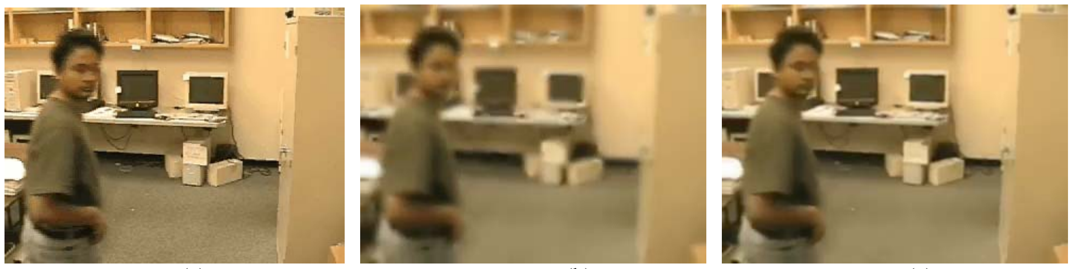
3.3.3.1. Evaluation Methodology

We have implemented the F-Mask using the Canny edge detector [Canny, 1986], and the M-Mask/O-Mask using the optical flow based multi-scale elastic matching algorithm given in [Luo, 2005b]. In order to have an objective quantification of video quality, we have used PSNR (peak signal to noise ratio) as the quality metric. Similarly, the bit rate is computed by measuring the size of the video file after FMOE-MR followed by MPEG-4 compression. A set of different videos has been used to compute the bit-rates and average PSNR per frame. The videos have been obtained under various background conditions (stationary, moving), various lighting conditions (moderately lighted, well lighted), various levels of motion complexity (single moving person, multiple moving persons), and various frame rates. The reported results in the section are based on the following four representative videos:

- *Video 1*: A 16 second video of a single person walking in a well lighted room. Frame rate: 30 fps, Frame Size: 320 × 240 pixels.
- *Video 2*: A 30 second panning view across a room in poor light (non-stationary background). Frame rate: 30 fps, Frame Size: 176 × 144 pixels.
- *Video 3*: Another panning video sequence of 30 seconds, this time at Frame Rate: 15 fps, Frame Size: 176 × 144 pixels.
- *Video 4*: A 40 second video of two people moving together in a well lighted room. Frame rate: 30 fps, Frame Size: 320 × 240 pixels.

3.3.3.2. FMOE-MR vs MPEG-4 FGS-AQ

In order to compare the proposed FMOE-MR technique with the existing MPEG FGS-AQ technique, we have compared the quality of video sequences (using PSNR) for a given target bit rate. We have used a Gaussian kernel for the low pass filter. **Figure 3.3.4** shows a frame of *Video 1*, encoded using FGS-AQ and FGS-MR, at the same bit rate (0.17 Mbps), using an edge mask as the F-mask. The visual quality of the video frame using F-MR (PSNR = 26.5 dB) is significantly better than that obtained by FGS-AQ (PSNR = 22.77 dB). We observed empirically that assigning $\sigma_L = 15$, $\sigma_H = 3$ in **equation (3.3.1)**, in order to obtain an F-MR representation of the videos, produces the best results for this video. We obtain these numbers by adjusting the parameters and asking human evaluators to judge the maximum extent of deterioration in the important and unimportant regions of the video frames (as depicted by the mask), that can be sustained such that the resulting video is still not unpleasant to view.



(a)
Original (5.12 Mbps)

(b)
MPEG-4 Adaptive Quantization
PSNR=22.77 dB (0.17 Mbps)

(c)
Proposed Multi-Resolution
PSNR = 26.5 dB (0.17 Mbps)

Figure 3.3.4: F-MR vs FGS-AQ: Comparison of video quality of *Video 1*, using the proposed F-MR (only edge mask is used) Base Layer encoding; compared with MPEG-4 Adaptive Quantization (FGS-AQ) Base Layer video encoding technique; (a) The original 320 X 240 frame; original AVI video is encoded at 5.12 Mbps (b) Video frame after MPEG-4 Adaptive Quantization for target bit rate around 0.17 Mbps; PSNR = 22.77 dB (c) Multi-Resolution Video frame; bit rate = 0.17 Mbps; PSNR = 26.5 dB.

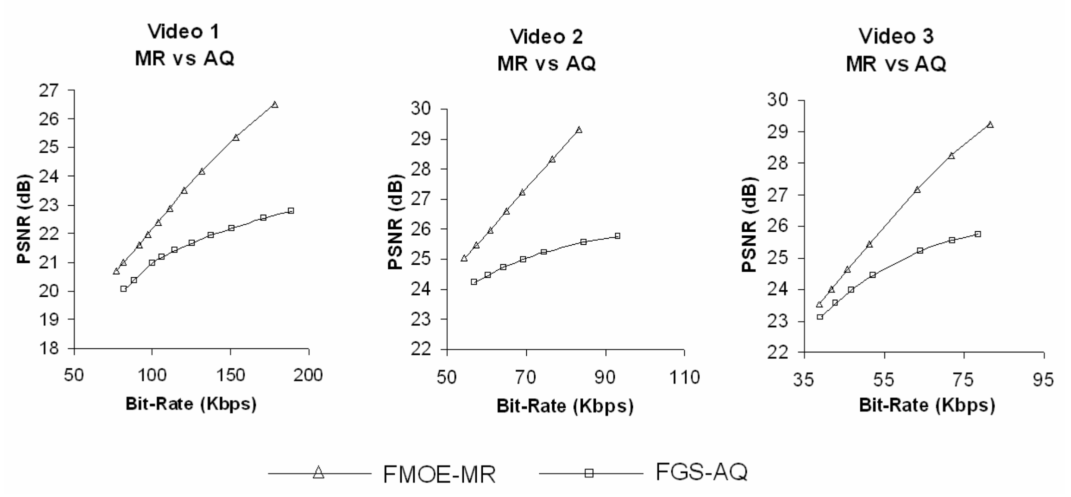


Figure 3.3.5: Rate-Distortion comparisons for the three video sequences, *Video 1* (30 FPS, 16 seconds), *Video 2* (30 FPS, 30 seconds) and *Video 3* (15 FPS, 30 seconds).

In order to provide objective evidence of the superior video quality resulting from FMOE-MR compared to FGS-AQ, **Figure 3.3.5** shows plots of the PSNR as a function of the target bit rate for *Video 1*, *Video 2* and *Video 3*, by fixing $\sigma_L = 15$, $\sigma_H = 3$ and varying σ_I from 3 to 25 in **equation (3.3.1)**. All the videos have been encoded using an F-

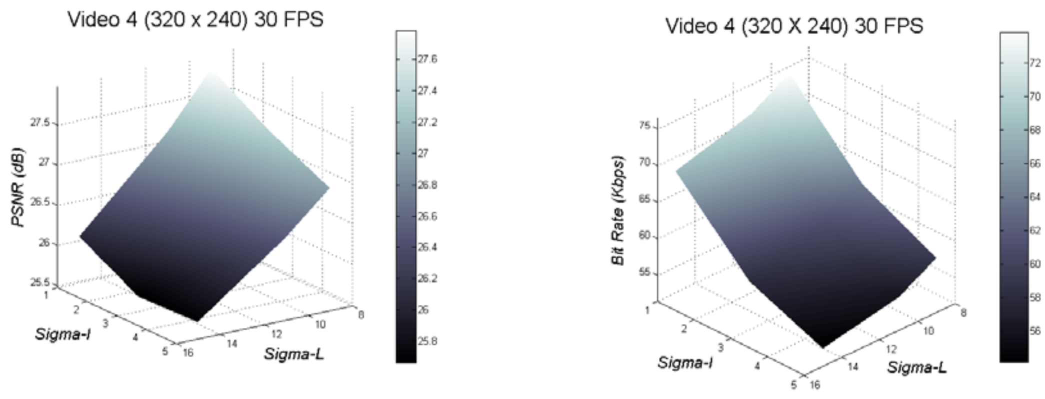


Figure 3.3.6: Rate distortion 3D plots of FMOE-MR vs σ_L σ_H and σ_I. $\sigma_H = 3$ in all cases.

Mask. Note that the PSNR values for FGS-MR are much higher than those for the FGS-AQ technique for the entire range of bit rates.

The results above show that the FMOE-MR technique yields a higher quality video for the same bit rate, compared to the FGS-AQ technique. This can be attributed to the fact that pixel-level enhancement can be performed in FMOE-MR, whereas FGS-AQ does block-based enhancements in the DCT space. Thus, in order to enhance even a single pixel in a block, FGS-AQ needs to enhance the whole block; although this improves the overall PSNR slightly, the overall bit rate requirement increases substantially. This is not the case for FMOE-MR, since FMOE-MR enhances a single pixel. The DCT coefficients in the corresponding block in the case of FMOE-MR may be enhanced less dramatically.

Thus, for the same target bit rate encoding of the Base layer for FGS, the resulting video quality (measured in terms of PSNR) is significantly better in the case of FMOE-MR adaptation, compared to standard MPEG-4 FGS-AQ adaptation.

In order to analyze the rate distortion performance of FMOE-MR as a function of the three MR-Parameters, σ_L, σ_H and σ_L, we have generated 3D plots of PSNR and bit rate as functions of the MR-Parameters. Since the plots for all the four videos are similar, we show the plots for only *Video 4* in **Figure 3.3.6**. This figure shows the dependence of PSNR and bit rate on the two MR-Parameters, σ_L and σ_I, by maintaining $\sigma_H = 3$ (Note that fixing σ_H to an empirically chosen value of 3 renders the foreground at a consistently good resolution). The two persons in *Video 4* have been tracked using the algorithm described in [Luo, 2005b], to create the motion mask (M-Mask), for the video sequence. As is evident from the 3-D plots in **Figure 3.3.6**, the bit rate varies more as a function of σ_I compared to σ_L. This is because σ_I smoothes the video frame uniformly overall, thus effectively truncating all the DCT coefficients of the frame. The dependence of PSNR on the values of σ_L than σ_I are not apparent, except for the fact that PSNR increases as either value decreases. From the analysis, it is clear that if the bit rate is the primary concern, the MR-parameter σ_I should be used to control the bit rate. The visual quality of the video frame depends on the proper combination of σ_L and σ_H. We have empirically found that $\sigma_H = 3$ renders the "*interesting*" regions defined by the mask at good visual quality, and also reduces the overall bit rate to some extent.

3.3.3.3. Computing the optimal values of the MR-Parameters (σ_L, σ_H, σ_I)

The optimal values of the MR-parameters can be computed by optimizing a suitable figure of merit function. We have devised a figure of merit function δ, which is the ratio of the visual quality (Q) to the obtained compression ratio (C) for a given video sequence. The metric δ is designed such that the higher its value, the better the rate distortion performance. We have defined δ as

$$\delta = Q/C \qquad (3.3.2)$$

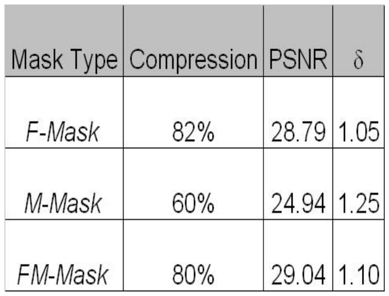
where Q = $2^{PSNR(\sigma L,\ \sigma H,\ \sigma I)/10}$ and C = Compression Ratio (FMOE-MR + MPEG-4 encoded versus MPEG-4-only encoded Base Layer). The expression Q is obtained from the standard PSNR equation: PSNR = 10log(Q), where Q is the ratio of the mean square error to the number of pixels in the frame. The metric δ is maximized for the video of best visual quality with the least bit rate requirement. Thus, the higher the metric δ, the more *efficient* the video encoding scheme. The values of σ_L, σ_H, σ_I which maximize δ can potentially be found by exhaustively enumerating all possible values of σ_L, σ_H, σ_I, and computing δ for each combination. The MR-Parameters σ_L, σ_H, σ_I are essentially the size of the Gaussian convolution masks, which take odd integral values, and are bounded from above. Hence, enumerating all possible combinations of σ_L, σ_H, σ_I is not as daunting as it seems, and is certainly computationally not very challenging for bounded, small values of the MR-Parameters.

3.3.3.4. FMOE-MR vs FMO-Mask

The FMO-Mask plays a significant role in the rate distortion performance of FMOE-MR. In order to compare the effect of the FMO-Mask on performance of FMOE-MR for *Video 4* (*Video 4* is chosen arbitrarily), we fix values of the MR parameters empirically as follows: $\sigma_L = 21$, $\sigma_H = 3$ and $\sigma_I = 1$. We have used three kinds of masks: F-mask (edge mask, implemented using the Canny edge operator [Canny, 1986]), M-mask (object tracking mask, implemented using the algorithm described in [Luo, 2005b]), and an *optimistic combination* of the two masks (FM-Mask).

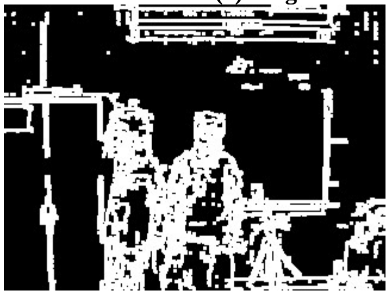
Figure 3.3.7 shows the multi-resolution re-rendering of the original frame based on the three types of masks mentioned above. The F-Mask has high resolution areas

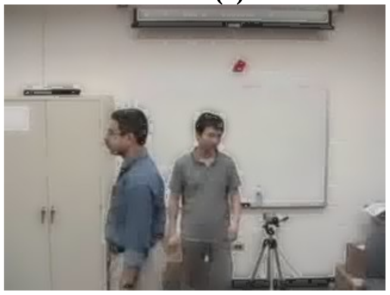
(a) Original

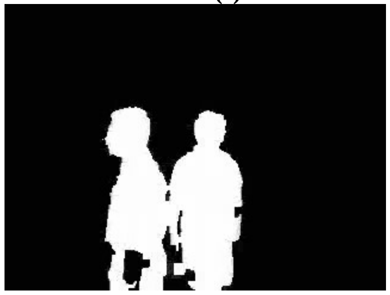


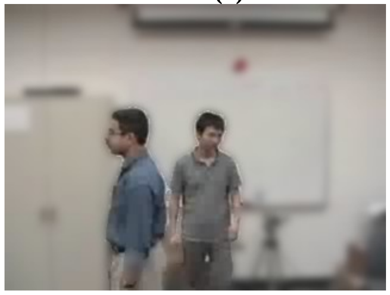
Mask Type	Compression	PSNR	δ
F-Mask	82%	28.79	1.05
M-Mask	60%	24.94	1.25
FM-Mask	80%	29.04	1.10

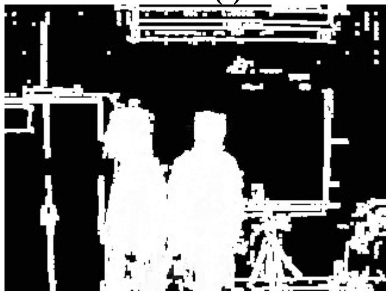
(b) Table 1


(c) F-Mask


(d) F-MR


(e) M-Mask


(f) M-MR


(g) FM-Mask

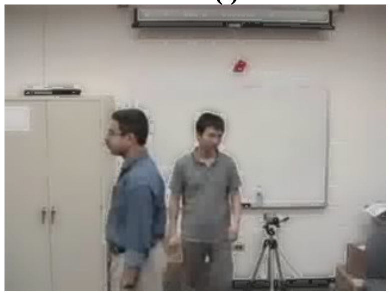

(h) FM-MR

Figure 3.3.7: Multi-resolution using different masks. Frame 1070 from *Video 4* (a) The original frame (b) The Table of results showing compression ratio, PSNR and evaluation metric (discussed in text) (c) The F-Mask, obtained by edge detection using Canny edge detector, and padding added (d) Multi-Resolution (MR) Frame, re-rendered aided by F-Mask (e) Motion mask using Motion tracking (f) MR-Frame aided by M-Mask (g) *Optimistically combined* FM-Mask (h) MR-Frame aided by FM-Mask.

spread over the entire frame. An interesting observation is that the effect of the M-mask is synonymous with the focusing of the human eye, which typically focuses on an object of interest by blurring the other uninteresting background objects. The *optimistically combined* FM-Mask delivers the best visual quality, as expected, but at a price of only a modest compression gain. **Figure 3.3.7(b)** shows the compression ratio (FMOE-MR versus standard MPEG-4), the corresponding PSNR obtained by using the various mask types, and the corresponding value of δ (using **equation (3.3.2)**) in a tabular format.

The values of δ reveal that the resulting compression from the M-Mask is the best in terms of rate distortion performance. A brief analysis will reveal why this is the case. Although the M-Mask seems to yield the worst overall PSNR, the bit rate obtained is significantly less as well, compared to that obtained by the other masks. The bit rate is significantly lower for the M-Mask because the white regions (i.e. semantically and visually important regions) in the frame are grouped together in the mask. The grouping results in large DCT coefficients in only the DCT blocks in which the white portions of the mask are present. In the multi-resolution frame resulting from the F-Mask and FM-Mask, on the other hand, the relevant regions (or white portions of the mask) are spatially distributed throughout the entire frame. This makes FMOE-MR less effective, because after using the proposed FMOE-MR scheme, the MPEG-4 DCT-based compression is faced with large DCT coefficients for almost all the DCT blocks in the frame.

3.3.3.5. Encoding time for FMOE-MR

The time T_{base} taken to encode a given raw video as an MPEG-4 FGS Base Layer is given by

$$T_{base} = T_f + T_m \qquad (3.3.3)$$

where T_f is the time taken to re-encode the given raw video frames in multi-resolution format, and T_m is the time required to encode the re-encoded raw video using the standard MPEG-4 video compression pipeline. Since standard MPEG-4 video encoding can be done in near real time, we limit our discussion to the complexity of T_f, given by:

$$T_f = T_{mask} + T_{\sigma L} + T_{\sigma H} + T_{merge} \qquad (3.3.4)$$

where T_{mask} is the time taken to create the mask, $T_{\sigma L}$ is the time taken to create the low resolution image, $T_{\sigma H}$ is the time taken to create the high resolution image, and T_{merge} is the time to merge the two different images. For an edge mask, the time to create the mask is *$O(nmN)$,* where the raw video has N 3-channel frames, each frame of size $n \times m$ pixels. Similarly, in the case of motion masks, optical flow analysis for each pair of successive frames is computationally the most complex aspect of the computation and takes *$O(nm)$* time; thus the overall time is *$O(nmN)$* for all the N frames. In general, the computational complexity for T_{mask} is *$O(nmN)$*. $T_{\sigma L}$ depends on the time taken for convolution of the image of size $n \times m$ with a Gaussian mask of constant size; thus $T_{\sigma L}$ is *$O(nmN)$* for all the N frames. Similarly, $T_{\sigma H}$ is also *$O(nmN)$.* Since the merging is done on a pixel-by-pixel basis, T_{merge} is *$O(nmN)$* for all the N frames. The video frame size of $n \times m$ pixels is $O(n^2)$ since n and m are typically scaled to a fixed ratio of 4:3. Thus, from equation (3.3.4), we get $T_f = O(n^2N)$, where n is the width (or height) of each video frame, and N is the total number of frames in the video. Theoretical analysis apart, we have achieved real time encoding speed quite comfortably (more than 30 frames per second) for *Edge-Mask* based MR.

3.3.4. Conclusions and future work

A novel multi-resolution Base Layer encoding technique for MPEG-4 fine grained scalability (FMOE-MR) video encoding has been described and implemented. Results show that the rate distortion performance of the proposed FMOE-MR technique is significantly better than that of the existing MPEG-4 adaptive quantization technique (FGS-AQ) for FGS Base Layer encoding. FMOE-MR entails "*smart*" preprocessing of the video prior to MPEG-4 encoding for creation of the Base Layer for FGS; as a result, existing codecs for creating FGS video can be easily used in the proposed scheme. In addition, FMOE-MR is transparent to the decoder; FGS-AQ, on the other hand, requires special AQ parameters, and components, at the decoder end to reconstruct the video.

Since FMOE-MR is a mask based technique; the effectiveness of the MR video depends on the creation of the FMO-Mask. The FMO-Mask is designed to highlight features, motion and objects in the video sequence. We have proposed unsupervised and semi-supervised algorithms for effective creation of the FMO-Mask from any given video sequence.

FMOE-MR is more than just a tool; it is a whole new approach to intelligent, content-based scalable video encoding. Since FMOE-MR is inherently parametric, a potential future research endeavor will be to compute the MR-Parameters automatically from the given video. In addition, many new types of masks may be used, such as application-specific object masks. We are working on real-time applications where the masks can be created in real time in order to facilitate applications such as video conferencing in the presence of dynamically changing quality constraints.

CHAPTER 4

HYBRID LAYERED VIDEO ENCODING

4.1. Introduction

Video playback on a mobile device such as a PDA, pocket-PC, multimedia-enabled mobile phone (such as an iPhone), or a laptop PC operating in battery mode, is a resource-intensive task in terms of CPU cycles and battery power [Sikora, 2005]. Video playback typically results in rapid depletion of battery power in the mobile device, regardless of whether the video is streamed from a hard drive on the device, or from a remote server. Several techniques have been proposed to reduce power consumption during video playback on the mobile device [Cucchiara, 2003], [Ni, 2003], [Liang, 2006] [Mohapatra, 2003] [Cornea, 2006]. These techniques use various hardware and software optimizations to reduce power consumption during video playback. Typically, power savings are realized by compromising the quality of the rendered video. This tradeoff is not always desirable, since the user may choose to watch the video at its highest quality if sufficient battery power is available on the device. Thus, it is desirable to formulate and implement a multi-layer encoding of the video such that distinct layers of the video display different power consumption characteristics. The lowest layer should consume the least power during video decoding and rendering. The power consumption during video decoding and rendering should increase as more layers are added to the video. Typically, the less battery power available to decode and render the video, the lower the quality of the rendered video [Dai, 2003]. Thus, it is necessary to enhance quality of the lower video layers in order to ensure that the quality of the rendered video is acceptable.

Traditional layered video encoding, as used by MPEG-4 Fine Grained Scalability profile (MPEG-FGS), is customized for varying bitrates, rather than power-adaptive usage. In this chapter, the design and implementation of a novel *Hybrid Layered Video* (HLV) encoding scheme is presented. The proposed representation is termed as “hybrid” due to the fact that its constituent layers are a combination of standard MPEG-based video encoding and a generative sketch-based video representation. The input video stream is divided into two components: a *sketch* component and a *texture* component. The sketch component is a *Generative Sketch-based Video* (GSV) representation, where the outlines of the objects of the video are represented as curves [Chattopadhyay, 2007b]. The evolution of these curves (termed as *pixel-threads*), across the video frames is explicitly modeled in order to reduce temporal redundancy. A considerable body of work on object-based video representation using graphics overlay techniques has been presented in the literature [Khan, 2003] [Salembier, 2003] [Ku, 2003]. These methods are based primarily on the segmentation of the video frames into regions and the subsequent representation of these regions by closed contours. A major drawback of the aforementioned contour-based representation is the fact that the complexity of the representation increases significantly with an increasing number of contours in the video frames. In contrast, the proposed GSV representation uses sparse parametric curves, instead of necessarily closed contours, to represent the outlines of objects in the video frames. This ensures that the number of graphical objects that one needs to overlay is small. In addition, whereas closed contours are capable of addressing local region-based consistency, global shape-based information may be seriously compromised. This is not so in the case of the proposed GSV representation, which ensures that the global shape is correctly represented. Although contour-based representations have been very successful in some specific applications involving low-bitrate videos such as video phones [Hakeem, 2003], generic contour-based video representations for a wider class of power-constrained devices have, thus far, not been studied in detail.

The texture component in the proposed HLV encoding scheme is represented by three layers; a base layer video, an intermediate mid-layer video, and the original video. The base layer represents a very low bitrate video with very low visual quality whereas the highest layer in the HLV representation denotes the original video. The base layer video can be augmented by the object outlines (that are emphasized with dark contours) using the Generative Sketch-based Video (GSV) representation mentioned above. This ensures that the visual quality of the base layer is improved significantly. The visual quality of the mid-layer video is better than that of the base layer video, but lower than that of the original video. The mid-layer video quality is further enhanced via high-level object-based re-rendering of the video at multiple scales of resolution. The result is termed as a *Features, Motion and Object-Enhanced Multi-Resolution* (FMOE-MR) video [Chattopadhyay, 2007a]. Note that although the visual quality of the mid-layer video is lower than that of the original video, some semantically relevant portions of the frames in the mid-layer video are highlighted by selectively rendering them at higher resolution, thus enhancing the overall viewing experience of the end user.

We show via formal analysis and experimental results that the various video layers in the proposed HLV representation have different power consumption characteristics. Thus, the overall power consumption of an HLV-encoded video depends on the combination of layers used during decoding and rendering of the video on the

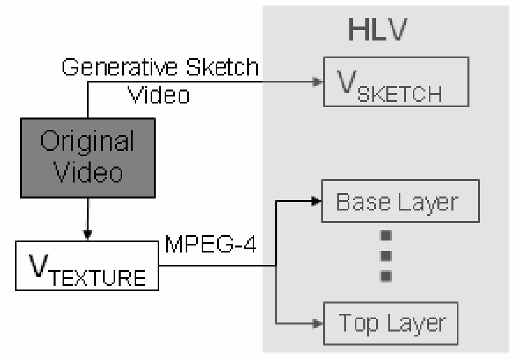


Figure 4.1. The Hybrid Layered Video (HLV) Scheme.

mobile end user device. A direct benefit of the proposed HLV representation is that the video content can be decoded and rendered at different levels of power consumption on the mobile device. Experimental results show that it is possible to save 45 minutes of battery power for an IBM Thinkpad laptop PC with a 2 GHz Centrino Processor and 1 GByte RAM by using the lowest layer of the proposed HLV representation (comprising only of the sketch component without any texture content) when compared to the playback of the original video. Adding approximated texture to the sketch component results in an average battery power savings of 30 minutes when compared to the playback of the original video. The proposed HLV representation is thus extremely well suited for video streaming to power-constrained devices, such as multimedia-enabled mobile phones, PDAs, pocket-PCs and laptop computers operating in battery mode, where the available power for video playback typically changes over the video playback duration. A schematic diagram depicting the proposed Hybrid Layered Video (HLV) representation is given in **Figure 4.1**.

In the following sections, we elaborate upon the two components of the proposed HLV representation, i.e., the sketch component, $\mathbf{V_{SKETCH}}$ and the texture component, $\mathbf{V_{TEXTURE}}$. This is followed by a description of how to combine the various video layers comprising the aforementioned components to result in a power-scalable video representation. We discuss issues pertaining to the implementation of the proposed HLV representation followed by a presentation and analysis of the experimental results. Finally, we conclude the chapter with an outline of future research directions.

4.2. Creating video component V_{sketch}

In this section, we describe a technique to represent a video stream as a sequence of *sketches,* where each sketch in turn is represented by a sparse set of parametric curves. The sketch-based video essentially represents the outlines of the objects in the video. The resulting video representation is termed a *Generative Sketch-based Video* (GSV).

The video is first divided into a series of *Groups of Pictures* (GOPs), in a manner similar to standard MPEG video encoding. Each Group of Pictures (GOP) consists of N frames (typically, N = 15 for standard MPEG/H.264 encoding) where each frame is encoded using the following four steps:

1. The object outlines are extracted in each of the N frames. These outlines are represented as a sparse set of curves.
2. The curves in each of the N frames are converted to a suitable parametric representation.
3. Temporal consistency is used to remove spurious curves which occur intermittently in consecutive frames. These spurious curves, if not removed, create an undesirable flickering effect.
4. Finally, the parametric curves in the N frames of the GOP are encoded in a compact manner. The first frame of the GOP enumerates the curve parameters in a manner that

is independent of their encoding, analogous to the I-frame in MPEG H.264 standard. The remaining N-1 frames in the GOP are encoded using motion information derived from previous frames, in a manner analogous to the P-frames in the MPEG H.264 standard.

The proposed GSV encoding is similar the MPEG video encoding standard. The GOP is a well established construct in the MPEG standard that enables operations such as fast forward, rewind and frame dropping to be performed on the encoded video stream. Motion vectors are used in GSV encoding to reduce temporal redundancy as in the case of MPEG video encoding, where motion vectors are used to describe the translation of frame blocks relative to their positions in previous frames. The error vector, in the case of GSV encoding, has the same form as the encoded representation of the moving object(s) in the video. This is analogous to MPEG video encoding, where the encoding error is represented in the form of macroblocks similar to the macroblock representation of the moving object(s) in the video.

The parametric curves used to represent the object outlines in each frame are termed as *pixel-threads*. A *pixel-thread* is derived from a polyline P:[0, N], which is a continuous and piecewise linear curve made of N connected segments. A polyline can be parameterized using a parameter $a \in \mathbf{R}$ (set of real numbers) such that $P(a)$ refers to a specific position on the polyline, with $P(0)$ referring to the first vertex of the polyline and $P(N)$ referring to its last vertex. Note that the *pixel-threads* contain information only about the vertices (or break points) of the polyline. Note that these break points can be joined by straight line segments (as in the case of a polyline), or by more complex spline-based functions to create smooth curves.

The *pixel-threads* essentially depict the outlines of objects in the underlying video stream. Each video frame is associated with its own collection of *pixel-threads* termed as a *Pixel-thread-Pool*. Thus, successive video frames are associated with successive *Pixel-thread-Pools*. Due to the temporal nature of the video, the *pixel-threads* and *Pixel-thread-Pool* are modeled as dynamic entities that evolve over time to generate the outlines of the moving objects in the video. The dynamic nature of the *pixel-threads* is modeled by the processes of *birth* and *evolution* of *pixel-threads* over time. We provide a detailed description of these processes in the following subsections.

4.2.1. Birth of pixel-threads

For a given video frame, the *Pixel-thread-Pool* is created by first generating (or sketching) the outlines of the objects in the video frame, and then representing these outlines parametrically in the form of *pixel-threads*.

4.2.1.1. Generating a sketch from a video frame

The edge pixels in a video frame are extracted using the Canny edge detector [Canny, 1986]. The edge pixels are grouped to form one-pixel wide edge segments or *edgels*, many of which are intersecting. Edgels of small length are removed to avoid excessively cluttered sketches. The threshold below which an edgel is considered “small” depends on the screen size. Since GSV encoding is typically meant for mobile devices with small screens, removal of these small edgels typically do not produce any adverse effect. It must be noted that the edge detection process is inherently sensitive to noise and several edgels may, in fact, be noisy artifacts. Edgels extracted in two successive frames may cause *flickering*; i.e., an edgel in a previous frame may disappear in the current frame, even in instances where the human eye can clearly discern an object boundary. A method to reduce this flickering effect is described in Section 4.2.4.

4.2.1.2. Creating pixel-threads from a sketch

The sketch thus obtained is converted to an approximate parametric representation using curve approximation techniques proposed by Rosin and West [Rosin, 1995]. Rosin and West [Rosin, 1995] describe the implementation and demonstrate the performance of an algorithm for segmenting a set of connected points resulting in a combination of parametric representations such as lines, circles, ellipses, super-elliptical arcs, and higher-order polynomial curves. The algorithm is scale invariant (i.e., it does not depend on the size of the edgels, or the size of the frame), nonparametric (i.e., it does not depend on predefined parameters), general purpose (i.e., it works on any general distribution of pixels depicting object outlines in any given video frame), and efficient (i.e., has low computational time complexity). Since a detailed discussion of the algorithm is beyond the scope of the chapter, it suffices to mention that we use this algorithm to determine break points on the various connected components (i.e., edge segments) that are generated after the edge pixels have been detected.

A curved edge segment is represented by a series of break points along the curve, determined using the algorithm of Rosin and West [Rosin, 1995]. The curved edge segment is deemed to represent a portion of the outline of an object in the scene. Thus, the fitting of straight lines between the break points results in a rendering of an *approximate* version of the original curve. The break points are essentially points of

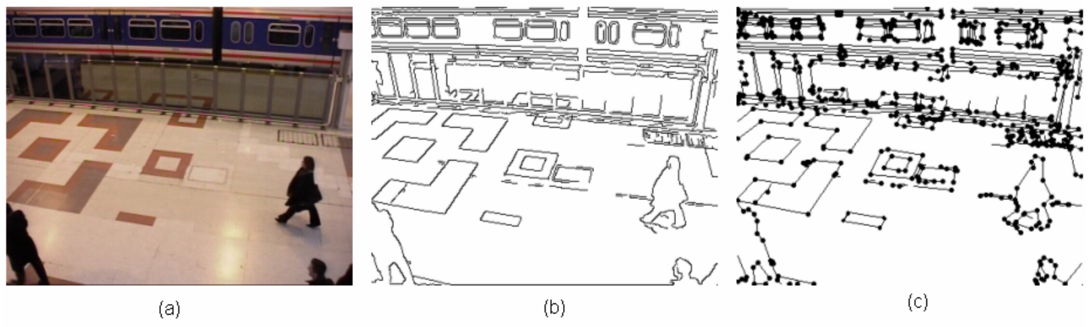


Figure 4.2. The creation of *pixel-threads* for a video frame (a) The original video frame; (b) Edges detected in the video frame, and filtered to remove small, spurious edges; (c) Break-points detected in the edge contours generated in the previous step.

significance along the curve, such as corners and high-curvature points. Altering the level of significance or threshold for break-point detection allows various levels of (break-point based) approximation of the contours in the video frame. This is discussed in greater detail in Section 4.5.4.1.

The break points along the curve are represented efficiently as a chain-coded vector. For each approximated curve *i*, one of the end points (first or last break point) is represented using absolute coordinates $\{x_0, y_0\}$ whereas the *p*-th break point, where $p > 0$, is represented by coordinates relative to those of the previous break point; i.e. $\{\delta x_p, \delta y_p\}$ where $\delta x_p = x_p - x_{p-1}$ and $\delta y_p = y_p - y_{p-1}$. The resulting chain-coded vectors constitute the *pixel-threads* which are approximations to the original curve. **Figure 4.2** illustrates the process by which *pixel-threads* are generated for a given video frame. Note that *pixel-thread* creation is done offline during the encoding process.

4.2.2. Evolution of a pixel-thread

Due to the temporal redundancy in a video sequence, a majority of the corresponding *pixel-threads* in successive video frames are often similar in shape and size. This temporal redundancy can be exploited to *evolve* some of the *pixel-threads* in the current frame to constitute the *Pixel-thread-Pool* for the successive frame. An advantage of evolving the *pixel-threads* over successive video frames is the fact that a *pixel-thread*, once *born* in a video frame, requires only motion information to characterize its behavior in successive frames. Motion modeling significantly reduces the amount of information required to render the set of *pixel-threads* corresponding to the next frame, resulting in a compact representation of the dynamic *pixel-threads*. The evolution parameters are determined during the encoding process, which is offline and typically not power constrained. As will be shown subsequently, the encoding is done in a manner such that the decoding is simple, and can be done in real time by a power constrained device.

The evolution of *pixel-threads* between two successive *Pixel-thread-Pools,* say TP_1 *and* TP_2, involves two steps; (a) establishing the *pixel-thread* correspondence between the two *Pixel-thread-Pools*, and (b) estimating the motion parameters.

4.2.2.1. Establishing pixel-thread correspondence

In order to model the underlying motion accurately, it is essential to establish the correspondence between *pixel-threads*, belonging to the *Pixel-thread-Pools* of two consecutive frames in the video stream. In order to determine the correspondence between *pixel-threads* in TP_1 and TP_2 one needs to determine for each *pixel-thread* in TP_1 its counterpart in TP_2.

First, we need to predict a position to which a *pixel-thread* $\mathbf{T_1}$ in TP_1 is *expected* to move in the next frame. The predicted *pixel-thread*, say **T'**, can be determined using a suitable optical flow function *OpF*, such that

$$\mathbf{T'} = OpF(\mathbf{T_1}) \qquad (4.1)$$

The function *OpF*() computes the coordinates of the break points of the *pixel-thread* **T'** in *TP*$_2$, given the coordinates of the break points of *pixel-thread* $\mathbf{T_1}$ in *TP*$_1$. The function *OpF*() implements a sparse iterative version of the Lucas-Kanade optical flow algorithm designed for pyramidal (or multiscale) computation [Bouguet]. The Lucas-Kanade algorithm is a popular version of a two-frame differential technique for motion estimation (also termed as optical flow estimation). For each break point location (x,y) of a *pixel-thread*, if the corresponding pixel location in the original image (frame) has intensity I(x,y); and is assumed to have moved by δx and δy between the two frames, then the image constraint equation is given by:

$$I_{\text{current-frame}}(x,y) = I_{\text{next-frame}}(x + \delta x, y + \delta y)$$

The Lucas-Kanade algorithm essentially embodies the above image constraint equation. The pyramidal implementation of the Lucas-Kanade algorithm computes the optical flow in a coarse-to-fine iterative manner. The spatial derivatives are first computed at a coarse scale in scale space (i.e., in a pyramid), one of the images is warped by the computed deformation, and iterative updates are then computed at successively finer scales.

Once the *pixel-thread* **T'** is obtained from $\mathbf{T_1}$ via the optical flow function, we hypothesize that if *pixel-thread* $\mathbf{T_1}$ in *Pixel-thread-Pool TP*$_1$ does indeed evolve to a corresponding *pixel-thread* $\mathbf{T_2}$ in *TP*$_2$, then **T'** and $\mathbf{T_2}$ should resemble each other (to a reasonable extent) in terms of shape and size. The key is to determine the *pixel-thread* $\mathbf{T_2}$ in *TP*$_2$, which is closest in shape and size to the *pixel-thread* **T'**.

The correspondence between pixel threads **T'** and $\mathbf{T_2}$ is determined using the *Hausdorff distance* [Atallah, 1983]. The Hausdorff distance is used as a measure of

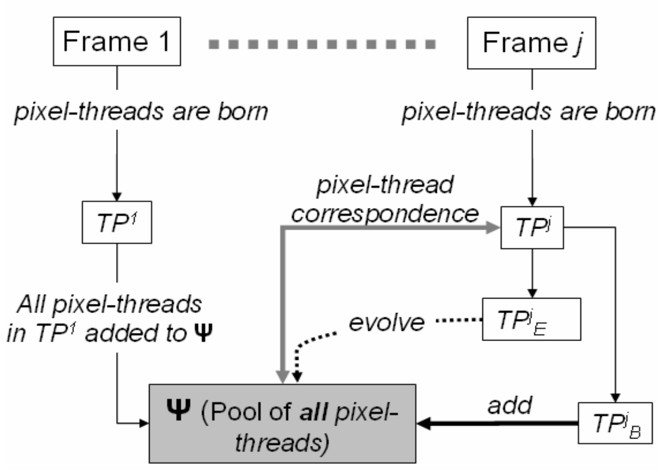

Frame 1
Frame j
pixel-threads are born
pixel-threads are born
TP1
pixel-thread correspondence
TPj
All pixel-threads in TP1 added to Ψ
evolve
TPjE
Ψ (Pool of all pixel-threads)
add
TPjB

Figure 4.3. Establishing *pixel-thread* correspondence for a frame *j* and the current *All-Threads-Pool* Ψ.

(dis)similarity between the *pixel-threads*, **T'** and $\mathbf{T_2}$. The Hausdorff distance between the two *pixel-threads* **T'** and $\mathbf{T_2}$, denoted by $\delta_H(\mathbf{T'}, \mathbf{T_2})$, is defined as

$$\delta_H(\mathbf{T'}, \mathbf{T_2}) = max_{\,\mathbf{a} \in \mathbf{T'}} \{min_{\,\mathbf{b} \in \mathbf{T2}} \{ d(\mathbf{a}, \mathbf{b}) \} \} \tag{4.2}$$

where **a** and **b** are break points, and d(**a**, **b**) is the Euclidean distance between them. Thus, given *pixel-thread* $\mathbf{T_1} \in TP_1$, $\mathbf{T_2}$ is essentially the *pixel-thread* in TP_2 which is most similar to **T'**, where **T'**, in turn, is obtained from $\mathbf{T_1}$ using the optical flow mapping function *OpF*(); i.e.

$$\mathbf{T_2} = argmin\{ \delta_H(OpF(\mathbf{T_1}), \mathbf{T}): \mathbf{T} \in TP_2\} \tag{4.3}$$

An important observation about the computation of the Hausdorff distance δ_H is that the two *pixel-threads* under consideration, $\mathbf{T_1}$ and $\mathbf{T_2}$, need not have the same number of break points.

Although the *pixel-thread* $\mathbf{T_2}$ in TP_2 is deemed to be the closest evolved *pixel-thread* to $\mathbf{T_1}$ in TP_1, it might still *not* have actually evolved from $\mathbf{T_1}$. As a result, we define a threshold $\varepsilon > 0$, such that if $\delta_H(OpF(\mathbf{T_1}), \mathbf{T_2}) < \varepsilon$, then we consider $\mathbf{T_2}$ to have evolved from $\mathbf{T_1}$; otherwise, $\mathbf{T_1}$ is deemed to have become *dormant*, and $\mathbf{T_2}$ in TP_2 is deemed to have been *born* in TP_2 and not evolved from TP_1. We specify the threshold ε as an empirically determined fraction of the video frame width.

Based on the above definitions of *birth* and *evolution*, the *pixel-threads* in TP_2 can be categorized as belonging to two mutually exclusive sets, TP^{Evolve} and TP^{born}. TP^{Evolve} is the set of all *pixel-threads* in TP_2 which are evolved from some *pixel-thread* in TP_1, and TP^{born} is the set of *pixel-threads* in TP_2 which are not evolved from TP_2; in other words, these *pixel-threads* are deemed to have been born in TP_2. **Figure 4.3** provides a schematic description of this process.

4.2.2.2. Motion modeling of pixel-threads

In this section, we discuss how to encode the motion information needed to specify the evolution of a *pixel-thread* $\mathbf{T_1}$ in TP_1 to its counterpart $\mathbf{T_2}$ in TP_2 once the correspondence between the *pixel-threads* $\mathbf{T_1}$ and $\mathbf{T_2}$ has been determined as described in the previous subsection. The visual quality and computational efficiency of the final encoding requires accurate estimation of the motion of *pixel-thread* $\mathbf{T_1}$ as it evolves into *pixel-thread* $\mathbf{T_2}$. To ensure compactness of the final representation, we assume that a linear transformation $L_{\mathbf{T}}$, specified by the translational parameters $\{t_x , t_y\}$, can be used for the purpose of motion estimation. It must be noted that transformations that incorporate additional parameters such as rotation and scaling, and are based on affine motion models can also be used. However, simple translational motion requires the least number of bytes for representation. Moreover, even if a more accurate and comprehensive motion model were to be used, an encoding error term would still need to be computed. The encoding error generated by an accurate and comprehensive motion model, although smaller in magnitude compared to that generated by a simple motion model consisting of translational parameters only, would still require approximately the same number of bytes for representation. For example, a byte would be required to represent the encoding error in both cases, whether the error is 1 pixel or 255 pixels. Thus, using the simplest motion model (consisting only of translational parameters), and keeping track of the resulting encoding error, results in a compact and efficient motion representation. Note that a similar motion model is used for motion compensation by the well established MPEG standard. Also note that the simple translational motion model is adequate when the temporal sampling rate of the video (measured in frames per second) is high enough compared to the velocities and complexities of the motions of the various objects within the video. In such cases, even complex motions between successive frames can be reasonably approximated by a motion model comprising only of translational parameters.

Thus, the estimated *pixel-thread*, $\mathbf{T_2}^{\mathbf{estimated}}$ is computed from $\mathbf{T_1}$ by using a mapping function $L_{\mathbf{T1}}$, such that

$$\mathbf{T_2}^{\mathbf{estimated}} = L_{\mathbf{T2}}(\mathbf{T_1})$$

The linear transformation coordinates in $L_{\mathbf{T2}}$ can be determined by computing the mean of the displacements of each break point, where the displacement of each break point is computed using the function *OpF()* (**equation (4.1)**). Since $\mathbf{T_2}^{\mathbf{estimated}}$ may not align exactly point-by-point with $\mathbf{T_2}$, it is necessary to compute the error between $\mathbf{T_2}^{\mathbf{estimated}}$ and $\mathbf{T_2}$. As discussed in the previous subsection, $\mathbf{T_2}^{\mathbf{estimated}}$ and $\mathbf{T_2}$ may not have the same number of break points. Suppose the number of break points of $\mathbf{T_1}$, and hence, $\mathbf{T_2}^{\mathbf{estimated}}$, is n_1 and that of $\mathbf{T_2}$ is n_2. In general, $n_1 \neq n_2$. Let the displacement error between $\mathbf{T_2}^{\mathbf{estimated}}$ and $\mathbf{T_2}$, be given by the displacement vector $\mathbf{\Delta T_2}$**.** Two cases need to be considered:

Case 1: $n_2 \leq n_1$: This means that there are fewer or equal number of break points in $\mathbf{T_2}$ compared to $\mathbf{T_2}^{\mathbf{estimated}}$. Note that, each component of $\mathbf{\Delta T_2}$ is a relative displacement required to move each break point of $\mathbf{T_2}^{\mathbf{estimated}}$ to one of the break points in $\mathbf{T_2}$. Obviously, there can be multiple break points in $\mathbf{T_2}^{\mathbf{estimated}}$ which map to the same break point in $\mathbf{T_2}$.

Case 2: $n_2 > n_1$: In this case the encoding is slightly different. The first n_1 components of $\mathbf{\Delta T_2}$ denote the displacements corresponding to break points in $\mathbf{T_2}$ in the same order. Each of the remaining $(n_2 - n_1)$ components of $\mathbf{\Delta T_2}$ are now encoded as displacements from the last break point in $\mathbf{T_2}$.

From the above description, it can be seen that the displacement vector $\mathbf{\Delta T_2}$ has max(n , n_2) components. Thus, the motion model required to evolve *pixel-thread* $\mathbf{T_1}$ into *pixel-thread* $\mathbf{T_2}$, is given by

$$\Theta_{\mathbf{T1}}(\mathbf{T_2}) = \{ t_x , t_y , \mathbf{\Delta T_2}\} \qquad (4.4)$$

The motion model $\Theta_{\mathbf{T1}}(\mathbf{T_2})$ essentially contains all the parameters needed to transform *pixel-thread* $\mathbf{T_1}$ in TP_1 to *pixel-thread* $\mathbf{T_2}$ in TP_2.

Let us now consider the number of bytes required to encode the motion model $\Theta_{\mathbf{T1}}(\mathbf{T_2})$. The transformation parameters $\{t_x,\ t_y\}$ can be designed to require a byte (character) each by restricting the displacement values to lie in the range (-127, 128). If t_x or t_y exceeds these bounds, then the result of the correspondence determination procedure is declared void, and **T'** is deemed to be a new *pixel-thread* that is born, instead of one that is evolved from *pixel-thread* **T**. However, in practice, for small display screens typical of mobile devices, this case occurs very rarely.

The prediction error vector $\mathbf{\Delta T_2}$ requires 2 bytes for each component, if the displacements δx and δy are restricted to lie within a range {-128, 127}. Thus, $\mathbf{\Delta T_2}$ requires 2•max(n_1 , n_2) bytes of storage. Hence, the total storage requirement of the motion model $\mathbf{\Theta_{T1}(T_2)}$ (in bytes), denoted by Bytes($\Theta_{\mathbf{T1}}$), is given by

$$\text{Bytes}(\Theta_{\mathbf{T1}}) = 2\bullet\max(n_1, n_2) + 2 \qquad (4.5)$$

In the next section, we present a method to evolve an entire generation of *pixel-threads* as a function of time. This results in the generation of a sketch-based representation of the original video sequence.

4.2.3. Evolution of a pixel-thread-pool

Given a video sequence of *N* frames, and the current frame *j*, let $\mathbf{\Psi}$ be the pool of all the *pixel-threads* which have been *born* or *evolved* thus far in frames 1 through *j* -1. All the *pixel-threads* in $\mathbf{\Psi}$ may not be active, i.e., some may be dormant. The dormant *pixel-threads* still belong to $\mathbf{\Psi}$, but represent *pixel-threads* which were not used to sketch a curve in the previous frame, *j* -1. The *pixel-threads* in $\mathbf{\Psi}$ belong to one of two subsets; $\mathbf{\Psi}_{\text{dormant}}$ or $\mathbf{\Psi}_{\text{active}}$. Clearly, $\mathbf{\Psi} = \mathbf{\Psi}_{\text{dormant}} \cup \mathbf{\Psi}_{\text{active}}$.

For the current frame *j*, the *pixel-threads* corresponding to frame *j* are first determined using the techniques discussed in Section 4.2.1. These recently acquired *pixel-threads* corresponding to frame *j* are grouped together in *Pixel-thread-Pool* TP_j. Assume that TP_j has n_j *pixel-threads* $\{\mathbf{T^1_j}, \mathbf{T^2_j} \ldots \mathbf{T^{nj}_j}\}$. Next, the correspondence between the *pixel-threads* in the set $\mathbf{\Psi}$ and the n_j *pixel-threads* in TP_j, is determined using the methods mentioned in Section 4.2.2. Note that per the terminology in Section 4.2.2, $\mathbf{\Psi}$ corresponds to TP_1 and TP_j corresponds to TP_2. Note that, during the correspondence determination procedure, the dormant *pixel-threads* in $\mathbf{\Psi}$ are also considered.

Let $TP_j^{wereEvolved}$ be the subset of the *pixel-threads* in TP_j which have evolved from $\mathbf{\Psi}$. Let $\mathbf{\Psi}_{TPj}$ be the subset of *pixel-threads* in $\mathbf{\Psi}$ which evolve to a corresponding *pixel-thread* in $TP_j^{wereEvolved}$. It must be noted that, the correspondence between $\mathbf{\Psi}_{\mathrm{TPj}}$ and $TP_j^{\mathrm{wereEvolved}}$ is determined in a manner such that a *pixel-thread* $\mathbf{T^i_j}$ belonging to $TP_j^{wereEvolved}$ and the corresponding *pixel-thread* in $\mathbf{\Psi}_{TPj}$ from which it has evolved are both assigned the same index *i.* Now, *pixel-threads* in $\mathbf{\Psi_{TPj}}$ can be evolved to corresponding *pixel-threads* in $TP_j^{wereEvolved}$ via a set of motion models $\mathbf{\Theta_{\Psi TPj}}$ (**equation (4.4)**). Since the remaining *pixel-threads* in TP^i_j cannot be evolved from any existing *pixel-thread* in $\mathbf{\Psi}$**,** these *pixel-threads* are considered to belong to the set TP_j^{born}; where $TP_j^{born} = TP_j - TP_j^{\mathrm{wereEvolved}}$.

Next, the set $\mathbf{\Psi}$ is updated in the following manner:

(a) *Pixel-threads* in $\mathbf{\Psi}_{\mathrm{TPj}}$ are *evolved* to corresponding *pixel-threads* in $TP_j^{\mathrm{wereEvolved}}$, using motion model parameters given by $\mathbf{\Theta_{\Psi TPj}}$. The new *pixel-threads* in TP_j^{born} are included in $\mathbf{\Psi}$**.**

(b) The new set of active *pixel-threads* is given by $\mathbf{\Psi_{active}} = \mathbf{\Psi} \cap TP_j$. These *pixel-threads* are used to generate the sketch-based representation of the new video frame. Naturally, the *pixel-threads* in this updated set $\mathbf{\Psi}$, that have no counterparts in TP_j, are deemed *dormant*; i.e.,

$$\mathbf{\Psi_{dormant} = \Psi - \Psi_{active}}$$

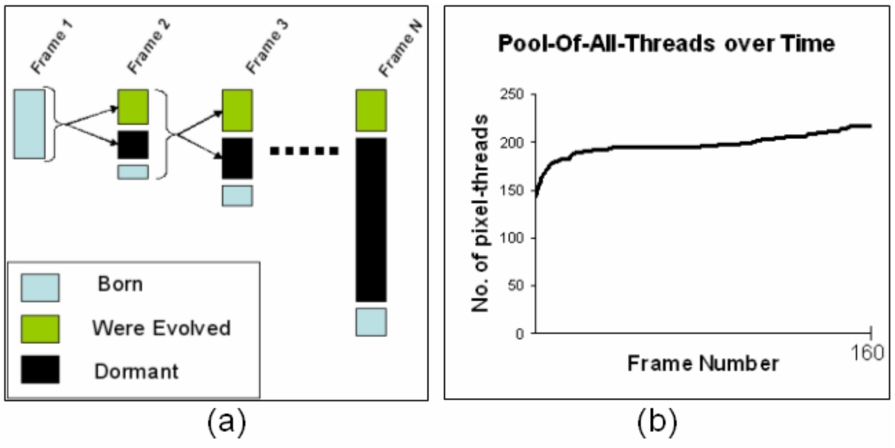


Figure 4.4. The process of birth and evolution of *pixel-threads* across the frames to create the Generative Sketch-based Video (GSV) (a) The vertical bars represent the state of the *Pool-of-All-Pixel-Threads* Ψ_i for frame i; (*b*) For a video sequence of 160 frames, the number of *pixel-threads* in each frame is plotted as a function of the frame number (time).

The data corresponding to frame j required to sketch the j^{th} video frame are given by the motion model parameters denoted by $\mathbf{\Theta_{\Psi TPj}}$. The newly born *pixel-threads* are included in TP_j^{born}. Thus, the entire process of evolution of all the *pixel-threads* across all the N frames of the video can be effectively represented as a *Generative Sketch-based Video (GSV)*, given by

$$\textit{Generative-Sketch-based-Video} = \{< \mathbf{\Theta_{\Psi TP1}}, TP_1^{born}>,.., < \mathbf{\Theta_{\Psi TPN}}, TP_N^{born}>\} \qquad (4.6)$$

A depiction of the process of evolution of the GSV is given in **Figure 4.4a**. **Figure 4.4b** shows a plot of the total number of *pixel-threads* in $\mathbf{\Psi}$ as a function of time during the entire process of evolution of all the *pixel-threads* in the GSV representation of a sample video. The curve shows that, after the initial *pixel-threads* are created in the first frame, very few new *pixel-threads* are born thereafter. The initial *pixel-threads* are seen to be adequate to evolve and generate most of the entire GSV representation of the sample video.

4.2.4. Flicker reduction

When the *pixel-threads* for each frame are rendered, flickering effects are observed. This is due to the fact that some pixel threads appear momentarily in a frame, only to become dormant in a series of successive frames. The resulting sudden appearance and disappearance of *pixel-threads* creates a flickering effect. The *All-Threads-Pool* **Ψ** contains the list of all the dormant *pixel-threads*. When a *pixel-thread*, which is dormant for some time, becomes active for a few frames, and then becomes dormant again, a flickering effect is observed. Thus, if such a *pixel-thread* is forced to be dormant instead of appearing for a short time, the flickering effect is considerably reduced. The history of activity and dormancy is maintained for each *pixel-thread* in each frame, while the frame is being encoded. Once the entire video has been encoded, a second pass is made to determine, for each *pixel-thread*, the frames in which the *pixel-thread* should be made dormant, using the algorithm described in **Figure 4.5**.

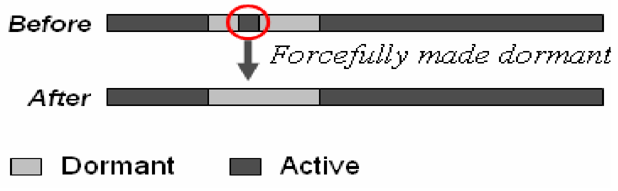


Figure 4.5. Flicker removal using the history of activity or dormancy of a *pixel-thread*. The encircled portion corresponds to a brief period of activity for the *pixel-thread*. The *pixel-thread* is made dormant to remove the flickering effect caused by this brief activity**.**

4.3. Encoding the texture - $V_{texture}$

In the previous section, we described in detail how a Generative Sketch-based Video (GSV) is obtained from the original video. The GSV is used as the *sketch* component within the proposed HLV representation. In this section, we describe how the second component of HLV, i.e., the *texture* component, is created. The texture of the video, given by the component $\mathbf{V_{TEXTURE}}$, consists of three sub-components termed as V_{org}, V_{mid} and V_{base} where V_{org} is the original video which is deemed to be of the highest visual quality, V_{mid} is the video of intermediate visual quality, and V_{base} is the base-level video of the lowest visual quality. All the above video sub-components are encoded using the MPEG H.264 standard. In this section, we will discuss in detail the procedure for generation of each of the three layers.

4.3.1. Generating the top-most video layer V_{org}

V_{org} is the original video which is encoded efficiently using a state-of-the-art MPEG H.264 encoder that is available in the public domain [Squared5]. A raw video encoded using the MPEG H.264 codec results in a very compact file representation. The MPEG H.264 codec uses inter-frame and intra-frame predictions to reduce significantly the spatial and temporal redundancy in the input video stream [Richardson, 2004], [ISO/IEC-visual, 2000].

4.3.2. Generating the intermediate video layer V_{mid}

The video layer V_{mid} represents an intermediate-level video which has a more compact representation than the original video albeit at the cost of lower visual quality. As will be shown in Section 4.5, a lower-size video file leads to reduction in overall power consumption during the decoding process. The video layer V_{mid} is generated using a novel multi-resolution video encoding technique termed as *Features, Motion and Object-Enhanced Multi-Resolution* (FMOE-MR) video encoding [Chattopadhyay, 2007a]. The FMOE-MR video encoding scheme is based on the fundamental observation that applying a low pass filter in the image color space is equivalent to DCT coefficient truncation in the corresponding DCT (frequency) space [Geusebroek, 2001]. The FMOE-MR video encoding scheme is a two step process as described in the following subsections.

4.3.2.1. Generating the FMO-Mask

Instances of Features, Motion and Objects (FMOs) are detected in the video sequence using state-of-the-art computer vision algorithms. A corresponding mask (FMO-Mask) is created to mark the regions corresponding to the presence of the FMOs. The mask contains floating point values between (and inclusive of) 0 and 1, where 0 represents a completely uninteresting region and 1 represents a region that is vital for visual and semantic understanding of the image. The FMO-Mask is essentially a combination of one or more of the following three individual masks; the Feature-Mask (F-Mask), Motion-Mask (M-Mask) and the Object-Mask (O-Mask).

Feature mask (F-mask)

The F-Mask captures the important low-level spatial features of the video frame. Edges are one of the most important low-level features of an image (or video frame), since human perception tends to first detect edges for the purpose of object recognition and general scene analysis. Edges can be detected automatically in a given image or video frame. There are many ways to perform edge detection. However, the majority of different methods may be grouped into two broad categories: gradient-based and Laplacian-based. The gradient-based methods detect the edges by seeking a maximum in the magnitude of the first derivative of the image intensity (or color) function. The Laplacian-based methods, on the other hand, search for zero crossings in the second derivative of the image intensity (or color) function to detect and localize the edges. In our current work we have used the Canny edge detector [Canny, 1986] which is a gradient-based method to detect and localize the edges in a video frame. Once the edges in the video frame are detected and localized using the Canny edge detector, an F-mask is created by assigning a value of 1 to regions in and around the edges, and the value of 0 elsewhere. Note that the mask is essentially a weighting matrix, where each pixel may be assigned a value between (and inclusive of), 0 and 1.

Motion mask (M-mask)

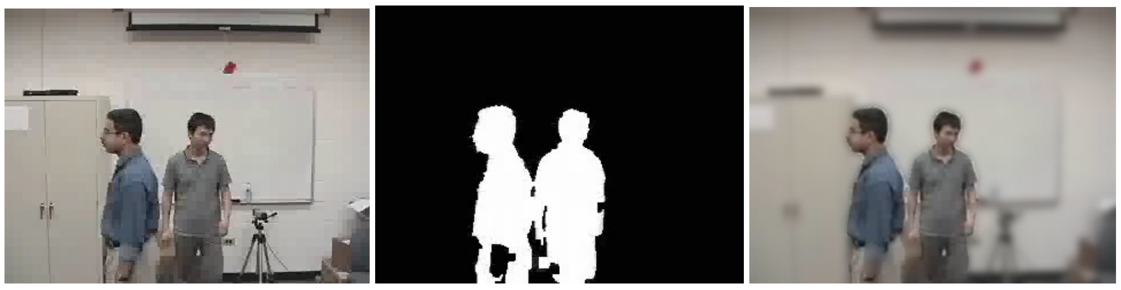
The motion within a video sequence constitutes a very important visual phenomenon since human perception of a dynamic scene tends to follow the moving objects and note their activities. Therefore, in situations which demand reduction in quality of the video, the image regions in the video that are characterized by significant motion can be rendered at high resolution and the remainder of the video frame at low resolution. A Motion mask (M-mask) is obtained by identifying the regions within the video frames that contain moving objects. This is essentially accomplished via a process of background subtraction [Cheung, 2006] [Javed, 2002] [Ivanov, 2002] [Luo, 2006]. Background subtraction is performed typically by first creating (or learning) a background model for a video sequence. The video frames are then compared with the background model to detect regions which are not part of background. These regions are classified as belonging to the dynamic foreground, i.e., containing moving objects. Background subtraction thus allows one to extract foreground objects which are moving relative to the camera, or had been moving until recently. We used the background models described in [Luo, 2005], [Luo, 2006] to extract the dynamic foreground (i.e., the moving objects) from the background.

Object mask (O-mask)

All the foreground objects in a video sequence may not be equally important from a visual or semantic perspective. For example, in a video sequence containing a news reader with a rotating background logo, the face of the news reader is more important than the moving logo which, in the current implementation, is also considered part of the foreground. In this case, the face is deemed an object of interest amongst the various foreground regions. Face detection and tracking is typically done by extracting feature points that characterize a face, and tracking these feature points in the video sequence. A detailed description of the face recognition and tracking algorithm is once again beyond the scope of this chapter. In our current implementation, faces in a video sequence are detected using the algorithms described in [Zarit, 1999], [Viola, 2002] and tracked using the algorithm described in [Luo, 2007] to create an O-mask based on human faces automatically.

Combining F, M, O masks to form a single FMO Mask

The three masks are superimposed to generate the final FMO mask. It is not always required to generate all the masks; for example, for a surveillance scenario, only the motion mask is required to capture the moving persons. Creation of the mask also



(a) (b) (c)

Figure 4.6. Using FMO-Mask for multi-resolution video frame re-rendering. (a) The original frame; (b) The FMO mask frame; (c) The frame re-rendered using FMOE-MR video encoding. The moving objects are rendered at high resolution whereas the background is rendered at low resolution. The obtained PSNR is 24.94

depends on the computational resources available; the F-and-M-mask creation can be typically done in real time, and combined in real time. For lower available power resources at the coding end, the F-mask only can be used; the edge detection by Canny is a low computational complexity task.

4.3.2.2. Multi-Resolution (MR) re-encoding

The original frame is re-encoded as a multi-resolution (MR) representation, guided by the FMO-mask such that regions corresponding to mask values close to 1 are at higher resolution than regions corresponding to mask values close to 0. The original video frame V_O is used to render two video frames, V_H and V_L, such that V_H is a high resolution rendering and V_L is a low resolution rendering of V_O. The video frames V_L and V_H are obtained by convolving V_O with Gaussian filters characterized by the smoothing parameters σ_L and σ_H respectively. Maintaining $\sigma_L > \sigma_H$ ensures that V_L is smoother than V_H, i.e., V_L is a lower resolution rendering of V_O than V_H. If the FMO mask is represented as a matrix **W** whose elements lie in the range [0, 1], then the MR frame V_{MR} is obtained via a linear combination of the two frames V_H and V_L as follows:

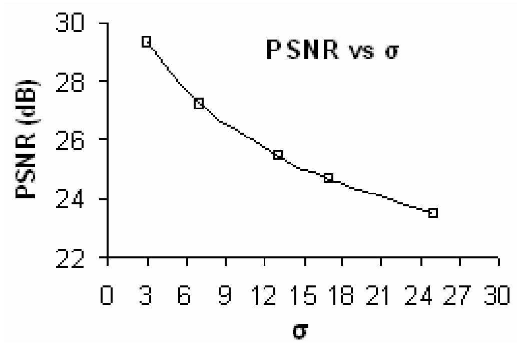
$$V_{MR} = \mathbf{W} \bullet V_H + (\mathbf{I} - \mathbf{W}) \bullet V_L$$

where **I** is a matrix all of whose elements are 1.

The values of σ_L and σ_H used to generate $V_{mid} = V_{MR}$ are selected empirically by the user. Empirical observations have revealed that $\sigma_L = 9$ or 11, and $\sigma_H = 3$, can be used, in most cases, to yield videos of reasonable visual quality with significantly smaller file sizes than the original video. **Figure 4.6** presents an example of a video frame where the foreground moving object has been extracted using an FMO-mask. As is apparent, the regions where the moving objects are situated are rendered at higher resolution compared to the stationary regions comprising the background. Since an exhaustive treatment of the FMOE-MR video encoding schemes is beyond the scope of this chapter, the interested reader is referred to [Chattopadhyay, 2007a] for further details. It must be noted that finally, V_{mid} too is encoded using standard MPEG H.264, after preprocessing using FMOE-MR.

4.3.3. Generating the base video layer V_{base}

The base video layer V_{base} is generated by first blurring each frame of the video using a Gaussian filter with smoothing parameter σ_{base} *prior* to MPEG H.264 encoding. Note that this is similar to the Gaussian smoothing performed in the case of FMOE-MR video encoding. The primary difference is that, in the case of the base video layer generation procedure, the smoothing is performed uniformly over the entire video frame in contrast to FMOE-MR video encoding where the extent of smoothing can vary within a video



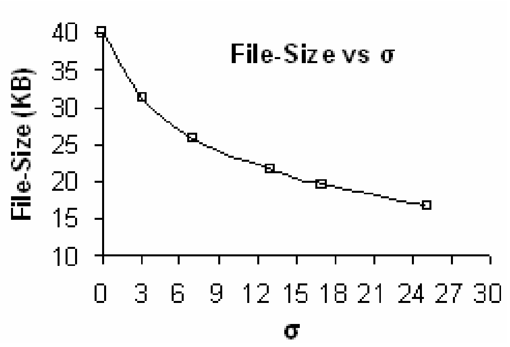


Figure 4.7. The change in PSNR and video file size as a function of the Gaussian smoothing parameter σ_{base} for the video.

frame based on the perceptual significance of the region under consideration. This results in further dramatic decrease in the file size upon MPEG H.264 encoding, albeit at the loss of video quality. Note that V_{base} is at of much lower visual quality than V_{mid} since object-based enhancement is not used. V_{base} essentially serves to provide approximate color information for the Generative Sketch-based Video (GSV) representation described previously.

4.3.4. Assessment of visual quality of $V_{TEXTURE}$

The visual quality of each of the aforementioned video layers comprising $\mathbf{V_{TEXTURE}}$ can be assessed in terms of PSNR values, as well as via subjective visual evaluation. A quantitative evaluation of the average PSNR of a sample video with respect to the percentage decrease in video size is depicted in **Figure 4.7**. It is apparent that the video size can be decreased significantly by using a high value of σ_{base}, albeit with a loss in video quality. We have observed empirically that values of σ_{base} in the range {19, 25} can be used to generate the base video layer V_{base}, resulting in a very small file size albeit at the cost of low resolution and low visual quality. However, approximate color information is still retained in the video layer V_{base}, to the point that the visual quality of

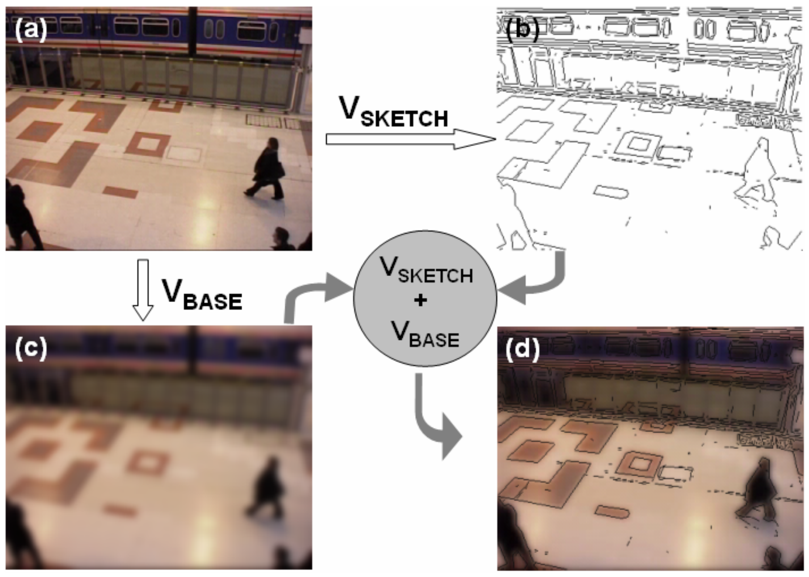


Figure 4.8. Super-imposing V_{SKETCH} and V_{BASE}.

the resulting video improves significantly when the object outlines from the GSV representation are superimposed on the video layer V_{base}.

4.4. Combining V_{sketch} and $V_{texture}$: Video states

As mentioned in the previous sections, the two components, $\mathbf{V_{TEXTURE}}$ and $\mathbf{V_{SKETCH}}$, are obtained independently of each other. First, a suitable texture frame is extracted from the $\mathbf{V_{TEXTURE}}$ component of the video by the video display controller. After this frame has been written to the frame buffer, the other component, $\mathbf{V_{SKETCH}}$, is used to superimpose the object outlines on the frame buffer containing $\mathbf{V_{TEXTURE}}$. Both events are independent in terms of processing; they are only related by order, i.e., $\mathbf{V_{TEXTURE}}$ is rendered first, followed by the superimposition of $\mathbf{V_{SKETCH}}$ on $\mathbf{V_{TEXTURE}}$. An example frame obtained by superimposing $\mathbf{V_{SKETCH}}$ on the V_{base} subcomponent of $\mathbf{V_{TEXTURE}}$ is shown in **Figure 4.8**.

Let us suppose that the $\mathbf{V_{TEXTURE}}$ component has L levels of resolution. In the current implementation, $L = 4$ which includes the three layers V_{org}, V_{mid} and V_{base} in decreasing order of visual quality and level 0 which denotes complete absence of texture information. Let $\mathbf{V^{j}_{TEXTURE}}$ $(0 \leq j \leq L\text{-}1)$ correspond to the MPEG H.264-based encoding of the video where $\mathbf{V^{0}_{TEXTURE}}$ denotes the complete absence of texture information, $\mathbf{V^{1}_{TEXTURE}}$ denotes the video layer of the least visual quality and resolution (with deliberately induced loss in visual quality to ensure a very low bitrate and small video file size), and $\mathbf{V^{L\text{-}1}_{TEXTURE}}$ denotes the video layer of the highest visual quality and resolution (i.e., the original video encoded using the MPEG H.264 standard with no deliberately induced loss in visual quality). Let the state of the HLV-encoded video be depicted as

$$\mathbf{\Gamma}(\textit{texture-level}, \textit{sketch-level}) = (\mathbf{V^{texture\text{-}level}_{TEXTURE}}, \mathbf{V^{sketch\text{-}level}_{SKETCH}}) \qquad (4.7)$$

such that $0 \leq$ *texture-level* $\leq L$-1, and *sketch-level* Є {*no-sketch*, *polyline-sketch*, *spline-sketch*}. The above state-based representation allows for various resolutions of texture with superimposition of sketch-based representations of varying degrees of complexity. Under the above formalism, $\mathbf{\Gamma}(L$, *no-sketch*) represents the original (i.e., best quality) video, and $\mathbf{\Gamma}(0$, *polyline-sketch*) represents the video that contains no texture, but only the object outlines represented by polylines (presumably, the lowest quality video).

Furthermore, the states in the above representation are linearly ordered such that a "higher" video state is deemed to consume more power than a "lower" video state. Thus, it is essential to order the different states of the video in the above representation in terms of their battery power usage. Let *Battery-Time*($\mathbf{X}$, t) be the battery time estimate provided by the operating system on the playback device t seconds after the video playback has been initiated, where **X** denotes the state of the video during playback. Let $\mathbf{\Gamma} = \{\mathbf{\Gamma_1}, \ldots, \mathbf{\Gamma_S}\}$ be the *S* distinct video states. We define a relation $\leq_p$ such that

$$\mathbf{\Gamma}_i \leq_p \mathbf{\Gamma}_j \text{ implies that } \textit{Battery-Time}(\mathbf{\Gamma}_i, t) \geq \textit{Battery-Time}(\mathbf{\Gamma}_j, t),\ t > 0 \qquad (4.8)$$

In other words, the states are linearly ordered from left to right such that for any state (except for the states $\mathbf{\Gamma}(L$, *no-sketch*) and $\mathbf{\Gamma}(0$, *polyline-sketch*)), the state on its left consumes less power while decoding the entire video whereas the state on its right consumes more power. The value *Battery-Time*($\mathbf{\Gamma}_{\text{current-state}}$, *t*) estimated using a simple operating systems call, which predicts the remaining battery time based on the current system load. In the following section, we present and analyze experimental results in order to validate the claim that the proposed HLV representation does indeed result in different power consumption estimates for distinct states in the aforementioned video state-based representation.

4.5. Experimental results and analysis

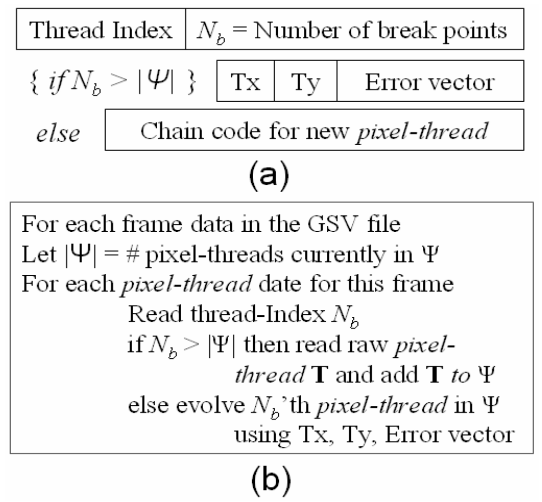
We have conducted experiments using two video examples to measure various aspects of power consumption, such as number of CPU cycles needed to decode the video, and the battery life of the device during video playback. We first describe how the various components of the proposed HLV representation are implemented. Next, we provide estimates of the sizes of the files used to store the various video states on hard disk. Finally, we show, using the *Train Station* video as an example, that the various video layers of the proposed HLV representation do indeed consume different amounts of power. This clearly demonstrates that the proposed Hybrid Layered Video (HLV) representation is indeed one that is power-scalable.

4.5.1. Implementing the video layers

As discussed previously, implementing the texture layers is straightforward. We used the Gaussian Smoothing filter with smoothing parameter $\sigma = 0$ (i.e., no smoothing) and $\sigma = 15$ to encode the video layers V_{org} and V_{base} respectively. Likewise, the values $\sigma_L = 9$ and $\sigma_H = 3$ were chosen as smoothing parameters for the Gaussian Smoothing filters associated with the FMOE-MR encoded video V_{mid} (Section 4.3.2). The values of σ in each case were selected empirically. Together, the sub-components V_{org}, V_{mid} and V_{base} constitute the $\mathbf{V_{TEXTURE}}$ component in the proposed HLV representation.

The object outlines generated by the GSV file are encoded using a GOP size of 15 frames in compliance with the MPEG H.264 standard, i.e., the 1st frame is encoded as I frame; whereas the remaining 14 frames are encoded as P-frames. The *pixel-threads* corresponding to the first frame (or I-frame) are stored as '*born*' pixel-threads. This means that each pixel-thread is represented as a chain-coded vector as discussed in Section 4.2.1.2. A chain-coded *pixel-thread* requires the first break point to be specified in terms of its absolute X and Y-coordinates with respect to the origin of the frame. Since typical video resolutions for mobile devices do not exceed the standard XGA video resolution (1024×768), 4 bytes are more than adequate to store the absolute X, Y positions of the first break point (note that 2 bytes for each coordinate value can represent a frame of size 65536×65536 pixels). The remaining break points require computation of displacements relative to the previous break point. These displacements can be restricted to lie between -127 and 128 (a character); thus 2 bytes are required for each of the remaining break points. Note that if a break point is too distant from the previous break point for its coordinates to be represented by two bytes, then an additional dummy break point can be inserted so that the resulting displacements can be represented using 2 bytes.

From the second frame onwards, some of the *pixel-threads* in the current frame are evolved from *pixel-threads* in the previous frame, whereas some *pixel-threads* are born in the current frame. This situation is similar to the motion compensation used by a typical MPEG in the case of P-frames. The pixel-threads which are deemed to be born are represented using the chain-coding scheme described above. For an evolved *pixel-thread*, the index of the *pixel-thread* from which it has evolved, and the corresponding motion model parameters need to be represented.


Thread Index
N_b = Number of break points
{ *if* $N_b > |\Psi|$ }
Tx
Ty
Error vector
else
Chain code for new *pixel-thread*
(a)
(b)

```
For each frame data in the GSV file
Let $|\Psi|$ = # pixel-threads currently in $\Psi$
For each pixel-thread date for this frame
        Read thread-Index $N_b$
        if $N_b > |\Psi|$ then read raw pixel-
                thread $\mathbf{T}$ and add $\mathbf{T}$ to $\Psi$
        else evolve $N_b$'th pixel-thread in $\Psi$
                using Tx, Ty, Error vector
```

Figure 4.9. The GSV file (a) The format used to encode a *pixel-thread*; (b) The algorithm used to read a GSV-encoded video file

Instead of describing whether a *pixel-thread* is born in the current frame, or evolved from a previous thread, it is sufficient to mention just the index of the *pixel-thread.* Based on the definition of the index of a *pixel-thread*, if the index of the *pixel-thread* under consideration is greater than the number of *pixel-threads* in **Ψ** (Section 4.2.3), then this *pixel-thread* is considered to have been be born in the current frame. Thus, subsequent information on this *pixel-thread* in the GSV file pertains to the chain-coded representation of its break points. On the other hand, if the index of the *pixel-thread* under consideration is less than the number of *pixel-threads* in **Ψ**, then it is

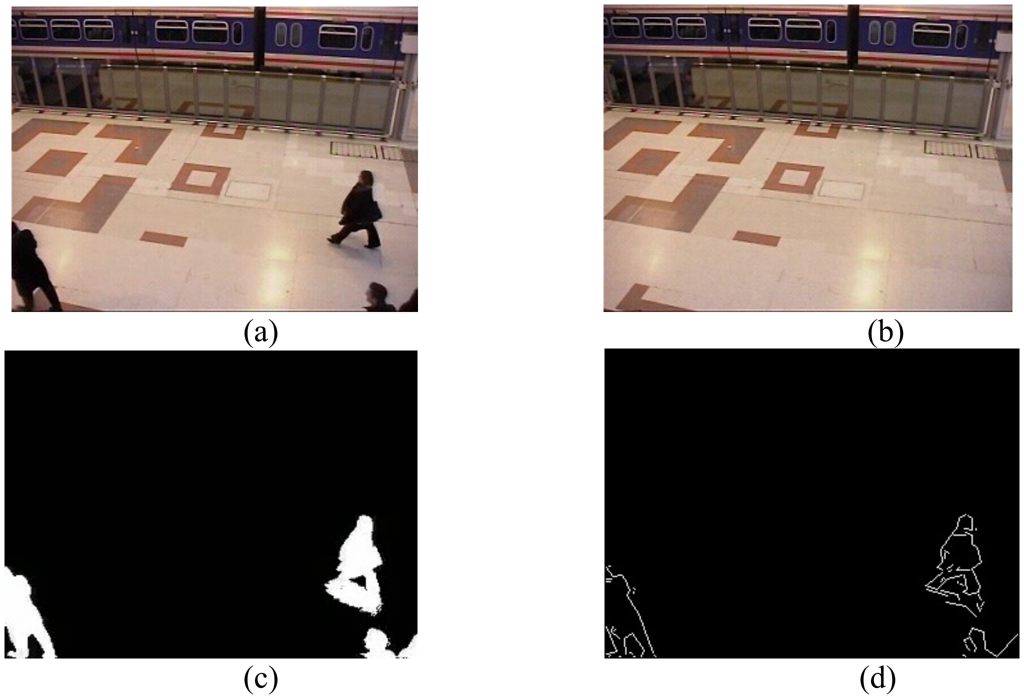


Figure 4.10. GSV encoding of foreground objects (a) The original video frame; (b) The background model learned from the first few frames of the video sequence; (c) The foreground mask obtained after background subtraction; (d) The portion corresponding to the foreground mask is encoded as a sketch.

deemed to have evolved from some *pixel-thread* in **Ψ**. Consequently, subsequent information on this *pixel-thread* in the GSV file pertains to the parameters of the motion model describing its evolution. The GSV file format is depicted in **Figure 4.9a** and the algorithm used to read and interpret the GSV file is described in **Figure 4.9b**.

We used a motion mask obtained using the background modeling and background subtraction algorithms described in [Luo, 2005], [Luo, 2006] to extract the moving foreground objects in the *Train Station* video sequence. Only the moving foreground objects in the *Train Station* video sequence are encoded in the resulting GSV file [Chattopadhyay, 2007a]. A sample video frame shown in **Figure 4.10** depicts the result of the extraction and encoding of the moving foreground objects. The result is an extremely compact representation, where the background is represented as a single parametric image, and the GSV file that encodes only the moving foreground objects in the 17-minute *Train Station* video sequence occupies only 6 MBytes of hard disk space. The same motion mask is then used to extract the motion-containing regions that are encoded at high spatial resolution in the V_{mid} video layer.

A synopsis of the resulting file sizes of the various components and sub-components of the video is as follows: V_{org} (95 MBytes), V_{mid} (35 MBytes), V_{base} (13 MBytes) and V_{sketch} (8 MBytes, where the background is encoded as a single GSV frame). As expected, the lower the visual quality of the video, the smaller the file size. The combined size of all the components and subcomponents of $\mathbf{V_{TEXTURE}}$ and $\mathbf{V_{SKETCH}}$ is 151 MBytes whereas the original video alone is 95 MBytes. Thus, the proposed power-scalable HLV encoding of the *Train Station* video resulted in a data storage overhead of approximately 56% compared to the flat file representation of the original video. Given the abundance and low price of data storage, data storage overhead of about 56% could be deemed acceptable for most video servers that specialize in streaming video files to mobile devices.

4.5.2. Visual quality of different video states

Based on the discussions in the previous section it can be seen that the various video layers (or video states) in the proposed HLV encoding scheme are generated by altering (or reducing) the quality of the resulting encoded video. In the next subsection, we show that the power consumed during the decoding process can also be made to vary significantly based on the chosen video layer or video state. However, it is important to

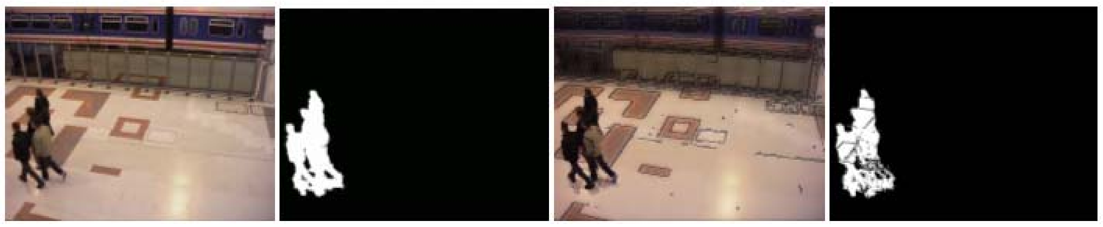


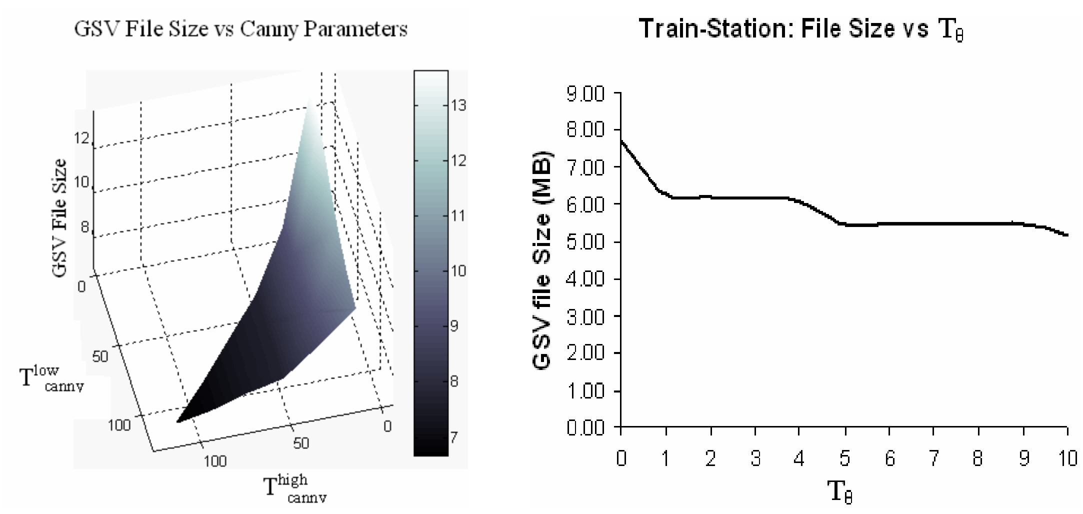
Figure 4.11. Objective comparison of the video states $\Gamma_3^{\text{train}} = \Gamma(V_{\text{base}}, \textit{polyline-sketch})$, and $\Gamma_6^{\text{train}} = \Gamma(V_{\text{org}}, \textit{null})$ (a) A video frame from the video state $\Gamma_6^{\text{train}} = \Gamma(V_{\text{org}}, \textit{null})$; (b) The foreground mask generated using background subtraction on the video frame in (a); (c) The corresponding video frame for the video state $\Gamma_3^{\text{train}} = \Gamma(V_{\text{base}}, \textit{polyline-sketch})$; (d) The foreground mask generated using background subtraction on the video frame in (c). The foreground masks in (b) and (d) are seen to overlap by more than 85%.

note that a video state in the proposed HLV encoding scheme is only an approximation to the original MPEG-encoded video; which raises the natural question, i.e., is the approximation good enough? Subjective evidence gathered from various students and co-workers has revealed that all the objects discernable in the MPEG-encoded video are also discernable in the HLV-encoded video. A comprehensive quality assessment of the proposed HLV encoding scheme, which includes a subjective end-user survey, is currently underway.

As an objective comparison of HLV-encoded video quality, we used the videos corresponding to video states $\mathbf{\Gamma_3^{train}} = \mathbf{\Gamma}(V_{\text{base}}, \textit{polyline-sketch})$, and $\mathbf{\Gamma_6^{train}} = \mathbf{\Gamma}(V_{\text{org}}, \textit{null})$, to perform some standard computer vision tasks such as background subtraction. Note that $\mathbf{\Gamma_6^{train}} = (V_{\text{org}}, \textit{null})$, in our case, corresponds to the video with the highest visual quality. As described previously, background subtraction is the process of first estimating the background of the dynamic scene, where the background is deemed to comprise of those regions within the video frames which do not move relative to the camera. The background, thus determined, is subtracted from each frame to extract the foreground, or moving regions within the video frames [Davies, 1990]. We hypothesize that the video states $\mathbf{\Gamma_3}^{\mathbf{train}}$ and $\mathbf{\Gamma_6}^{\mathbf{train}}$ yield videos of comparable quality if both videos result in similar foreground regions upon background subtraction.

Figure 4.11 shows the resulting foreground masks after background subtraction has been performed on a $\mathbf{\Gamma_6}^{\mathbf{train}}$ video frame and the corresponding $\mathbf{\Gamma_3}^{\mathbf{train}}$ frame. As is evident from **Figure 4.11**, both videos were observed to yield similar foreground masks. We further computed the percentage overlap between the foreground masks generated from the two videos. The mask generated from the $\mathbf{\Gamma_3}^{\mathbf{train}}$ video frame was observed to have an 85% overlap with the mask generated from the original video, $\mathbf{\Gamma_6}^{\mathbf{train}}$. Other vision-based tasks such as face detection using skin tone yielded similar results since the skin tones were observed to be well preserved in the texture component of the $\mathbf{\Gamma_3}^{\mathbf{train}}$ video [Chattopadhyay, 2007a]. Note that we could not use standard metrics such as the Peak Signal-to-Noise Ratio (PSNR) to measure quality of HLV-encoded video, since the PSNR measure treats the graphics overlay of outlines as noise.

It must be noted that for video examples with very detailed images, the lower layers of the proposed HLV encoding are often not acceptable, as they cannot capture all the fine details present in the original image. But for most typical applications that require limited attention to detail (such as wide-area surveillance, wide-area traffic monitoring and some sports and news coverage), the lower layers of the HLV encoding can be used as an alternative to the original video (or the highest layer of the HLV-encoded video). The lower layers of the HLV-encoded video are observed to consume significantly less power than the original video during the decoding process, as shown in the next subsection.



(a) (b)

Figure 4.12. GSV file size vs V_{SKETCH} parameters (a) T^{low}_{canny} and T^{high}_{canny}; (b) T_θ.

4.5.3. Analysis of power consumption of different video states

Before comparing power consumption for the different states of the HLV-encoded video, we first examine the various parameters affecting the power consumption of the HLV-encoded video. In the following subsections we discuss the impact of each of these parameters on the overall power consumption profile of the HLV-encoded video. Finally, we discuss objective evaluation measures that demonstrate that distinct video states do indeed exhibit different power saving trends.

4.5.3.1 Parameters for V_{SKETCH}

The first step in the creation of the $\mathbf{V_{SKETCH}}$ component is the generation of the initial pixel-based sketch of the video frame using the Canny edge detector [Canny, 1986]. The Canny edge detector uses two thresholds, T^{low}_{canny} and T^{high}_{canny}. The range [T^{low}_{canny}, T^{high}_{canny}] determines the number of edges that are detected and the resulting continuity

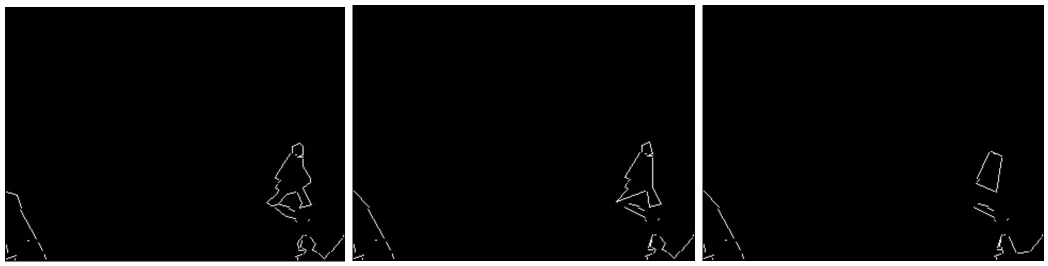

(a) (b) (c)

Figure 4.13. Visual quality vs T_θ (a) $T_\theta = 0^o$ (i.e., the original frame); (b) $T_\theta = 5^o$; (c) $T_\theta = 10^o$

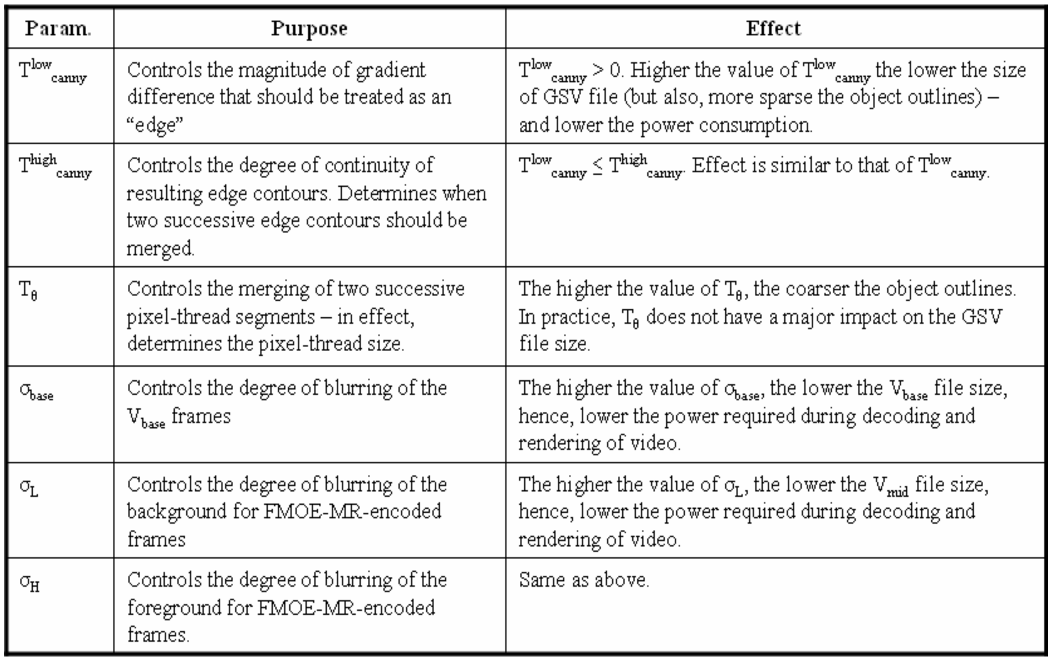
of the edge contours. If T^{high}_{canny} is set too high, important edge information could be missed and the resulting edge contours would be discontinuous. On the other hand, if T^{low}_{canny} is set too low, a large number of false (noisy) edge pixels will be detected. The choice of threshold values can directly affect the power consumption during the decoding of the GSV-encoded video. This is so because an encoding that entails fewer *pixel-threads* would require fewer CPU cycles for the purpose of decoding. As will be shown in Section 4.5.4.3., the power consumed during decoding of the GSV-encoded file is directly proportional to its size. **Figure 4.12a** shows a plot of the GSV file size as a function of the Canny edge detector parameters.

In the second part of the algorithm used to generate the $\mathbf{V_{SKETCH}}$ component, break points are generated for each edge contour using the algorithm of Rosin and West [Rosin, 1995]. The result is a set of non-collinear break points in 2D, where the line segments connecting two consecutive break points are termed as *pixel-segments*. For any two consecutive *pixel-segments* one can define a *segment-angle* θ between them. In order to reduce the number of break points, and hence the number of bytes required to store each *pixel-thread*, consecutive *pixel-segments* with a *segment-angle* less than T_θ can be joined to form a single *pixel-segment*, thus eliminating a break point on the *pixel-thread*.

This results in a reduction in the size of the GSV file. **Figure 4.12b** shows a plot of the GSV file size as a function of T_θ. However, even a low value of T_θ, say 2^o, results in a visually discernable deterioration in image quality. **Figure 4.13** depicts the effect of the choice of threshold value T_θ on the resulting *pixel-threads*. As shown in **Figure 4.13**, a small value of T_θ can alter the shapes of the objects within the video dramatically. As a result, in our encoding experiments, we have not used a threshold to reduce the number of break points in the individual *pixel-threads*.

We have so far demonstrated the effect of the three aforementioned threshold parameters, on the file size of the GSV-encoded video. As shown in Section 5.4.3, the size of the GSV-encoded file, in turn, is shown to determine the power consumed during the video decoding process.

Table 4.1. List of all the parameters affecting HLV power consumption.



Param.	Purpose	Effect
T^{low}_{canny}	Controls the magnitude of gradient difference that should be treated as an "edge"	$T^{low}_{canny} > 0$. Higher the value of T^{low}_{canny} the lower the size of GSV file (but also, more sparse the object outlines) – and lower the power consumption.
T^{high}_{canny}	Controls the degree of continuity of resulting edge contours. Determines when two successive edge contours should be merged.	$T^{low}_{canny} \leq T^{high}_{canny}$. Effect is similar to that of T^{low}_{canny}.
T_θ	Controls the merging of two successive pixel-thread segments – in effect, determines the pixel-thread size.	The higher the value of T_θ, the coarser the object outlines. In practice, T_θ does not have a major impact on the GSV file size.
σ_{base}	Controls the degree of blurring of the V_{base} frames	The higher the value of σ_{base}, the lower the V_{base} file size, hence, lower the power required during decoding and rendering of video.
σ_L	Controls the degree of blurring of the background for FMOE-MR-encoded frames	The higher the value of σ_L, the lower the V_{mid} file size, hence, lower the power required during decoding and rendering of video.
σ_H	Controls the degree of blurring of the foreground for FMOE-MR-encoded frames.	Same as above.

4.5.3.2. Parameters for $V_{TEXTURE}$

As discussed in Section 4.3.2, the V_{mid} video layer is encoded using the proposed FMOE-MR encoding technique. The FMOE-MR encoding technique, in turn, depends on the two Gaussian smoothing parameters; σ_L and σ_H. As discussed previously, we have empirically determined the values of these parameters to be $\sigma_L = 9$ or 11, and $\sigma_H = 3$ or 5. The video layer V_{base} is created via uniform blurring of the video frame using a Gaussian filter with smoothing parameter σ_{base}**.** Recall that **Figure 4.7** shows that varying the value of the Gaussian smoothing parameter σ_{base} results in a dramatic decrease in the file size. The CPU power consumption during decoding of the texture layer V_{mid}, is shown to be approximately proportional to the size of the resulting file, as will be shown in the next subsection. **Table 4.1** summarizes the various parameters used in the generation of video layers V_{mid} and V_{base} and their impact on the overall power consumption in the device on which the video is being decoded and rendered.

4.5.3.3. Power consumption estimate for various states

Since the CPU actually contributes about 15% to the overall power consumed in a mobile device during the process of video playback (comprising of downloading, decoding and rendering of the video) [Hennessy, 2002], it is important to measure the overall power consumption for the entire system rather than the CPU power consumption alone. As a result, we compare the overall power consumption for the various video states by measuring the time-remaining statistics during the process of video playback in each case. We have used the following, empirically determined, parameter values: $T^{low}_{canny} = 60$, $T^{high}_{canny} = 70$, $T_\theta = 0$, $\sigma_L = 3$, $\sigma_H = 9$ and $\sigma_{base} = 21$. The experiments have been

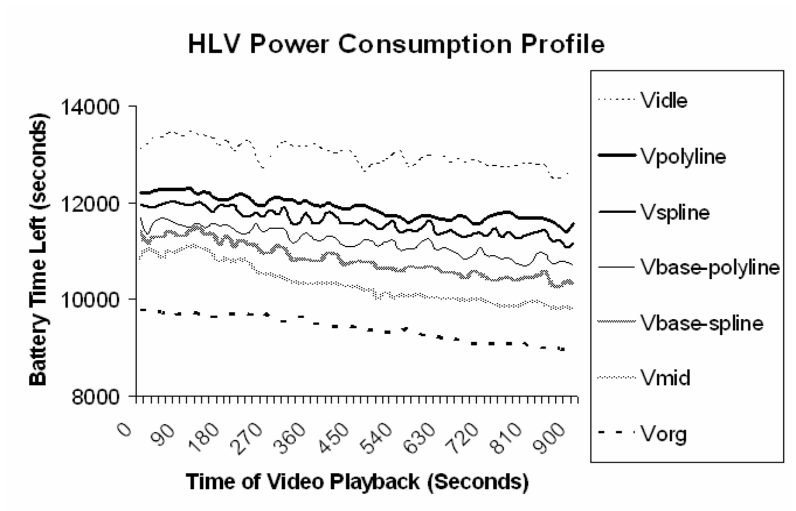


Figure 4.14. Power consumption profile for the various video states of the *Train Station* video. The dotted line at the very top represents the idle state; i.e., when the device is idle and not running any video.

conducted using a laptop PC with a 2 GHz CPU, 2 MBytes L2 Cache, 1 GByte RAM and 60 GBytes, 5400 rpm hard drive running in battery mode.

From **Figure 4.14**, it is apparent that lower states consume less power than the higher states in the proposed HLV representation. It is also apparent that the file size on hard disk (regardless of whether it is a GSV file or a texture file) for each of the video states directly affects the overall power consumption during video playback. We have compared the mean difference between the *battery time remaining* statistics of the various video states. The mean difference between the battery time remaining statistics for the state $\mathbf{\Gamma}$(null, *spline-sketch*) and the state $\mathbf{\Gamma}(V_{org}$, *null*) is ≈ 45 minutes. Thus, it is possible to have ≈ 45 minutes of extra system battery life if the video is viewed using the sketch-only version. Similarly, the difference in battery time remaining between the states $\mathbf{\Gamma}(V_{base}$, *spline-sketch*) and $\mathbf{\Gamma}(V_{org}$, *null*) state is ≈ 30 minutes. Likewise the

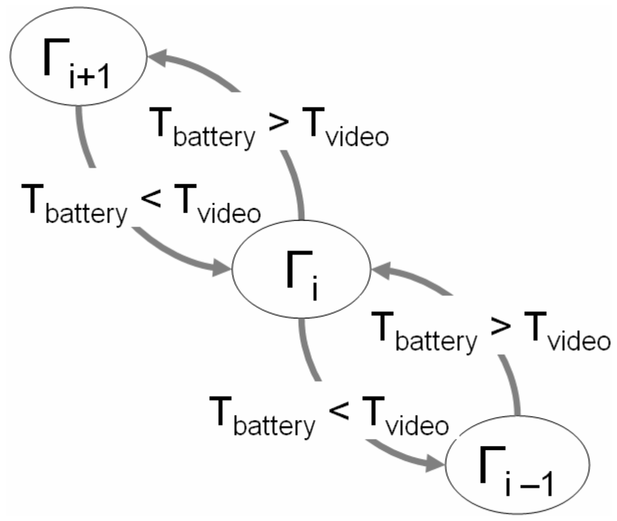

Γ_{i+1}
$T_{battery} > T_{video}$
$T_{battery} < T_{video}$
Γ_i
$T_{battery} > T_{video}$
$T_{battery} < T_{video}$
Γ_{i-1}

Figure 4.15. State diagram depicting the state transition rules. The current state transitions to a higher state if the available battery time ($T_{battery}$) is greater than the remaining running time of the video (T_{video}). Similarly, the current state transitions to a lower state if $T_{battery} < T_{video}$.

difference in battery time remaining between the states $\mathbf{\Gamma}(V_{mid}, \mathit{null})$ and $\mathbf{\Gamma}(V_{org}, \mathit{null})$ is ≈ 15 minutes.

4.5.3.4. Real Time Video State Selection

When the video playback is initiated on the client device, the video is displayed at the highest quality, corresponding to state $\mathbf{\Gamma}_6 = \mathbf{\Gamma}(V_{org}, \mathit{null})$. Suppose the video, which has a total running time of T_v seconds, has been playing for t seconds and the current video state is $\mathbf{\Gamma}_{current\text{-}state}$. Three situations may arise:

Battery-Time($\mathbf{\Gamma}_{current\text{-}state}$, t) > (T_v - t): This means that if the current state of the video is maintained, it can be viewed comfortably with current battery drainage rate computed by the system. In this case, a higher video state could be chosen to improve the visual experience of the end user.

Battery-Time($\boldsymbol{\Gamma}_{\text{current-state}}$, *t*) $\approx$ (T_v - *t*): This means that the remaining battery time is approximately equal to the remaining video duration. In this case, the current state of the video is maintained.

Battery-Time($\boldsymbol{\Gamma}_{\text{current-state}}$, *t*) < ($T_v$ - *t*): This means that the current state of the video is too power intensive for the system to handle and continued video playback in the current state runs the risk of draining the battery completely before the entire video can be viewed. In this case, the system transitions to a lower video state for the purpose of video playback, with the expectation the battery power usage rate will slow down. **Figure 4.15** depicts the rules that govern the change in video states.

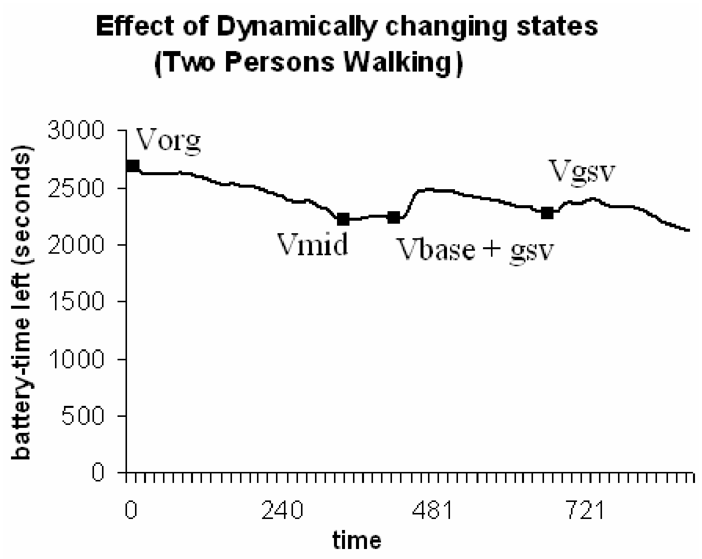


Figure 4.16. The effect of video state transitions on the remaining battery time. The video playback starts with Γ(V_{org}, *no-sketch*), changes state to Γ(V_{mid}, *no-sketch*) then to Γ(V_{base}, *spline-fit*) and finally to Γ(0, *spline-fit*). It is apparent that there is an improvement in remaining battery time every time a lower state is chosen.

As an additional example, consider another sample video *Two persons walking*. **Figure 4.16** shows how the remaining battery time changes dynamically with changing video states. From **Figure 4.16**, it is clear that the remaining life of a battery can be dynamically varied by transitioning between various video states during video playback.

From above results and analysis, we can derive the general relationship between the aforementioned video states:

$$\boldsymbol{\Gamma}_1^{\text{train}} \leq_p \boldsymbol{\Gamma}_2^{\text{train}} \leq_p \ldots. \boldsymbol{\Gamma}_5^{\text{train}} \leq_p \boldsymbol{\Gamma}_6^{\text{train}} \qquad (4.9)$$

Likewise, we can conclude that the proposed Hybrid Layered Video (HLV) representation can be used to deliver different "styles" of video with varying power consumption characteristics, to a heterogeneous ensemble of mobile devices with different power resource constraints.

4.6. Conclusions and future directions

We have proposed a novel layered video representation scheme, termed as Hybrid Layered Video (HLV), where distinct video layers display different battery power consumption profiles during video playback on a mobile device operating in battery mode. The video is divided into two components – a texture-based component and a sketch-based component. The texture component of the video is further divided into three sub-components. The first sub-component is the original video that is encoded efficiently using a state-of-the-art MPEG H.264 encoder. The second sub-component, which is of lower visual quality, and consumes less power, is visually enhanced using a Features, Motion and Object Enhanced Multi Resolution (FMOE-MR) video encoding scheme. The third sub-component is of very low visual quality, containing enough information to barely display the various colors and textures in the video, but devoid of the shapes of the object outlines. This sub-component is visually enhanced by artificially overlaying outlines of the objects within the video to add further definition to the object boundaries. A novel Generative Sketch-based Video (GSV) representation scheme has been implemented to encode the object outlines while reducing the spatial and temporal redundancy for maximum efficiency. The GSV encoding scheme provides the sketch component of the proposed HLV encoding scheme. Objective evaluation measurements have revealed that it is possible to increase the battery life of a laptop computer by approximately 45 minutes by using the lower-most layer of the video. The experimental results have led to the conclusion that the proposed HLV encoding scheme is indeed well suited to disseminate video to mobile devices, with different power resource constraints (i.e., battery life specifications).

A very important future research direction is to create the different components such that they are incremental in nature; i.e., a higher level can be derived the previous lower level by a process of incremental enhancement. Further, since the proposed HLV scheme is inherently content-aware, more sophisticated MPEG-7 or MPEG-21 encoding paradigms can potentially be used. Integration of the proposed HLV scheme with the well-established MPEG standard is a challenging future task. Finally, due to the layered nature of HLV, efficient caching schemes can be developed to disseminate the video efficiently such that the latency experienced by the user end requesting the video is minimized.

CHAPTER 5

VIDEO CACHING FOR POWER CONSTRAINED DEVICES

5.1 Introduction

In order to improve performance of servers hosting video files, caching portions of server video data near the client end is a popular method. The proximity of the cache, compared to the server, results in lower latency of file download/streaming observed by the client. Using a cache also offloads computational and network load from the server, which is typically inundated with thousands of requests for video files. Each cache handles request for several clients, thus acting as a buffer between the client and the server.

In chapters 3 and 4, various multilayered, content aware, resource constraint adaptive video transcoding methods have been described. These transcoded video files are typically hosted in a video server. In order to reduce server load, as discussed above, it is required to cache these transcoded video files. Due to the layered nature of these transcoded video files, smart caching schemes can be developed, which exploit the inherent layered structure of the transcoded video files in order to efficiently cache the files.

In the next two sections, two novel video caching techniques have been discussed in details.

5.2. A Framework for Encoding and Caching of Video for Quality Adaptive Progressive Download

Progressive download of video over the Internet has become an increasingly popular alternative to multimedia streaming [Youtube]. Progressive download of video is similar to standard HTTP download of web content, except for the fact that video playback starts the moment a sufficient amount of video content has been downloaded. Most streaming servers can dynamically adapt the video bitrate to allow for continuous streaming in environments characterized by dynamically changing bandwidth. However, progressive download schemes often lack this bitrate adaptation capability since the video is typically encoded at a single bitrate. Moreover, caching of video for progressive download typically involves caching of the entire video such that the entire video is replaced during cache replacement. Thus, when the replaced video is requested by the client again, the entire video has to be fetched from the server, resulting in inefficient utilization of server and network resources. Although there is considerable published literature on caching of adaptive multimedia streams [Rejaie, 2000], [Kangasharju, 2001], the techniques described therein cannot be directly applied to progressive download of video

In this chapter, a framework for quality adaptive progressive download (QAPD) of video has been proposed. The proposed framework makes two major contributions:

- A novel layered video representation scheme, inspired by the MPEG Fine Grained Scalability (FGS) profile, which can be hosted simply on a standard HTTP web server.
- An efficient caching mechanism to significantly reduce the client-observed latency. In the proposed scheme, a simple proxy web server can perform the role of a proxy video server. Appropriate, quality metrics, that are specific to layered multimedia

representation, are proposed to assess the performance of the proposed QAPD caching framework.

The proposed QAPD framework, with caching, provides considerable advantage over quality adaptive FGS (QAFGS) streaming. First, QAPD files can be hosted simply on a cost-effective web server with fewer CPU and memory requirements than those of a streaming server in case of QAFGS files. Second, the load on the simple web server in case of QAPD files is considerably lower than that on a dedicated streaming server in the case of QAFGS files. This is so because QAPD is driven mainly by simultaneous client requests for downloading various multimedia files. Finally, the QAPD proxy cache can also be hosted on a simple web server, whereas QAFGS requires the proxies to also have streaming capabilities.

5.2.1. QAPD

The proposed Quality Adaptive Progressive Download (QAPD) of video can be achieved by designing a layered representation of the video, which can be hosted on a HTTP server, followed by a method to enable variable bitrate progressive download.

5.2.1.1. Layered Representation of video

Layered encoding for quality adaptive streaming is achieved by using the MPEG-FGS profile [Li, 2001], [Radha, 2001], which partitions the video file into two layers; the base layer and the enhancement layer. The proposed layered representation scheme is an adaptation of the MPEG-FGS profile. Layered encoding of a video file V is achieved by first dividing V temporally into Groups of Pictures (GOPs). For each GOP, the first frame

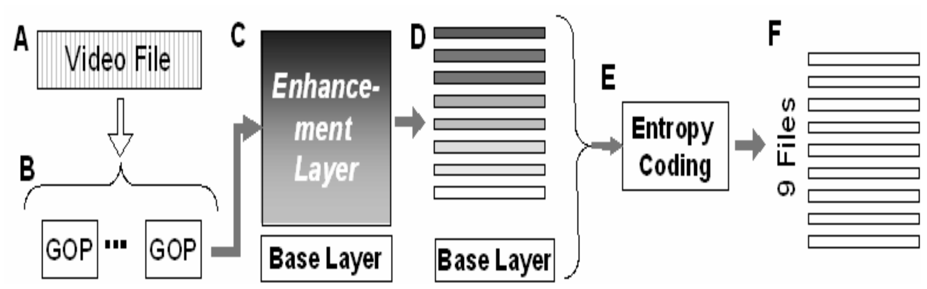


Figure 5.2.1. Quality adaptive Layered video creation

is the I-frame; and the remaining frames are encoded as P-frames or B-frames using motion prediction [Richardson, 2004]. Next, the DCT coefficients are computed for 8×8 blocks of the motion-predicted frames, followed by quantization of the coefficients, such that the quantized DCT coefficients can be represented using 8 bits. The base layer for the layered video representation can be generated by smoothing the frames in their respective color spaces, or equivalently, truncating the corresponding DCT coefficients [Gonzalez, 1992]. The DCT-residue layer is computed by subtracting the truncated and quantized DCT coefficients of the base layer from the quantized DCT coefficients of the original frame. The DCT-residue layer is used to generate the enhancement layers. The first enhancement layer is generated by taking the most significant bit of the quantized residue of each DCT coefficient from the DCT-residue layer; the second layer generated by taking the second-most significant bit, and so on. In all, 8 enhancement layers can be generated from the DCT-residue layer, which, in conjunction with the base layer, constitute the best quality image. Finally, the base layer and the 8 enhancement layers are subject to lossless entropy coding (run length coding followed by arithmetic coding [Richardson, 2004]) to generate the final, compressed set of 9 files, corresponding to a single multimedia file, as shown in **Figure 5.2.1**.

5.2.1.2 Layered multimedia-enabled QAPD

Using the above layered representation technique, a video file *f* is represented by 9 files, *f.0, f.1, f.2, ..., f.8.* When a client sends a request for file *f* to the server, it essentially sends a request for the nine files *f.0, ..., f.8.* The moment a GOP of the base layer is received, the client can commence video playback at the base quality. If the GOPs corresponding to the higher enhancement layers are also received, then a correspondingly higher quality video can be displayed simultaneously. Note that the proposed QAPD scheme entails parallel requests from the client which is in contrast to server-initiated variable bitrate streaming via dynamic rate adaptation, as is typically done in the case of MPEG-FGS. QAPD is thus essentially a client initiated technique.

5.2.2. Layered video caching

The success of progressive download is heavily dependent on ensuring low client-experienced latency. Thus it is desirable to cache portions of these files in the proximity of the client, so that the client can access the files quickly without waiting for the remote server. Note that traditional caching schemes cannot be used readily for QAPD as they are designed primarily for flat files. Consequently, we have designed a novel caching scheme for QAPD, termed as the Layered Multimedia Cache (LMC). The LMC acts as a proxy for the original server which hosts the media file denoted by V.

We assume that the LMC has a maximum storage capacity of G and is initially empty. A client request for the media file V is tantamount to a request for the base layer V_0 followed by a request for each of the enhancement layers V_i $1 \leq i \leq 8$. Since none of these layers are in the LMC, the LMC requests the corresponding files from the main server, and caches them while relaying them to the client. We assume that, at a given point in time, the LMC at full capacity contains *M* media files, each consisting of a base layer file and 8 enhancement layer files. When trying to cache media file V_{M+1}, the LMC observes that the cache capacity *G* is exceeded. Obviously, some layers of some of the existing media files in the LMC need to be deleted in order to make space for the new media file V_{M+1}. This calls for an efficient cache replacement policy.

We propose an improvised version of the state-of-the-art Greedy Dual Size (GDS) cache replacement policy [Jin, 2000], [Cao, 1997] used commonly in web proxy caches. The improvised cache replacement scheme is termed as the *Layered GDS* (LGDS) cache replacement algorithm. The standard GDS algorithm is ineffective since it does not exploit the dependencies between the various layers of a media file. The proposed LGDS algorithm is described as follows. Each media file cached in the LMC can be viewed as a bin which contains the base layer and one or more enhancement layers corresponding to the media file. Each media file or bin is associated with a *retention-value* which quantifies the loss incurred by the LMC if some layers of that media file are deleted from the LMC. The lower the *retention-value*, the more dispensable the media file or its constituent layers. Since each media file consists of multiple layers, *the retention-value* of a media file (or bin) is that of its topmost layer which can be removed immediately. The *retention-value* of the j^{th} layer, ($0 \leq j \leq 8$) of the i^{th} media file $V_{i,j}$ depends on three parameters: Latency(i, j), Size(i, j) and δPSNR(i, j). Latency(i, j) is the time taken to get $V_{i,j}$ from the server; Size(i, j) is the size (in bytes) of. $V_{i,j}$ and δPSNR(i, j) is the resulting change in PSNR when $V_{i,j}$ is added to the exisiting layers of the media file. Note that

PSNR is a commonly used metric of visual quality of a video frame [Davies, 1990]. The *retention-value* of $V_{i,j}$ is denoted by RV(i, j) and is given by:

$$\mathrm{RV}(i, j) = K \times \delta\mathrm{PSNR}(i, j) \times \mathrm{Latency}(i, j) / \mathrm{Size}(i, j)$$

where K is a constant. The *retention-value* of a bin i, denoted by *retention-value(i),* is computed as

$$\textit{retention-value(i)} = \mathrm{RV}(i, \mathrm{TopLayer}(i)) \qquad (5.2.1)$$

where $0 \leq \mathrm{TopLayer}(i) \leq 8$ is the topmost cached layer of the media file V_i. Note that the *retention-value* as computed in **equation (5.2.1)** is different from that used in the standard GDS algorithm, since the standard GDS algorithm does not consider the δPSNR value in its computation of the *retention-value.* Also note that the proposed LGDS algorithm, unlike the standard GDS algorithm, results in a caching scheme that is aware of the structural relationships amongst the various layers of a media file. The proposed LGDS algorithm uses the *retention-value*(i) of each bin i to decide which layer to cache and which layer to delete from the cache. When a media file or bin in the LMC scores a hit, its *retention-value* reverts to its original value, i.e., the *retention-value* assigned when the media was first saved in the cache.

Having described the working of the LMC, we now describe the methods used to evaluate its performance. It must be noted that layered multimedia caching differs from standard flat file caching, since each layer in the LMC is associated with a different visual quality enhancement factor. Thus, any scheme for LMC performance evaluation has to take this factor into account. In the following section, we describe the methodology and experiments for performance evaluation of the proposed LGDS algorithm in the context of the LMC followed by a discussion of the results.

5.2.3 Experiments and results

In this section, we present the experimental results of the evaluation of the efficiency of the proposed LMC, and the gains obtained by using the proposed LGDS cache replacement scheme.

5.2.3.1. LMC: Evaluation Methodology

In order to demonstrate the effectiveness of the proposed LGDS cache replacement policy for the LMC, we have compared the LGDS algorithm with two other popular cache replacement policies: Least Recently Used (LRU) and Least Frequently Used (LFU). Both, the LRU and LFU schemes were improvised to account for the structural dependency between the layers in the LMC. The LRU scheme is implemented as follows. For each bin, the time when the file was last accessed is recorded as the *retention-value* for that bin. When a layer is removed to make space for the new layer to be cached, the topmost layer is removed from the bin and the *retention-value* of the bin is left unchanged. When a video file (or bin) is accessed, the *retention-value* of the file (bin) is reassigned to the latest time of access (hit time). The LFU scheme is implemented in a similar manner by deleting the topmost layer from the selected bin based on the number of accesses recorded for that bin (media/video file).

For performance evaluation of the LMC we devise a metric which accounts for the change in visual quality of the media file as well as the latency incurred at the client end to receive a media file of that quality. First, we compute the layered representation of N media files. The N media files are assigned an arbitrary rank between 1 and N. The m^{th} layer of the n^{th} media file is denoted by $f(n, m)$, where $1 \leq n \leq N$, and $0 \leq m \leq 8$. R

requests are made to the LMC using a Zipf distribution, which is known to emulate the access pattern for ranked files [Baldi, 2003]. Requesting a file with rank k, from the client's point of view, is tantamount to requesting all the files $f(k, i)$, $0 \leq i \leq 8$ from the LMC. After each client request, the time taken to receive the i^{th} layer of the k^{th} file, $T'(k, i)$, $0 \leq i \leq 8$, and the resulting change in the PSNR value, δPSNR(k, i), $0 \leq i \leq 8$, are noted. The mean of all the $T'(k, i)$ values, where $1 \leq k \leq N$ and $0 \leq i \leq 8$, is computed over all the R requests. After the R client requests have been serviced, the mean latency for layer j of the file with rank i is denoted by $T(i, j)$, $1 \leq i \leq N$, and $0 \leq j \leq 8$. Note that each of these layers corresponds to a physical file on the disc with size given by Size(i, j). We normalize these latencies using the size information to get a metric $P'(i, j)$ given by

$$P'(i, j) = T(i, j)/\text{Size}(i, j)$$

$P'(i, j)$ is essentially the latency per byte of information for each layer of each file. Finally, the mean of $P'(i, j)$ is computed over all the N files, for each layer. The resulting metric $P(j)$, $0 \leq j \leq 8$, represents the average latency per byte for the j^{th} layer of the LMC. Similarly, the values of δPSNR(i, j), $1 \leq i \leq N$, and $0 \leq j \leq 8$ are used to compute the mean cumulative PSNR, PSNR(j), $0 \leq j \leq 8$, for each layer. PSNR(j) serves as an objective evaluation of the visual quality of the files after the layer j is added to the layers 0, 1,.., j - 1. A plot of $P(j)$ versus PSNR(j), $0 \leq j \leq 8$, is used to assess the performance of the LMC.

The second performance metric is based on the cumulative sum S of the number of bytes transferred from the server during a cache miss, and the time T taken to send the media file to the client. The ratio, S/T, denotes the bandwidth efficiency of the server for a given cache replacement policy. A detailed discussion on the two metrics based on experimental results is given in the next subsection.

5.2.3.2. LMC: Evaluation Methodology

Since trace data are not available for the proposed technique, we simulate network behavior using well known distributions. We create 20 layered video files (*N* = 20), each with one base layer and 8 enhancement layers. Each video is of 5 seconds duration, and with a GOP size of 15 and encoding efficiency of 1 frame per second. We simulate server latency by introducing a random delay modeled by a Gaussian distribution **N**(μ, σ), where μ = 200 milliseconds, and σ = 50 milliseconds, in response to a request. We use cache sizes that are 22%, 55% and 88% of the sum of all

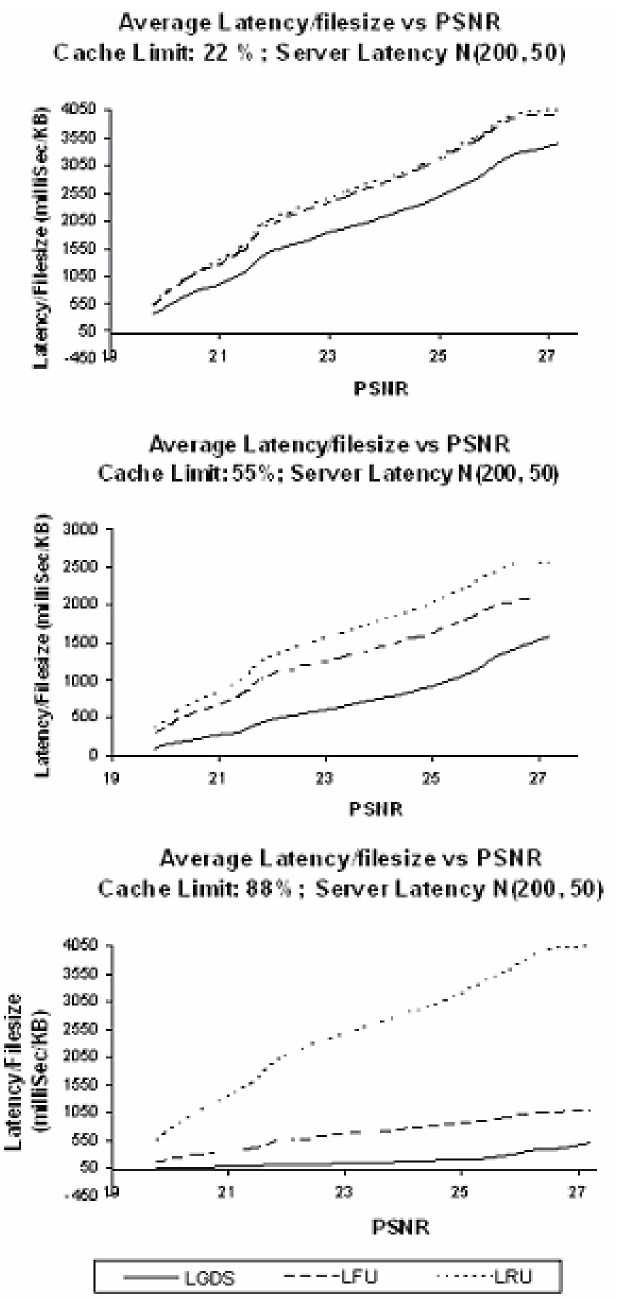


Figure 5.2.2. Latency (in milliseconds) per KB of each layer vs average PSNR increase after each layer is added. Three cache sizes have been used: 22%, 55% and 88% of the total size of all the layers of all the files hosted in the server. The three lines correspond to three cache replacement policies implemented at the Layered-Multimedia-Cache.

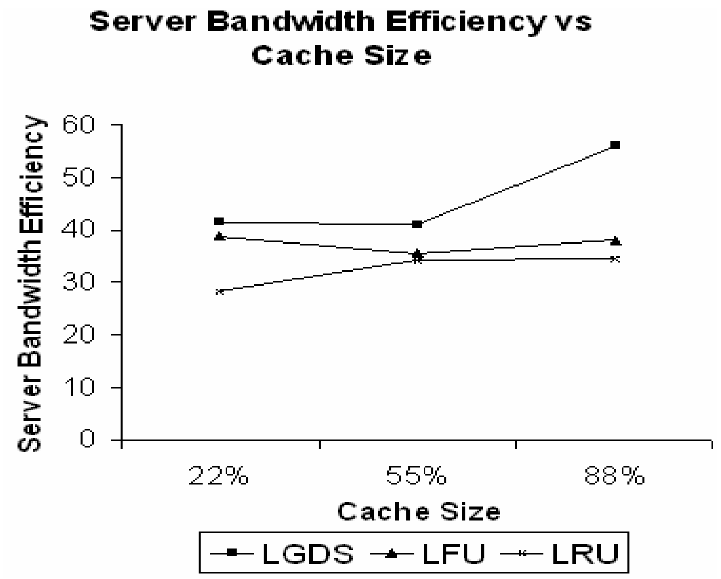


Figure 5.2.3. The server bandwidth efficiency (KB/s) versus the cache size after 1000 client requests.

the media file sizes to observe the effects of change in cache size. We send R = 1000 requests to the LMC, using a Zipf distribution with $\alpha = 0.9$. For each value of cache size, we compute *P(j)* and PSNR(*j*) as explained in the previous subsection.

Plots of *P(j)* versus PSNR(*j*) for the three cache sizes are given in **Figure 5.2.2**. The plot essentially shows the latency per byte incurred while improving the quality of the media files. The lower the latency per byte incurred to improve the quality (i.e., increase PSNR value), the faster the client gets to view the better quality media file. In other words, lower latency per byte value for a given PSNR value signifies that better quality media can be progressively downloaded at a lower latency. From the plot, it is clear that the LMC using the proposed LGDS cache replacement policy outperforms both, the LFU and LRU cache replacement policies.

Figure 5.2.3 plots the second metric, i.e., server bandwidth efficiency, versus the cache size. The larger the server bandwidth efficiency, the better the server performance, and hence more efficient the cache is, in utilizing the server. From **Figure 5.2.3**, it is evident that the proposed LGDS cache replacement policy results in comparable, if not greater, server efficiency, and hence, a lighter server load, compared to the LFU and LRU cache replacement policies. Thus, with similar or lighter server load compared to the standard LFU and LRU cache replacement policies, the proposed LGDS cache replacement policy results in a significant reduction in the client-experienced latency during progressive download of the layered video.

5.2.4. Conclusions

In this chapter, we have proposed a novel framework for quality adaptive progressive download (QAPD) of multimedia files. The proposed QAPD framework incorporates an efficient multi-layered video representation that is suitable for progressive video download at varying bitrates, an efficient Layered Multimedia Cache (LMC) in conjunction with a novel Layered Greedy Dual Size (LGDS) replacement policy, and novel evaluation metrics to quantify the performance of the QAPD system. Experimental comparisons with two other popular cache replacement policies, i.e., LRU and LFU, show that the proposed QAPD scheme, using the proposed LGDS cache replacement policy, significantly outperforms the conventional LFU and LRU replacement policies.

5.3. Video Caching for Video Personalization Servers

The current proliferation of mobile computing devices and networking technologies has created enormous opportunities for mobile device users to communicate with multimedia servers. As handheld mobile computing and communication devices such as personal digital assistants (PDAs), pocket-PCs and cellular devices have become increasingly capable of storing, rendering and display of multimedia data, the user demand for being able to view streaming video on such devices has increased several-fold. For example, a mobile handheld client may be interested in viewing a video showing traffic conditions on the road and browsing the weather forecast for his/her travel destination. One of the natural limitations of typical handheld mobile devices is that they are resource constrained, i.e., constrained by their battery capacity, screen resolution, video decoding and rendering capability, and, in many situations, by the available network bandwidth connecting them with video servers. Thus, the original video content often needs to be personalized in order to fulfill the client's request while simultaneously satisfying various client-side and system-level resource constraints. Numerous video personalization strategies have been developed [Merialdo, 1999],[Tseng, 2003],[Tseng, 2004] in order to provide these resource-constrained devices with personalized video content that is most relevant to the client's request and the available client-side and network resources.

Since mobile clients are typically not within network proximity of the personalizing server, it is often desirable to intelligently cache portions of the video files in order to reduce the client-observed latency and also offload the data load on the server to local caches. For the work proposed in this chapter, the design and implementation of a video personalization server (VPS) [Wei, 2007c] [Wei, 2007a] which performs automatic video segmentation and video indexing based on semantic video content, and generates personalized video content based on the client's content preferences and resource constraints using a Multiple-Choice Multi-Dimensional Knapsack Problem [Hernandez, 2005],[Vanderbei, 1997] (MMKP)-based video personalization strategy, has been used.

A novel cache design, accompanied by a novel cache replacement algorithm, which is specifically suited for caching video files generated by the proposed VPS, has been proposed and implemented in this section. In the proposed cache, different versions of the cached video files are created using state-of-art content-based video encoding techniques. The proposed cache replacement algorithm performs significantly better than several state-of-the-art cache replacement policies.

Communication between clients and the VPS typically occurs in a two phases, wherein the client first asks for the content to be generated and then downloads the contents. Since the actual download occurs in the second phase of communication, the cache, which serves as an intermediary between the client and the VPS, can learn important statistical facts about the requests in order to pre-customize the video to be disseminated based on the download pattern that is to follow the first phase of communications.

The proposed cache uses a content-aware video re-encoding algorithm termed as Features, Motion and Object Enhanced Multi-Resolution FMOE-MR video encoding [Chattopadhyay, 2007a], in order to create two versions of the original video file of lower visual quality and correspondingly lower file size. These versions are useful in delivering videos of different visual quality to different clients depending on their levels of “membership”, defined as “paying” and “non-paying”.

Since the proposed cache is aware of the two-phase communication between the clients and the VPS, and has the ability to generate videos of different visual quality to serve multiple clients with different levels of membership, it is observed to perform significantly better than standard caching techniques.

In the following sections, we describe the proposed VPS, which hosts and serves newscast videos from various sources. Next I describe the video personalization cache in detail, followed by results and conclusions.

5.3.1 Video Personalization Server

The videos hosted by the video personalization server (VPS) are first segmented and indexed. The videos are then transcoded at multiple levels of abstraction based on their content. This allows for video personalization based on clients' preferences and resource constraints.

5.3.1.1. Video Segmentation and Indexing

A stochastic multi-level Hidden Markov Model (HMM)-based algorithm is used for video segmentation and indexing wherein the input video stream is classified frame by frame into semantic units [Wei, 2007b]. A semantic unit within a video stream is a video segment that can be associated with a clear semantic meaning or concept, and consists of a concatenation of semantically and temporally related video shots or video scenes. Instead of detecting video shots or scenes, it is often much more useful to recognize semantic units within a video stream to be able to support video retrieval based on high-level semantic content. Note that visually similar video shots or video scenes may be contained within unrelated semantic units. Thus, video retrieval based purely on detection of video shots or video scenes will not necessarily reflect the semantic content of the video stream.

The semantic units within a video stream can be spliced together to form a logical video sequence that the viewer can understand. In well organized videos, such as TV broadcast news and sports programs, the video can be viewed as a sequence of semantic units that are concatenated based on predefined video program syntax. Parsing a video file into semantic units enables video retrieval based on high-level semantic content and playback of logically coherent blocks within a video stream. Automatic indexing of semantic components within a video stream can enable a viewer to jump straight to points of interest within the indexed video stream, or even skip advertisement breaks during video playback.

In the proposed scheme, a video stream is modeled at both, the semantic unit level and the program model level. For each video semantic unit, an HMM is generated to model the stochastic behavior of the sequence of feature emissions from the image frames. Each image frame in a video stream is characterized by a multi-dimensional feature vector. A video stream is considered to generate a sequence of these feature vectors based on an underlying stochastic process that is modeled by a multi-level HMM.

Two categories of features from each image frame in the video stream are extracted. The first category includes a set of simple features. The dynamic characteristics of the image frames comprising the video stream are captured by the differences of successive image frames at both, the pixel level and the histogram level. Various motion-based measures describing the movement of the objects in the image frames are used, including the motion centroid of the image, and intensity of motion. Measures of illumination change at both, the pixel level and the histogram level are also included in the multi-dimensional feature vector. Definitions of these features are given in [Eickeler, 1999]. In the second feature category, Tamura features [Flickner, 1995] are used to capture the textural characteristics of the image frames at the level of human perception. Tamura contrast, Tamura coarseness and Tamura directionality have been used successfully in content-based image retrieval [Flickner, 1995]. In our work, inclusion of these features is observed to improve the accuracy of temporal video segmentation and video indexing.

In the proposed video segmentation and video indexing scheme based on semantic video content, six semantic concepts for TV broadcast news video, i.e. *News Anchor*, *News*, *Sports News*, *Commercial*, *Weather Forecast* and *Program Header*, and three semantic concepts for Major League Soccer (MLS) video, i.e. *Zoom Out*, *Close Up* and *Replay*, are used. An HMM is formulated for each individual semantic concept. The optimal HMM parameters for each semantic unit are learned from the feature vector sequences obtained from the training video data. In the proposed scheme, the HMMs for individual semantic units are trained separately using the training feature vector sequences. This allows for modularity in the learning procedure and flexibility in terms of being able to accommodate various types of video data. In our work, we adopt a universal left-to-right HMM topology, i.e., an HMM topology where no backward state transitions are allowed, with continuous observations of the feature vector emissions. The distribution of the feature vector emissions in the HMM is approximated by a mixture of Gaussian distributions.

The search space for the proposed single-pass video segmentation and video indexing procedure is characterized by the concatenation of the HMMs corresponding to the individual semantic units. The HMM corresponding to an individual semantic unit essentially models the stochastic behavior of the sequence of image features within the scope of that semantic unit. Transitions amongst these semantic unit HMMs are regulated by a pre-specified video program model. The Viterbi algorithm is used to determine the optimal path in the concatenation of the HMMs in order to segment and index video stream in a single pass [Wei, 2007b].

5.3.1.2 Video Personalization

The objective of video personalization is to present a customized or personalized video summary that retains as much of the semantic content desired by the client as possible, but within the resource constraints imposed by the client.

5.3.1.2.1. Relevance Value of a Video Segment and its Summary

Video segments are indexed using semantic terms. Each video segment is assigned a relevance value based on the client's preference with regard to video content. Assume video segment S_i is indexed by a semantic term T_i. In its request, the client specifies a preference for video content using a descriptive term labeled as *P*. The relevance value V_i assigned to the video segment S_i is then given by:

$$V_i = similarity(T_i, P), 0 \le V_i \le 1 \qquad (5.3.1)$$

In the current implementation the *similarity* is evaluated using the *lch* semantic similarity measurement algorithm [Leacock, 1998].

Each indexed video segment is summarized at multiple levels of abstraction using content-aware key frame selection and motion panorama computation algorithms [Wei, 2007c]. Each video summary consists of a set of key frames and motion panoramas. For each video segment, its original version is assumed to contain the greatest amount of detail; whereas its summary at the highest level of abstraction is assumed to contain the least amount of detail. It is reasonable to assume that the amount of information contained within a video summary (relative to original version) is related to its duration, i.e.

$$v_i = v_{i0} \cdot f(L_i / L_0) \qquad (5.3.2)$$

where v_{i0} is the relevance value of the original video segment, and L_0 and L_i are the time durations of the original video segment and the video summary respectively. Typically, the amount of information contained within a video summary (relative to original version) does not necessarily increase linearly with its relative duration. The empirical Zipf's law to quantify $f(L_i / L_0)$ [Wei, 2007c] has been used.

5.3.1.2.2. MMKP-based Video Personalization

The objective of video personalization is to present a customized or personalized video summary that retains as much of the semantic content desired by the client as possible, but within the resource constraints imposed by the client. The client typically wants to retrieve and view only the contents that match his/her content preferences. In order to generate the personalized video summary, the client preferences, the client usage environment and client-side and system-level resource constraints need to be considered. The personalization engine selects the optimal set of video contents (i.e., the most relevant set of video summaries) for the client within the resource constraints imposed by the client. This chapter presents the design and implementation of an MMKP-based video personalization strategy to generate a customized response to the client's request while satisfying multiple client-side and system-level resource constraints. Compared to the 0/1 Knapsack Problem (KP)-based and the Fractional Knapsack Problem (FKP)-based video personalization strategies presented in [Merialdo, 1999], [Tseng, 2003] and [Tseng, 2004], the proposed MMKP-based video personalization strategy is shown to include more relevant information in its response to the client's request. The MMKP-based personalization strategy is also shown to support multiple client-side constraints, in contrast to the 0/1KP-based and the FKP-based personalization strategies which can support only a single client-side resource constraint at a time.

In the video content database, each video segment and summary is assigned a relevance value based on the client's content preferences, as computed in **equations (5.3.1)** and (**5.3.2**) respectively. In many applications, it is desirable to provide the client with as much information as possible. In such cases it may be preferable to include two shorter video summaries in the response rather than a single video segment of longer duration that contains more details. For example, if a client needs to browse the sports news of the day, it might be helpful to provide him/her with multiple, though short, sports news summaries rather than a single long and detailed video segment containing news of a specific sport. We propose a Multiple-Choice Multi-Dimensional Knapsack Problem (MMKP)-based video personalization strategy [Akbar, 2001], [Hernandez, 2005].

Let v_{ij} be the relevance value of the j^{th} summary of the video segment S_i, $\overset{\varpi}{r_{ij}} = (r_{ij1}, r_{ij2}, ..., r_{ijm})$ be the required resource vector for the j^{th} summary of the video segment S_i and $\vec{R} = (R_1, R_2, ..., R_m)$ be the resource bound on the client side. The problem is to determine

$$V = \max(\sum_{i=1}^{n} \sum_{j=1}^{l_i} x_{ij} v_{ij}),$$

subject to

$$\sum_{i=1}^{n} \sum_{j=1}^{l_i} x_{ij} r_{ijk} \le R_k, k = 1,2,...,m$$

and

$$\sum_{j=1}^{l_i} x_{ij} = 1, x_{ij} \in \{0,1\} \qquad (5.3.3)$$

The above MMKP can be solved using the branch and bound integer programming (BBIP) algorithm described in [Vanderbei, 1997].

When the VPS receives multiple client requests within a predefined time window (i.e., batching window), it clusters these requests based on the clients' content preferences and client-side resource constraints. The two-phase clustering algorithm described in [Wei, 2007a] is employed for this purpose. In the first phase, clustering is performed based on the clients' content preferences. This is followed by the second phase where clustering is performed on the results of the first phase, based on the client-side resource constraints. The goal of the two-phase clustering procedure is to reduce the computation burden on the VPS, since the video personalization is done on a per-cluster basis rather than a per-client request basis.

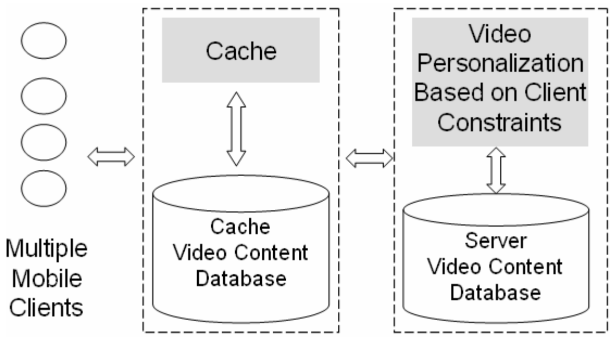

Cache
Video Personalization Based on Client Constraints
Multiple Mobile Clients
Cache Video Content Database
Server Video Content Database

Figure 5.3.1: Multi-client Cache and Video Personalization Server architecture

5.3.2. Video personalization cache

In the previous section, we have detailed the design of a video personalization server (VPS) that receives requests from multiple clients, and creates personalized videos for them. The cache serves as an intermediary between the clients and the VPS (**Figure 5.3.1**). In the following subsections, we first detail the design of the proposed cache. Next, we describe how the proposed cache creates different versions of the video files using content-aware video processing. Finally, we describe in detail the protocol followed by the cache and clients in order to synchronize with the VPS.

5.3.2.1 Cache design

The proposed cache serves the following two purposes:

(a) It acts as a buffer between the VPS and multiple clients, in order to reduce the overall latency observed by the clients.

(b) It acts as a generative proxy video server that generates videos of different visual quality in order to increase cache efficiency.

From each video segment that the cache receives for storage, two additional versions, or layers, of the same video are created. The original video segment, or layer, is

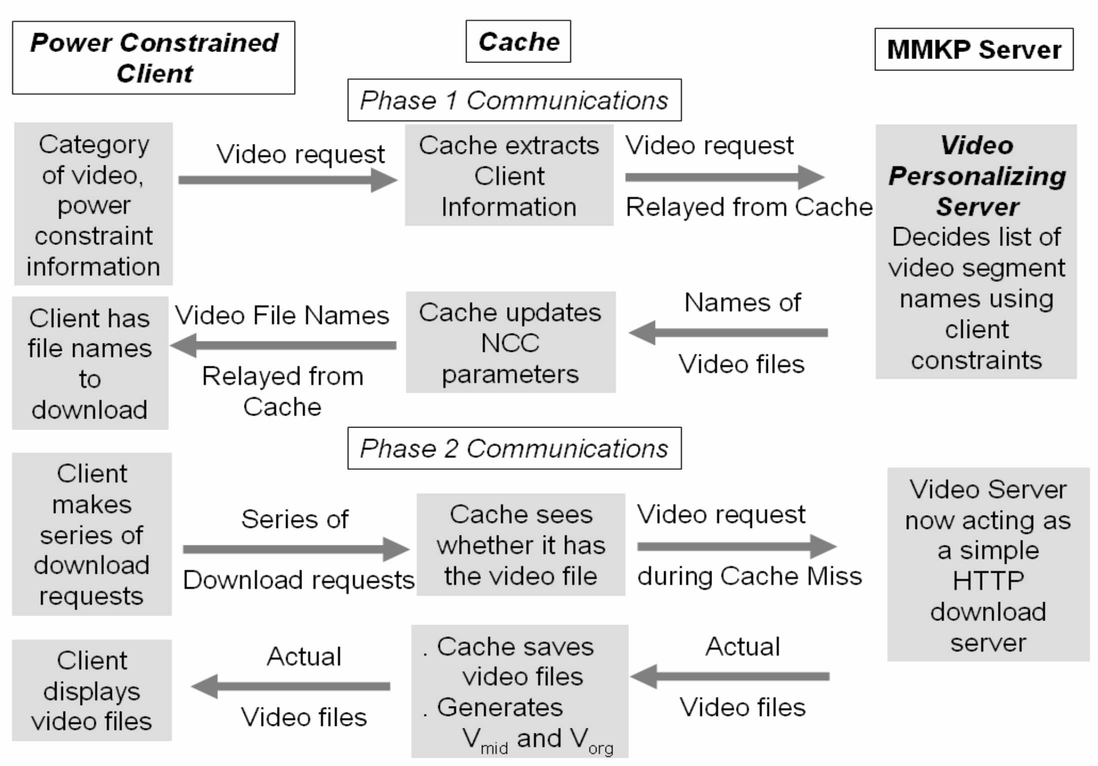

Power Constrained Client
Cache
MMKP Server
Phase 1 Communications
Category of video, power constraint information
Video request
Cache extracts Client Information
Video request
Relayed from Cache
Video Personalizing Server
Decides list of video segment names using client constraints
Client has file names to download
Video File Names
Relayed from Cache
Cache updates NCC parameters
Names of
Video files
Phase 2 Communications
Client makes series of download requests
Series of
Download requests
Cache sees whether it has the video file
Video request
during Cache Miss
Video Server now acting as a simple HTTP download server
Client displays video files
Actual
Video files
. Cache saves video files
. Generates V_mid and V_org
Actual
Video files

Figure 5.3.2. The client-cache-server communication protocol. The protocol is to be read from top to bottom. NCC stands for number of common clients. V_{mid} and V_{org} are lower quality videos generated at the cache from each original video segment.

designated as V_{org}. V_{mid} is an intermediate layer video of lower visual quality (and smaller file size) than V_{org} whereas V_{base} is the base layer video of lowest visual quality and smallest file size. As will be seen in Section 5.3.3.3, the lower quality video layers V_{mid} and V_{base} are used to design an efficient cache replacement policy. In the next section, we describe in detail the process by which V_{mid} and V_{base} are created. **Figure 5.3.2** depicts the proposed multi-client video personalization and caching system.

5.3.2.2. Creating different layers of the video

The different video layers are essentially transcoded versions of the original video at different levels of visual quality with file sizes that are significantly smaller than the file size of the original video. We have experimented with novel transcoding methods such as Features, Motion and Object Enhanced Multi Resolution (FMOE-MR) video encoding [Chattopadhyay, 2007a] and Ligne-Claire video encoding [Chattopadhyay, 2007b]. Due to its relative simplicity in terms of implementation for the purpose of caching, we have chosen FMOE-MR as the trancoding technique for this chapter. In the following subsections, we describe in detail how the two layers, V_{mid} and V_{base}, are created from the original video segment, Vorg.

5.3.2.2.1. Creating V_{mid}

The video layer V_{mid} represents an intermediate-level video which has a more compact representation than the original video V_{org}, albeit at the cost of lower visual quality. The video layer V_{mid} is generated using a novel multi-resolution video encoding technique termed as Features, Motion and Object-Enhanced Multi-Resolution (FMOE-MR) video encoding [Chattopadhyay, 2007a]. The FMOE-MR video encoding scheme is based on the fundamental observation that applying a low pass filter in the image color space is equivalent to DCT coefficient truncation in the corresponding DCT (frequency) space [Geusebroek, 2001].

The FMOE-MR video encoding scheme is a two step process as described below. First, instances of Features, Motion and Objects (FMOs) are detected in the video sequence using state-of-the-art computer vision algorithms. A corresponding mask (FMO-Mask) is created to mark the regions corresponding to the presence of the FMOs. The mask contains floating point values between (and inclusive of) 0 and 1, where 0 represents a completely uninteresting region and 1 represents a region that is vital for visual and semantic understanding of the image.

After the FMO-mask is created, the original frames of the video are re-encoded as a multi-resolution (MR) representation, guided by the FMO-mask such that regions corresponding to mask values close to 1 are at higher resolution than regions corresponding to mask values close to 0. The original video frame V_O is used to render two video frames, V_H and V_L, such that V_H is a high-resolution rendering and V_L is a low-resolution rendering of V_O. The video frames V_L and V_H are obtained by convolving V_O with Gaussian filters characterized by the smoothing parameters σ_L and σ_H respectively. Maintaining $\sigma_L > \sigma_H$ ensures that V_L is smoother than V_H, i.e., V_L is a lower resolution rendering of V_O than V_H. If the FMO mask is represented as a matrix W whose elements lie in the range [0, 1], then the MR frame V_{MR} is obtained via a linear combination of the two frames V_H and V_L as follows:

$$V_{MR} = W \bullet V_H + (I - W) \bullet V_L$$

where I is a matrix all of whose elements are 1.

The values of σ_L and σ_H used to generate $V_{mid} = V_{MR}$ are selected empirically by the user. Empirical observations have revealed that $\sigma_L = 9$ or 11, and $\sigma_H = 3$, can be used, in most cases, to yield videos of reasonable visual quality with significantly smaller file sizes than the original video.

Since an exhaustive treatment of the FMOE-MR video encoding schemes is beyond the scope of this chapter, the interested reader is referred to [Chattopadhyay, 2007a] for further details. It must be noted that finally, V_{mid} is encoded using the MPEG

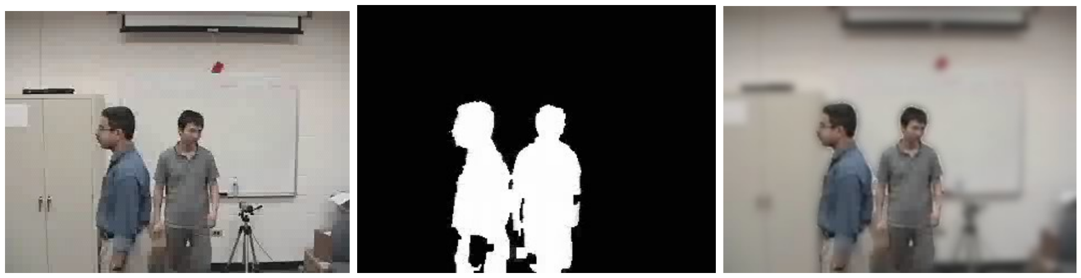

(a) (b) (c)

Figure 5.3.3. A V_{mid} frame using Features, Motion and Object-enhanced Multi-Resolution (FMOE-MR) video encoding. (a) The original frame; (b) The FMO mask frame; (c) The frame re-rendered using FMOE-MR video encoding. The moving objects are rendered at high resolution whereas the background is rendered at low resolution. The obtained PSNR is 24.94

H.264 standard, after preprocessing using FMOE-MR, in a manner similar to the original video segment. An example video frame of a V_{mid} video layer is given in **Figure 5.3.3**.

5.3.2.2.2. Generating V_{base}

The base video layer V_{base} is generated by first blurring each frame of the video using a Gaussian filter with smoothing parameter σ_{base} prior to MPEG H.264 encoding. Note that this is similar to the Gaussian smoothing performed in the case of FMOE-MR video encoding. The primary difference is that, in the case of the V_{base} video layer, the smoothing operation is performed uniformly over the entire video frame in contrast to FMOE-MR video encoding where the extent of smoothing can vary within a video frame based on the perceptual significance of the region under consideration. This results in further dramatic decrease in the file size upon MPEG H.264 encoding, albeit at the loss of video quality. Note that V_{base} is of much lower visual quality than V_{mid} since object-based enhancement is not used.

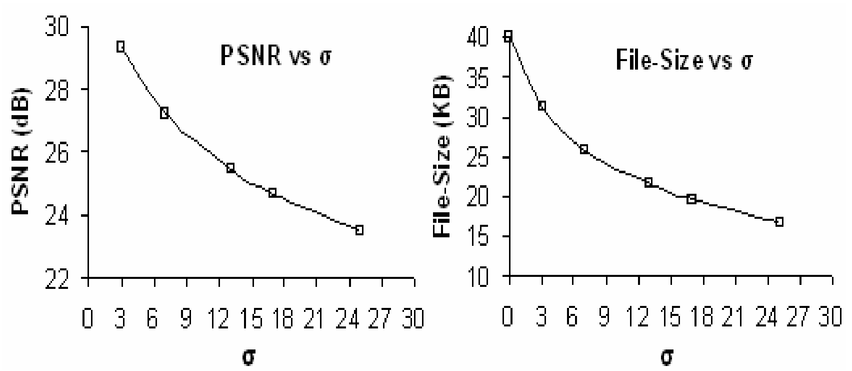


Figure 5.3.4. The effect of blurring parameter σ on the file-size, and quality (measured by PSNR) of a V_{base} video.

5.3.2.2.3. Visual quality of V_{mid} and V_{base}

The visual quality of V_{mid} and V_{base} can be assessed in terms of PSNR values, as well as via subjective visual evaluation. A quantitative evaluation of the average PSNR and file size, of a sample video, with respect to value of the Gaussian blurring parameter is depicted in **Figure 5.3.4**. It is apparent that the video size can be decreased significantly by using a high value of σ_{base}, albeit with a loss in video quality. We have observed empirically that values of σ_{base} in the range {19, 25} can be used to generate the base video layer V_{base}, resulting in a very small file size albeit at the cost low visual quality and low resolution.

5.3.2.2.4. File Size overhead

V_{mid} and V_{base} are additional files stored in the cache. For empirically derived encoding parameter values for σ_L, σ_H and σ_{base}, the total sum of all the files, V_{org}, V_{mid} and V_{base} is around 1.5 times that of V_{org}; i.e., the combined size of V_{base} and V_{mid} adds ≈ 50% overhead to storage space required for storing the video segment.

Thus ensuring reasonably good visual quality for V_{mid} and V_{base} makes it possible to discard V_{org} to allow for extra space in the cache, and yet have videos of reasonable visual quality reside in the cache, during a cache replacement.

Having discussed methods to create additional videos from the original videos in the cache, and the cache design in general, we now discuss the client-cache protocol in the next section.

5.3.2.3. Client-Cache-Server communication

We have considered a setup involving a single VPS, a single cache, and multiple clients. The client-cache-server communication is done in two phases:

Phase 1:

Multiple clients send their queries or requests to the cache, which relays the requests to the VPS. Each client request essentially consist of one category out of six possible categories (*News Anchor*, *News*, *Sports News*, *Commercial*, *Weather Forecast* and *Program Header*), and the available battery time in the client device as a resource constraint. The VPS generates a list of video segments based on the client requests and resource constraints, and sends the names of the video segments to be downloaded (in a specified order), as metadata back to cache. The cache relays this metadata information back to each client.

Phase 2:

Each client now makes a series of requests to the cache to fetch the files belonging to one of the four categories. The files can be streamed, or can be progressively downloaded, depending on the type of service available at the cache and the VPS. In either case, recall that the names of the files, and the order they appear in, have been specified in the metadata obtained in *Phase 1* of the communication. When a client requests a video segment file from the cache, it is scored as either a *hit* or a *miss*. **Figure 5.3.2** depicts the two phases of communication described above.

A novelty in the proposed cache design is that fact that a hit or a miss score depends on the *client-type*. A *client-type* defines the membership status of the client with regard to the video personalization service. Without loss of generality, we have used two categories of clients - either they are *paying* or *subscribing* clients, who are paying for the best quality of service, or they are *non-paying* clients who receive videos of varying quality depending on the availability of the videos in the cache. For non-paying clients, the best quality video is not guaranteed; however the best quality video is guaranteed for the paying client.

If the requesting client is *non-paying*, then the cache checks if the requested file V_{org} is present in the cache. If it does not exist in the cache, it checks whether a lower quality version of the file, i.e., V_{mid}, is present in the cache. If V_{mid} is absent, then it checks whether V_{base} is present in the cache. If neither V_{mid} nor V_{base} are present in the cache the client request is treated as a cache miss. In contrast, if the requesting client is *paying* then the client request is treated as a cache miss if the requested file V_{org} is absent in the cache. Thus, for a *paying* client, the best quality video is provided, whereas for the non-paying client, the quality of video which is present in the cache is present; it might not be the best quality one.

In the event of a cache miss, whether the client type is a *paying* or *non-paying*, the requested file is relayed from the VPS, and also stored in the cache simultaneously. If there is not enough space available in the cache, a cache replacement policy is enforced to replace existing file(s) in the cache in order to make space for the requested file. If the cache cannot accommodate the requested file, an existing file with the minimum *retention-value* (RV) is discarded. The *retention-value* is essentially a number associated with each file in the cache; the lower the *retention-value*, the less valuable is the file to the cache. Thus, if an existing file is to be discarded from the cache in order to make space in the cache, the one with the lowest *retention-value* is removed.

The computation of the *retention-value* is specific to a cache replacement policy. For the proposed cache, we compute the *retention-value* as follows:

$$\textit{retention-value} = \text{NCC}/\textit{fileSize}$$

where NCC is the *number of common clients* requesting that particular file, and *fileSize* is the size of the file. This cache replacement policy has been termed NCCS (Number of Common Clients - Size) algorithm. NCCS essentially denotes the number of client requests for this particular file normalized by the file size. In order to assign higher priority to files which are requested by paying clients, the number of clients (NCC) is incremented by 10, instead of 1, for paying clients as opposed to non-paying clients. This information is obtained during the first phase of the communication between the cache and the VPS.

As will be seen in the following experimental results section, the proposed NCCS cache replacement policy that uses the proposed *retention-value* as the replacement criterion is better suited for caching of personalized video than other standard cache replacement algorithms.

Recall that in the case of a miss, the file with the least *retention-value* in that category is discarded. The file replacement process is as follows. First, the video file with the minimum *retention-value* is identified. After that, an attempt is made to remove the original video segment; i.e., V_{org} corresponding to that video file. If V_{org} had already been removed previously, then V_{mid} is removed; if V_{mid} has also been removed, then V_{base} is removed. This top down approach ensures that the layer which takes up the largest amount of space in the cache is removed, so that space for more popular video segments, can be made.

5.3.3. Results and discussions

In section, a comparison is made of the proposed NCCS cache replacement policy with several existing cache replacement policies. First, we describe the experimental setup. Next, we describe the performance of the cache in terms of two existing cache performance metrics. Finally, we discuss the effect of real time creation of multiple alternative versions or layers of the original video.

5.3.3.1. Experiment setup

In order to simulate multiple clients, we have considered 100 clients which send their requests to the VPS with time lag determined by the following equation:

$$\text{TimeLag(clientID)} = K*(1 + \textit{rand}*\text{clientID})$$

where K is a constant, typically depicting several seconds, $0 \leq \text{clientID} < 100$, and *rand* is a random number between 0 and 1. It is assumed that around 40% of the clients are paying clients; the rest are non-paying.

At the VPS, the video segments and their abstractions are stored as independent video files, which can be independently downloaded/streamed from the VPS as desired. The combined size of the video segments in VPS is 140 MB.

The space available at the cache is typically considerably less than that available at the VPS. We chose two cache sizes; one that is one-third the space capacity of the VPS (33%), and the other that is half the space capacity of the VPS (50%), in order to note the effects of changing cache size.

5.3.3.2. Cache efficiency: hit ratio

In order to come up with a comprehensive study of the cache behavior, we have compared several types of cache replacement policies – each with its own method of deriving the *retention-value*. At any time in its life in the cache, for the cache replacement policies considered, the *retention-value* of each video segment depends on the following parameters:

- *nClients*: The number of clients which are requesting this file. This information is derived during Phase 1 of the communication when the clients make initial requests for the files to the cache. If a requesting client type for a file is non-paying, *nClients* is incremented by 1. If the client is a paying client, then *nClients* is incremented by 10 (the value 10 is empirically chosen). This scheme can accommodate multiple grades of service; for example, for even more special clients, the value of *nClients* can be incremented by 20, and so on.

- *nMisses*: The number of misses that a video segment has experienced so far. The first time a video file is requested is obviously scored as a miss; as a result, the minimum value of *nMisses* is 1.
- *nHits*: The number of hits that a video segment has experienced so far.
- *Size*: The file size on hard disk of the video segment. Typically, larger files are not preferred in the cache because the general philosophy is to have more, smaller-size files, instead of one, large file.
- *Latency*: Latency incurred while fetching the video segment file. If the video segment files are distributed over multiple video personalizing servers, the latency is an important criterion since it is typically not desirable to fetch files that are far away in terms of network distance.
- *hitTime*: The time (in nanoseconds) at which the latest hit to the video segment was made.

An important fact to note is that the *retention-value* is computed for a video segment file V which contains three components: V_{org}, V_{mid} and V_{base}. It can be said that the three video layers together form a *video bin*; the *retention-value* is computed for the V_{org} component of the *video bin*.

We have considered four other standard cache replacement policies in order to do a performance comparison with the proposed NCCS cache replacement algorithm. The replacement algorithms, with their corresponding *retention-values*, are as follows:

GDS (Greedy-Dual-Size): The GDSF cache replacement algorithm [Jin, 2000] uses a retention value given by.

$$retention\_value = Latency / Size$$

In addition to removing files with the minimum retention value during a cache replacement, the GDS algorithm also subtracts this minimum *retention-value* from the *retention-value* for each of the other files in the cache. When a file is scored as a hit, the original *retention-value* of the file is recomputed.

GDSF (Greedy-Dual-Size-Frequency): The GDSF cache replacement algorithm [Cherkasova, 1998] uses a retention value given by.

$$retention\_value = nHits \times Latency / Size$$

Similar to GDS, GDSF subtracts this minimum *retention-value* from all the *retention-value* in each of the other files in the cache. When a file is hit, the original *retention-value* of the file is recomputed.

LRU (least Recently Used): The retention value is given by:

$$retention\text{-}value = \text{hitTime}$$

This simple retention value is used to replace files that were accessed farthest in the past.

LFU (Least Frequently Used): The retention value is given by:

$$retention\text{-}value = nHits + nMisses$$

In other words, the *retention-value* is the number of accesses to the file, which can be simply used as a measure of the frequency of access.

We computed the standard metrics *Hit Ratio* and *Byte Hit Ratio*, in order to measure cache efficiency. They are defined as follows:

Hit Ratio: The number of requests satisfied by the proxy cache as a percentage of total requests.

Byte Hit Ratio: The number of bytes that transferred from the proxy cache as a percentage of total number of bytes transferred for all the requests.

Figure 5.3.5 shows the *Hit Ratio* and *Byte Hit Ratio* for a cache size that is one third that of the VPS storage capacity. **Figure 5.3.6** shows the same for cache size that is

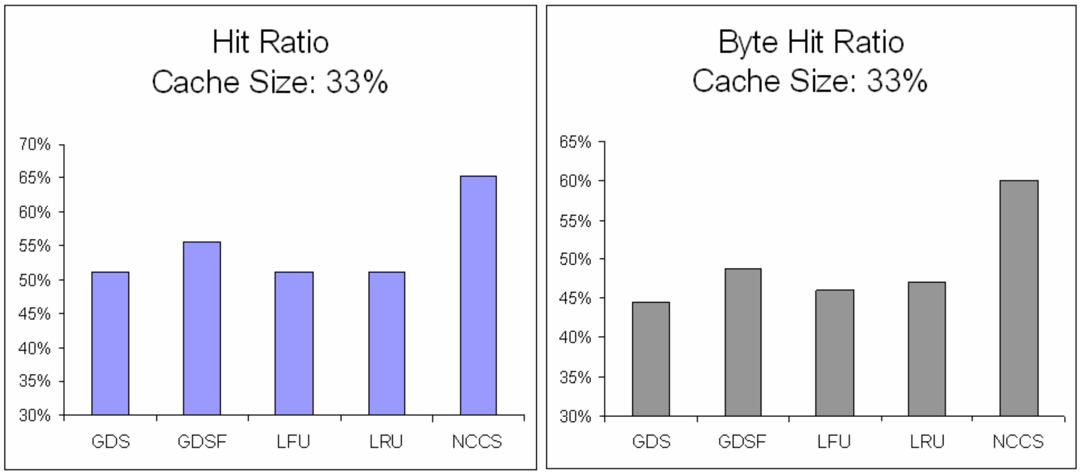


Figure 5.3.5. The hit ratio and the byte hit ratio for cache size = 33% of that of the server.

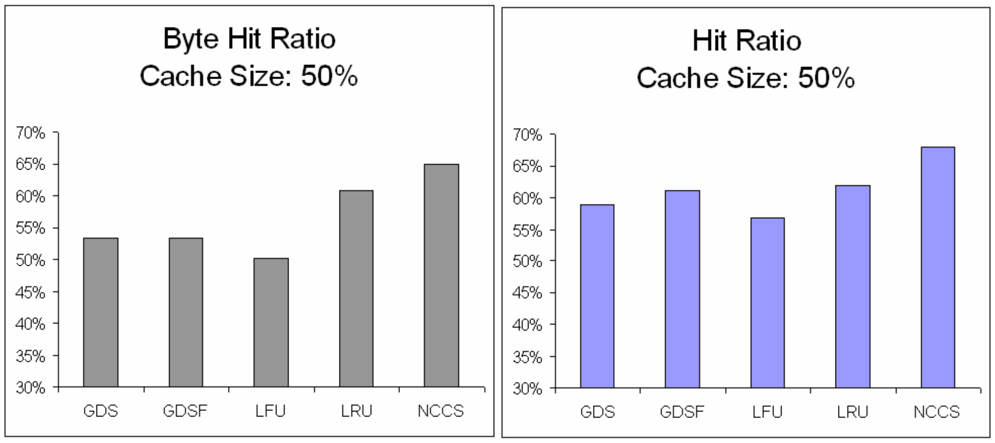


Figure 5.3.6. The byte hit ratio and the hit ratio for cache size = 50% of that of the server.

half the size of the VPS storage capacity. As is apparent from **Figures 5.3.5** and **5.3.6**, the proposed NCCS cache replacement policy outperforms significantly the other cache replacement policies.

We surmise that the reason proposed NCCS cache replacement policy outperforms other cache replacement schemes is because of the fact that, in this particular case, due to the two-phase communication initiated between the clients and VPS, the NCCS scheme knows in advance which files will be asked for by the clients. Note that this is a special situation that can arise only in the case of the above two-phase communication. Thus, the cache captures the popularity of a file by computing the commonality of that particular file (video segment) amongst all the clients who will request the file in the near future. This fact is captured by the NCC statistics (number of common clients). In addition, the Size factor in the NCCS cache replacement policy makes NCCS prefer more, smaller files, compared to fewer, larger files, which should be the case for a good caching scheme. As a result, the proposed NCCS cache replacement policy is observed to be the most successful.

5.3.3.3. Cache efficiency: generating video layers

Experimental results show that the cache, running on a 2.0 GHz Pentium Processor with 2 GB RAM and 2 MB L2 cache, could create V_{mid} and V_{base} in real time; i.e., around 30 frames per second. The file access for each client is, in a simplistic case where the client requests the series of video segments one by one without buffering, where the requests are spaced by the time needed to view the file. Since most video segments are typically in the order of magnitude of a minute or shorter, the creation of both, the V_{mid} and the V_{base}

layer, for each video segment, is around that order. While V_{mid} or V_{base} is being created, it is not present in the cache. Thus, the cache replacement policy sees the extra video layers created only after a certain time lag.

5.3.4. Conclusions

A server-cache architecture has been proposed and implemented, which can disseminate personalized video contents to resource-constrained devices. A novel caching mechanism, dedicated to cache video segments generated by a video personalization server (VPS), has been proposed. The proposed caching mechanism exploits the commonality of files across clients to use a novel cache replacement mechanism termed as NCCS (Number of Common Clients Size). In addition to caching video files, the cache also generates two lower quality versions of each video file. These versions aid in the dissemination of video files to multiple clients entitled to different levels of service based on whether they are paying for the video service or not. Results show that the proposed cache, in conjunction with the proposed NCCS cache replacement policy, outperforms standard cache replacement policies such as GDS, GDSF, LRU and LFU. Thus, the video-cache architecture is suitable for personalized video dissemination to resource-constrained mobile devices such as mobile phones, PDAs, Pocket PCS and laptop computers operating in battery mode.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Contributions

In the first chapter of this dissertation, I gave an overview of the challenges associated with the development of novel methods for multimedia transcoding for resource constrained environments. In the subsequent four chapters that followed, I had elucidated various novel algorithms and techniques for multimedia transcoding, and architecture designs for multimedia caches, that I had developed over my PhD research. In the following sections, I will highlight my contributions to the field of multimedia transcoding, with a more philosophical and aesthetic point of view.

6.1.1. Contribution to Computer Animation

The reader must now be familiar with the various MoCap data compression algorithms that I discussed in the second chapter, i.e. BAP-Sparsing, BAP-Indexing, BAP-Sparse-Indexing, and weighted PCA. These novel algorithms, besides being efficient, demonstrate, for the first time, that MoCap data is more than just a matrix of numbers; it is, in fact, the reflection of an underlying structure in the data, which has been driven by the hierarchical structure of the model for which the data is. The fact that errors for different joints count for different overall error in the virtual human pose, has been one of my major observations. Harnessing this observation to develop efficient compression algorithms was another major contribution to the field of Motion capture data compression. While pondering over possible data loss associated with lossy MoCap data compression, I realized that the ultimate goal was that the pose, as observed by the viewer using the virtual human model, should not have strange, unnatural artifacts such as body parts intersecting into each other. The developed algorithms are sensitive to these constraints, thus giving a lot of importance to the ultimate displacement of the body joints as a result of lossy MoCap compression.

The developed algorithms contributed to the state-of-art of power and network resource constrained MoCap compression. BAP-Sparsing has been shown to be useful for low bitrate MoCap data streaming; BAP-Sparse-Indexing has shown to be effective for MoCap data use in power constrained devices.

6.1.2. Contribution to Layered video encoding

The developed content aware multi-resolution algorithms, FMOE-MR (Features, Motion and Object Enhanced Multi Resolution) video encoding, and MMR (Masked based Multi Resolution) image encoding technique, have been strongly driven by my aim to preserve the aesthetic value of a video or image sequence. There is, of course, no objective manner to describe aesthetics. Personally, I feel aesthetics in an image, or image sequence (video), lies in the details preserved for *visually important* regions of the video/image. The natural question is the following: what are the *visually important regions* for, say, a video sequence? I have observed that, if objects are in motion in a video, they capture the maximum attention of the viewer; the surrounding gets less attention. Similarly, for a more still setting, viewers tend to decipher objects contents in the scene (image/video) by the outlines, and basic texture of the object. Details are important to preserve more information, but can often be removed if the corresponding transcoding benefits are significant. The algorithms, FMOE-MR and MMR, are essentially based on these observations.

Another fundamental contribution is the use of the observation that it is possible to truncate DCT coefficients (in its frequency space) for the compressed version of a video/image by blurring the image in its color space. Although this observation had been known, its use had, to the best of my knowledge, never been done before for multi-resolution image/video re-rendering. My major contributions to the proposed novel algorithms are the use of strong content awareness, coupled with this fact, which led to a new direction of video super-compression, without having to change existing codecs. These algorithms have proven to be very effective for network bandwidth adaptive video encoding for resource constrained environments.

6.1.3. Contribution to generative layered video encoding

Hybrid Layered Video (HLV) has been one of my most significant contributions to the field of power-adaptive generative video encoding for resource constrained environments. One of the novel contributions of the research work for this dissertation is the Generative Sketch Video (GSV). The use of a sparse set of curve, compactly represented by their break points, and temporally evolved to exploit temporal coherency dramatically, has been quite a breakthrough. The motivation for developing GSV technique has been the fact that outlines of the objects in a video sequence are very important features that are naturally tracked by the human eye in order to get information about the object and its activities. A certain aesthetic perspective had allowed me to think of the boundaries as a sparse set of curves, instead of necessarily closed contours, and their complex evolving mechanism, as was popular in literature. GSV was further enhanced by adding a layer of very low quality version of the original video, which does the work of adding approximate color information, which, to most observers, is surprisingly effective, yet at a much less computation decoding price compared to other standard video encoding techniques. This technique was termed by me as Ligne-Claire video, inspired from the works of Herge, in “The Adventure of Tintin”. The HLV encoding combines GSV, Ligne-Claire and FMOE-MR, to form an effective layered video encoding scheme, suitable for power adaptive video playback. HLV has been a unique contribution to the area of power adaptive layered video encoding.

6.1.4 Contribution to layered video dissemination

The algorithms developed for MoCap, video and image transcoding, have benefits for both power constrained, and network bandwidth constrained, environments. The dissertation title of multimedia transcoding for resource constrained (power and network bandwidth) environments, is thus shown to be well justified. For completion, I have contributed to the state-of-art in Internet dissemination of the developed layered video techniques, such as FMOE-MR combined with MPEG FGS (Fine Grained Scalability) profile, HLV and video personalization services. The novel cache replacement policies, LGDS (Layered Greedy Dual Size), and NCCS (Number of Common Clients Size), are well suited for caches specializing in caching layered videos. The caches show improvement over existing standard cache replacement policies, which again is a significant contribution in the state-of-art of cache replacement policies used in the Internet today.

6.1.5. Overall: contribution to multiple domains

Overall, my contribution to the field of multimedia transcoding has been broad, ranging from computer graphics, video, images, and also dissemination of multimedia over the Internet. Each sub-domain, by itself, has a formidable amount of existing work in transcoding, compression and dissemination. My contributions to these multiple sub-domains have put light on the possible use of content-aware themes in to multimedia encoding of any type. I have contributed to a complete spectrum of multimedia data usage; from the creation of compressed multimedia content ready for use, to its dissemination. I have contributed to a new, rising school of thought, where the "meaning" of the digital data is analyzed and utilized for compaction, as opposed to statistical data compaction, which is not typically aware of the "meaning" of the data.

In a broader sense, my contribution in this dissertation is a small step towards making computers *intelligent*, by processing data by their contents, rather that their form, similar to what humans do.

6.2 Dissertation Conclusions

This dissertation reports several novel algorithms and transcoding methods for compact representation of multimedia data suitable for power -and -network constrained environments. Special emphasis on content-aware techniques has been given. Multimedia sub domains of computer animation, videos and images have been considered. The first sub-domain considered is motion capture (MoCap) data used for computer animation of virtual human life characters. Several novel compression methods have been reported. Results show that the proposed algorithms lead to more compact file representations compared to existing standard MoCap compression algorithms, with the same amount of data loss, and also consume less battery power compared to MoCap data compressed at similar bitrates by other standard MoCap compression algorithms. Several image and video transcoding algorithms have been proposed, which transcode video as layers in order to allow power - and - network bandwidth adaptive video playback and dissemination. Typical to the overall theme of the dissertation, strong power-aware concepts have been utilized to create novel, creative methods to transcode video. Results have shown remarkable power - and network bandwidth - adaptive capabilities of the videos, which surpass performance of existing standards of layered video encoding. Finally, efficient caching schemes have been proposed and implemented in order to efficiently disseminate layered video, created using the proposed technologies to power - and network bandwidth - constrained clients, over the Internet. Thus, a complete spectrum, starting from multimedia data transcoding, to multimedia dissemination over the Internet, has been addressed.

The knowledge and techniques from a wide range of techniques from the domains of computer vision, image processing, data compression, distributed computing, computer networks and power aware concepts of system architectures, have been used. Finally, I believe that the proposed technologies addressed in this dissertation have benefited the field of computer science, in its understanding of using content information for multimedia transcoding. The novel algorithms reported in this dissertation are small steps towards making multimedia processing in computers *intelligent*, by processing data by their contents rather that their form, similar to what humans do.

6.3. Future directions

The work in this dissertation has, to my belief, opened the door for a wide range of exciting projects which can be based on, or be extensions of, the reported technologies. I will now mention possible extensions to the work described in this report. Each sub-section is dedicated to possible future work based on my current work.

6.3.1. Future work on MoCap compression

The algorithms presented in this report for MoCap compression are based on structural hierarchy in the model. I believe that it is also possible to take into account more advanced form of information, such as motion between the joints. This might be useful content information, as high speed motions might require lower accuracy, and may sustain more loss in data, compared to lowed speed motion, as high speed motion is difficult to see by the common human observer; hence, errors can be detected less. Another useful research direction is the integration of multiple motion capture data as a single data unit, and processing the entire unit together for compaction. This might lead to smarter compaction methods.

6.3.2. Future work on content-aware multi-resolution layered video

The layered video concepts used in chapter three are based on the standard MPEG Fine Grained Scalability (FGS) profile. A good future direction is to adapt the technologies reported for other scalable video encoding techniques, such as those based on wavelet transformations. Another research direction is to improve the encoding speed of layered video using FMOE-MR (one of the technologies reported here) such that FMOE-MR based layered video can be generated in near real time.

6.3.3. Future work on HLV

An important area of content-aware video encoding can be the integration of the proposed Ligne-Claire video encoding into mainstream MPEG encoding. I believe this is a non-trivial task, and would require smart use of existing MPEG parameters to smartly replace the Ligne-Claire parameters. This would ensure that Ligne-Claire can be used by standard mainstream media players. An important future work on the hybrid layered video (HLV) technique that has been proposed in this report will be to make the layers of HLV additive, so that one layer can be *derived* form a lower layer. This will improve the cache performance of the caches which are used to cache HLV for Internet dissemination.

6.3.4 Future work on layered video caching

The reported layered video caching techniques have been designed to work on single server, single cache and multiple client scenarios. A natural extension to these works is to consider multiple servers, and multiple caches distributed over the Internet. Cache replacement policies for layered video caching have been proposed; in addition, other important aspects such as cache coherency protocols for layered video caching can also be addressed as an extension to these works.

REFERENCES

Akbar, M.D., Manning, E.G., Shoja, G.C., and Khan, S., "Heuristic Solutions for the Multiple-Choice Multi-dimension Knapsack Problem", *Proc. of Intl. Conf. Computational Science*, 2001, pp. 659-668.

Alexa, M., and Muller, W., "Representing animations by principal components", *Proc. of EUROGRAPHICS '00,* 2000, pp. 411-418.

Arikan, O., "Compression of Motion Capture Data", *Proc. of ACM Transactions on Graphics (ACM TOG),* Vol. 25, No. 3, pp. 890-897, 2006.

Atallah, M. J., "Linear time algorithm for the Hausdorff distance between convex polygons", *Information Processing Letters*, vol. 17, no. 4, pp. 207-209, 1983.

Aubel, A., Boulic, R., and Thalmann, D., "Animated impostors for real-time display of numerous virtual humans", *Proc. of 1st Int'l Conf. Virtual Worlds,* (VW-98), vol. 1434, Springer, Berlin, pp. 14-28, 1998.

Baldi, P., Frasconi, P. and Smyth, P., "Modeling the Internet and the Web: Probabilistic Methods and Algorithms", *Wiley*, New York, NY, 2003.

Barakonyi, I., Psik, T., and Schmalstieg, D., "Agents that talk and hit back: animated agents in augmented reality," *Proc. of the Third IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR),* Washington D.C Nov 2-5, pp. 141-150, 2004.

Benz, U. C., Hofmann, P., Willhauck, G., Lingenfelder, I., and Heynen, M., **"**Multi-resolution, object-oriented fuzzy analysis of remote sensing data for GIS-ready information", *ISPRS Journal of Photogrammetry and Remote Sensing,* Volume 58, Issues 3-4, pp. 239-258. 2004.

Bertini, M., Cucchiara, R., Del Bimbo, A. and Prati, A., "Object-based and Event-based Semantic Video Adaptation", *Proc. of IAPR International Conference on Pattern Recognition (ICPR 2004),* vol. 4, Cambridge, UK, pp. 987-990, Aug, 23-26, 2004.

Besl P. J., and Jain, R., "Segmentation through Variable-Order Surface Fitting", *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 10, no. 2, pp. 167-192, 1988.

Bouguet, J.Y., "Pyramidal Implementation of the Lucas Kanade Feature Tracker", Intel Corporation, Microprocessor Research Labs; included in the distribution of OpenCV.

Bowden, R., "Learning statistical models of human motion", *Proc. of IEEE Workshop on Human Modeling, Analysis and Synthesis*, 2000.

Brand, M., and Hertzmann, A., "Style machines", *Proc. of SIGGRAPH 2000,* 183–192.

Canny, J., "A computational approach to edge detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 8, pp. 679 - 698, 1986.

Cao, P. and Irani, S., "Cost Aware WWW Proxy Caching Algorithms", *Proc. of USENIX Symp. Internet Technologies and Systems (USITS),* Monterey, CA, Dec. 1997, pp. 193 - 206.

Capin, T.K., Pandzic, I. S., Thalmann, N. M., Thalmann, D., "A dead-reckoning algorithm for virtual human figures", *Proc. of VRAIS '97*, 1997, pp.161 - 169.

Capin, T. K., and Thalmann, D., “Controlling and Efficient Coding of MPEG-4 Compliant Avatars”, *Proc. of IWSNHC3DI'99,* Santorini, Greece, 1999.

Capin, T.K., Pandzic, I.S., and Magnenat-Thalmann, N., “Avatars in Networked Virtual Environments”, John Wiley and Sons, 1999

Capin, T. K., and Thalmann, D., “Controlling and efficient coding of MPEG-4 compliant avatars”, *Proc. of IWSNHC3DI'99*, Santorini, Greece, 1999.

Capin, T., K., Petajan, E., and Ostermann, J., “Very low bitrate coding of virtual human animation in MPEG-4”, *Proc. of IEEE International Conference on Multimedia and Expo (ICME)*, New York, NY, volume: 2, 2000.

Capin T. K., Petajan, E., and Ostermann, J., “Efficient modeling of virtual humans in MPEG-4”, *Proc. of ICME’2000,* New York, NY, July 2000.

Cavazza, M., Martin, O., Charles, F., Mead, S.J., Marichal, X., “Interacting with virtual agents in mixed reality interactive storytelling”, *Proc. of Intelligent Virtual Agents,* Kloster Irsee, Germany, 2003.

Chandra, S., “Wireless network interface energy consumption implications for popular streaming formats”, *Proc. of Multimedia Systems*, vol. 9, pp. 185 - 201, 2003.

Chattopadhyay, S., Bhandarkar S. M. and Li, K., “BAP Sparsing: A novel approach to MPEG-4 Body Animation Parameter Compression”, *Proc. of IEEE International Conference on Multimedia Communications Systems (ICMCS'05),* Montreal, Canada, Aug 2005. pp. 104 - 109.

Chattopadhyay, S., Bhandarkar S. M. and Li, K., “Compression by Indexing: An Improvement over MPEG-4 Body Animation Parameter Compression”, *Proc. of*

ACM Multimedia Computing and Networking (MMCN'06), San Jose, California, Jan 2006.

Chattopadhyay, S., Luo, X., Bhandarkar, S. M. and Li, K., "FMOE-MR: content-driven multi-resolution MPEG-4 fine-grained scalable layered video encoding", *Proc. of ACM Multimedia Computing and Networking Conference (ACM MMCN' 07),* San Jose, California, January, pp. 650404-1- 11, 2007.

Chattopadhyay, S., Bhandarkar, S.M. and Li, K., "Ligne-Claire Video Encoding for Power Constrained Mobile Environments", *Proc. of ACM Conf. Multimedia*, Augsburg, Germany, Sept. 2007, pp. 1036 - 1045.

Chattopadhyay, S., Bhandarkar, S.M. and Li, K., "A Framework for Encoding and Caching of Video for Quality Adaptive Progressive Download", *Proc. of ACM Conf. Multimedia*, Augsburg, Germany, Sept. 2007, pp. 775 - 778.

Cherkasova, L., "Improving WWW Proxies Performance with Greedy-Dual-Size-Frequency Caching Policy", HP Laboratories Report No. HPL-98-69R1, April, 1998.

Cheung, S.-C. and Kamath, C., "Robust Background Subtraction with Foreground Validation for Urban Traffic Video," *Proc. of EURASIP Journal on Applied Signal Processing*, vol. 14, pp. 1-11, 2005. UCRL-JRNL-201916.

Cornea, R., Nicolau, A. and Dutt, N., "Software annotations for power optimization on mobile devices", *Proc. of Conf. Design, Automation and Test in Europe, Munich, Germany*, 2006, pp. 684 - 689.

Cucchiara, R., Grana, C., Prati, A. and Vezzani, R., "Computer vision techniques for PDA accessibility of in-house video surveillance", *Proc. of 1st ACM SIGMM*

International Workshop on Video Surveillance, pp. 87 - 97, Berkeley, California, 2003.

Dai, M., Loguinov, D., "Analysis and Modeling of MPEG-4 and H.264 Multi-Layer Video Traffic", *Proc. of IEEE Infocom*, 2005.

Davies, E., “Machine Vision: Theory, Algorithms and Practicalities”, Academic Press, San Diego, CA, pp. 42 – 44, 1990.

De La Torre, F., and Black, M. J., “A framework for robust subspace learning”, *Intl. Journal of Computer Vision*, vol. 54, 117–142.

Eickeler, S., and Müller, S., “Content-based Video Indexing of TV Broadcast News using Hidden Markov models”, *Proc. of IEEE Intl. Conf. Acoustics, Speech and Signal Processing*, March 1999, 2997-3000.

Elsen, I., Hartung, F., Horn, U., Kampmann, M., and Peters, L., “Streaming Technology in 3g Mobile Communication Systems”, *IEEE Computer*, vol. 34, no. 9, pp. 4652, September 2001.

Endo, M., Yasuda, T., Yokoi, S., “A distributed multi-user virtual space system”, *IEEE Computer Graphics and Applications*, (Vol. 23, No. 1), pp. 50 – 57, 2003.

Feng, W., Choi, J., Feng, W., and Walpole, W., “Under the Plastic: A Quantitative Look at DVD Video Encoding and Its Impact on Video Modeling”, *Proc. of Packet Video*, Nantes, France, April, 2003.

Finkelstein, A., Jacobs, C. E., and Salesin, D. H., “Multiresolution video”, *In ACM Computer Graphics Proceedings, SIGGRAPH'96*, Annual Conference Series, pages 281-290, 1996.

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 Magazine*, September, pp. 23 – 32.

Geusebroek, J.-M., Smeulders, A. W. M. and Van De Weijer, J., “Fast Anisotropic Gaussian Filtering”, *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, no. 3, March 2001.

Giacomo, T., Joslin, C., Garchery, S., and Magnenat-Thalmann, N., “Adaptation of facial and body animation for MPEG-based architectures”, *Proc. of International Conference on Cyberworlds,* pp. 221, 2003.

Glardon, P., Boulic, R. and Thalmann, D., “PCA-based Walking Engine using Motion Capture Data”, *Proc. of Computer Graphics International ‘04, Greece*, pp. 292-298.

Gleicher, M., “Re-targeting Motion to New Characters”, *Proc. of SIGGRAPH 98*.

Goldberg, D.E., “Genetic Algorithms in Search, Optimization, and Machine Learning”, Addison Wesley, 1989.

Gonzalez, R.C. and Woods, R.E., “Digital Image Processing”, Addison-Wesley, Reading, MA, 1992.

Grochow, K., Martin, S. L., Hertzmann, A., and Popovic, Z., “Style-based Inverse Kinematics”, Proc. of *ACM Transactions on Graphics* Vol. 23, No. 3, July 2004.

Gutiérrez, M., Vexo, F., and Thalmann, D., “Controlling Virtual Humans Using PDAs”, *Proc. of the 9th International Conference on Multimedia Modeling* (MMM'03), Taiwan, 2003.

Hakeem, A., Shafique, K., and Shah, M., “An object-based video coding framework for video sequences obtained from static cameras”, *Proc. of the 13th Annual ACM International Conference on Multimedia*, pp. 608 - 617, Singapore, 2005.

Havinga, P. J. M. “Mobile Multimedia Systems”, PhD thesis, *Univ. of Twente*, Feb 2000.

Hennessy, J.L. and Patterson, D.A., “Computer Architecture - A Quantitative Approach”, 4th Edition, Appendix D, Morgan Kaufmann, San Francisco, CA, 2007.

Hernandez, R.P., and Nikitas, N.J., “A New Heuristic for Solving the Multichoice Multidimensional Knapsack Problem”, *IEEE Trans. System, Man and Cybernetics*, Part A, Vol. 35, No. 5(2005), pp. 708-717.

Hijiri, T., Nishitani, K., Cornish, T., Naka, T., and Asahara, S., “A spatial hierarchical compression method for 3D streaming animation”, *Proc. of the Fifth Symposium on Virtual Reality Modeling Language (Web3D-VRML),* Monterey, California, USA, pp. 95 – 101, 2000.

ISO/IEC 14496-1:1999, “Coding of Audio-Visual Objects”, Systems, Amendment 1, December 1999.

ISO/IEC 14496-2:1999, “Coding of Audio-Visual Objects”, Visual, Amendment 1, December 1999.

ISO/IEC 14496-2/FPDAM4, “Coding of Audio-Visual Objects”, Part-2 Visual, Amendment 4: Streaming Video Profile, July 2000.

ISO/IEC “Coding of Audio-Visual Objects”, Part-2 Visual, Amendment 4: Streaming Video Profile, July 2004.

Ivanov, Y., Bobick, A., and Liu, J., “Fast Lighting Independent Background Subtraction”, *International Journal of Computer Vision*, vol. 37, no. 2, June, pp. 199-207, 2000.

Jacobs, C. E., Finkelstein, A. and Salesin, D. H., “Fast multiresolution image querying”, *Proc. of SIGGRAPH'95,* pp. 277-286. ACM, 1995.

Jain, A.K., Duin R.P.W., and Mao, J., "Statistical pattern recognition: a review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 4-37, 2000.

Javed, O., Shafique, K., and Shah, M., “A hierarchical approach to robust background subtraction using color and gradient information”, *Proc. of IEEE Workshop on Motion and Video Computing*, pp. 22-27, 2002.

Jenkins, O. C., and Mataric, M. J., “Deriving action and behavior primitives from human motion data”, *Proc. of IEEE/RSJ Intl. Conference on Intelligent Robots and Systems (IROS),* pp. 2551–2556. 2002.

Jin, S. and Bestavros, A., “Popularity-aware greedy dual-size web proxy caching algorithms”, *Proc. of 20th IEEE Intl. Conf. Distributed Computing Systems (ICDCS),* Taipei, Taiwan, Apr. 2000, pp. 254-261.

Jolliffe, T., “Principal Component Analysis”, *Springer Series in Statistics. Springer-Verlag*. 1986.

Joslin, C., Molet, T., Thalmann, N. M., “Advanced real-time collaboration over the internet”, *Proc. of the ACM Symposium on Virtual reality software and technology,* Seoul, Korea, pp. 25 - 32, 2000.

Kang, S. and Zakhor, A., “Packet Scheduling Algorithm for Wireless Video Streaming”, *Proc. of 12th Packet Video Workshop 2002*, Pittsburg, PA, Apr. 2002.

Kangasharju J., Hartanto F., Reisslein M. and Ross K. W., ”Distribution layered encoded video through caches”, *Proc. of IEEE INFOCOM*, Anchorage, AK, Apr. 2001, pp. 1791 - 1800.

Khan S. and Shah, M., “Object based segmentation of video using color motion and spatial information”, *Proc. of IEEE Conf. Computer Vision and Pattern Recognition*, vol. 2, pp. 746–751, 2001.

Krasic, C., Walpole, J., and Feng, W., "Quality-Adaptive Media Streaming by Priority Drop", *Proc. of 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video* (NOSSDAV 2003), Monterey, California, June 2003.

Kruppa, M. and Krüger, A., "Concepts for a combined use of Personal Digital Assistants and large remote displays", *Proc. of Simulation and Visualization, Magdeburg, Germany,* 2003, pp. 349-361, 2003.

Ku, C.-W., Chen, L.-G., and Chiu, Y.-M., "A Very Low Bit-Rate Video Coding System based on Optical Flow and Region Segmentation Algorithms ", *Proc. of the SPIE Conf. Visual Communication and Image Processing,* Taipei, vol. 3, pp. 13181327, May 1995.

Leacock, C., and Chodorow, M., "Combining Local Context and WordNet Similarity for Word Sense Identification", *WordNet: An Electronic Lexical Database,* C. Fellbaum (Editor), MIT Press, Cambridge, MA, 1998, pp. 265-283.

Li, W. "Overview of Fine Granularity Scalability in MPEG-4 Video Standard", *IEEE Trans. Circuits and Systems for Video Technology*, vol. 11, no. 3, Mar. 2001, pp. 301-317.

Liang C., Mohapatra, S., Zarki, M.E., Dutt, N., Venkatasubramanian, N., "A backlight optimization scheme for video playback on mobile devices", *Proc. of Consumer Communications and Networking Conference*, (CCNC 2006), vol. 3, no. 2, January 8-10, pp. 833 – 837, 2006.

Lim, I. S., and Thalmann, D., "A Key-Posture Extraction out of Human Motion Data", *Proc. of 23rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC '01)*, Istanbul, Turkey, vol. 2, pp. 1167 - 1169. 2001.

Liu, S., Babbs, C. F., and Delp, E. J., "Multiresolution detection of spiculated lesions in digital mammograms," *IEEE Transactions on Image Processing*, Vol. 10, No. 6, pp. 874-884. June 2001.

Luo, X. and Bhandarkar, S.M., “Robust Background Updating for Real-time Surveillance and Monitoring”, *Proc. of International Conference on Image Analysis and Recognition*, Toronto, Canada, September, 2005. pp.1226-1233.

Luo, X. and Bhandarkar, S.M., “Tracking of Multiple Objects Using Optical Flow Based Multi-scale Elastic Matching”, *Proc. of Workshop on Dynamical Vision at the International Conference on Computer Vision*, Beijing, China, October, 2005.

Luo, X., Bhandarkar, S.M., Hua, W., Gu, H. and Guo, C., “Nonparametric Background Modeling Using the CONDENSATION Algorithm”, *Proc. of IEEE International Conference Advanced Video and Signal-based Surveillance (AVSS 2006)*, Sydney, Australia, pp. 13 – 18, November 2006.

Luo, X. and Bhandarkar, S.M., “Tracking of Multiple Objects Using Optical Flow-based Multiscale Elastic Matching”, *Springer Lecture Notes in Computer Science – Workshop on Dynamical Vision 2005/2006*, R. Vidal, A. Heyden and Y. Ma (eds.), Vol. 4358, pp. 203 – 217, 2007.

McNamee, D., Krasic, C., Li, K., Goel, A, Steere, D., and Walpole, J., “Control Challenges in Multi-Level Adaptive Video Streaming”, *Proc. of IEEE Conference on Decision and Control (CDC2000)*, Sydney, Australia, Dec, 2000.

Merialdo, B., Lee, K.T., Luparello, D., and Roudaire, J., “Automatic Construction of Personalized TV News Programs”, *Proc. of ACM Conf. Multimedia*, (Orlando, FL, Sept. 1999), pp. 323 - 331.

Mohapatra, S., Cornea, R., Dutt, N., Nicolau, A., and Venkatasubramanian, N., “Integrated Power Management for Video Streaming to Mobile Handheld Devices”, *Proc. of the 11th ACM International Conference on Multimedia*, Berkeley, CA, pp. 582 – 591, 2003.

Ni, P., Isovic, D. and Fohler, G., "User-friendly H.264/AVC for remote browsing", *Proc. of the 14th Annual ACM International Conference on Multimedia*, 2006.

Preda, M., Preteux, F., “Advanced virtual humanoid animation framework based on the MPEG-4 SNHC standard”, *Proc. of EUROIMAGE International Conference on Augmented, Virtual Enviroments and Three-Dimensional Imaging (ICAV3D'01)*, Mykonos, Greece, pp. 311-314, 2001.

Preda, M., and Preteux, F., “MPEG-4 human virtual body animation”, *in M. Bourges-Sévenier (Ed.), MPEG-4 Jump-Start (Chapter 9),* Prentice Hall, Upper Saddle River, NJ., 2002.

Preda, M., Salomie, A., Preteux, F. and Lafruit, G., “Virtual character definition and animation within the MPEG-4 standard”, in *M. Strintzis, N. Sarris (Ed.), 3D Modeling and Animation: Synthesis and Analysis Techniques for the Human Body (Chapter 2)*, IRM Press, Hershey, PA, pp. 27-69, 2004.

PROJECT WEB PAGE: http://www.cs.uga.edu/~siddh/projects/wpca.html

Radha, H., van der Schaar, M. and Chen, Y., “The MPEG-4 fine-grained scalable video coding method for multimedia streaming over IP”, *IEEE Trans. Multimedia*, vol. 3, pp. 53–68, Mar. 2001.

Rejaie, Handley R., H. Yu, M. and Estrin, D.,”Multimedia proxy cache mechanism for quality adaptive streaming applications in the Internet”, *Proc. of IEEE INFOCOM,* Tel-Aviv, Israel, Apr. 2000, pp. 980-989.

Richardson, Iain E. G. “H.264 and MPEG-4 Video Compression: Video Coding for Next Generation Multimedia”, Wiley, 2004.

Rosin, P. L. and West, G. A. W., "Non-Parametric Segmentation of Curves into Various Representations", *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol 17, no. 12, pp. 1140-1153, 1995.

Rowe, L. A., Patel, K. D., Smith, B., C., and Liu, K., "MPEG Video in Software: Representation, Transmission, and Playback", *Proc. of High Speed Networking and Multimedia Computing*, San Jose, CA, February 1994.

Rowley, H., Baluja, S., and Kanade, T., "Neural Network-Based Face Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, number 1, pp. 23-38, Jan 1998.

Safonova A., Hodgins J., and Pollard N. 2004. "Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces", *Proc. of the 31st Annual Conference on Computer Graphics and Interactive Techniques '04*, ACM Press.

Salembier, P., Marques, F., Pardas, M., Morros, J.R., Corset, I., Jeannin, S., Bouchard, L., Meyer, F., Marcotegui, B., "Segmentation-based video coding system allowing the manipulation of objects", *IEEE Trans. Circuits and Systems for Video Technology*, vol. 7, no. 1, pp. 60-74, Feb 1997.

Salomon, D., "Data Compression: The Complete Reference", Springer-Verlag, Berlin, Germany, (ISBN 0-387-40697-2, LCCN QA76.9 D33S25), 2004.

Samet H., "Application of Spatial Data Structures: Computer Graphics, Image Processing, and GIS", Addison Wesley, Reading, MA, 1990.

Sikora, T., "Trends and perspectives in image and video coding," *Proc. of the IEEE*, vol. 93, no. 1, pp. 6-17, Jan. 2005.

Squared5: http://www.squared5.com

Stemm M., Gauthier P., Harada D., and Katz R. H., “Reducing power consumption of network interfaces in hand-held devices”, *Proc. of 3rd Intl. Workshop on Mobile Multimedia Comm.*, September 1996.

Sudarsky, S., and House, D. 1998. “Motion capture data manipulation and reuse via b-splines”, *Lecture Notes in Artificial Intelligence*, pp. 1537:55 – 69.

Tseng, B.L., and Smith, J.R., “Hierarchical Video Summarization Based on Context Clustering,” *Proc. of SPIE*, Vol. 5242(Nov. 2003), pp. 14-25.

Tseng, B.L., Lin, C.Y., and Smith, J.R., “Video Personalization and Summarization System for Usage Environment”, *Jour. Visual Communication and Image Representation*, Vol. 15(2004), pp. 370–392.

Vacchetti, L., Lepetit, V., Papagiannakis, G., Ponder, M., and Fu, P., “Stable real-time interaction between virtual humans and real scenes”, *Proc. of 3DIM 2003*, Banff, AL, Canada, 2003.

Van der Schaar, M., Chen, Y., and Radha, H., “Adaptive Quantization Modes for Fine-Granular Scalability”, *Contrib. to 48th MPEG Meeting*, pp. m4938, July 1999.

Van der Schaar, M. and Lin, Y.-T., “Content-based selective enhancement for streaming video”, *Proc. of IEEE Intl. Conference on Image Processing*, vol. 2, 2001, pp. 977–980.

Vanderbei, R. J., “Linear Programming: Foundations and Extensions”, Kluwer Academic Publishers, 1997.

Viola, P. and Jones, M., "Rapid Object Detection using a Boosted Cascade of Simple Features," *Proc. of IEEE Conf. Computer Vision and Pattern Recognition*, vol. 1, pp. 511-518, 2001.

Wang, H., Schuster, G. M. and Katsaggelos, A. K., “Rate distortion optimal bit allocation scheme for object-based video coding”, *IEEE Trans. Circuits and Systems for Video Technology*, vol. 15, no. 9, pp. 1113-1123, Sept. 2005.

Wei, Y., Bhandarkar, S.M. and Li, K., "Video Personalization in Resource-Constrained Multimedia Environments", *Proc. of ACM Conf. Multimedia,* Augsburg, Germany, Sept. 2007, pp. 902-911.

Wei, Y., Bhandarkar, S.M. and Li, K., "Semantics-based Video Indexing Using a Stochastic Modeling Approach", *Proc. of IEEE Intl. Conf. Image Processing (ICIP)*, San Antonio, TX, Sept. 2007.

Wei, Y., Bhandarkar, S.M. and Li, K., "Client-centered Multimedia Content Adaptation", *Proc. of ACM Trans. Multimedia Computing, Communications and Applications (ACM TOMCCAP)*, in press.

Wheeler, E.S., "Zipf's Law and Why it Works Everywhere". *Glottometrics*, vol.4(2002), pp. 45-48.

Wilson, R., Todd, M. and Calway, A. D., "Generalised quadtrees: a unified approach to multiresolution image analysis and coding", *SPIE Visual Communication and Image Processing*, pp. 1-2, May 1990.

Wu, D., Hou, Y., Zhu, W., Zhang, Y., and Peha, J., "Streaming Video over the Internet: Approaches and directions", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 1, pp. 1-20, Feb. 2001.

Youtube: http://www.youtube.com

Zarit, B. D., Super, B.J. and Quek, F.K.H., "Comparison of Five Color Models in Skin Pixel Classification", *Proc. of Intl. Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems*, Washington DC, USA, September 1999, pp.58-63.

Zhang, Z. L., Nelakuditi, S., Aggarwal, R., and Tsang, R. P., "Efficient Selective Frame Discard Algorithms for Stored Video Delivery across Resource Constrained Networks", *Proc. of IEEE Infocom 99*, New York, Mar. 1999, pp. 472-479.