B.Comp. Dissertation

Supporting Zooming In Video Playback

By

Ngo Quang Minh Khiem

Department of Computer Science

School of Computing

National University of Singapore

2009/10

B.Comp. Dissertation

Supporting Zooming In Video Playback

By

Ngo Quang Minh Khiem

Department of Computer Science

School of Computing

National University of Singapore

2009/10

Project No: H020580
Advisor: Assoc Prof Wei Tsang Ooi
Deliverables:
Report: 1 Volume
Source Code: 1 DVD

Abstract

In this project, we are studying different approaches for streaming Region-of-Interest over the network - Baseline, Monolithic streaming, Tiled streaming and Hybrid, in which Monolithic streaming and Hybrid are the methods proposed by us. We evaluate the performance of these approaches, in terms of bandwidth efficiency and compression efficiency and explore how encoding parameters packet size, motion vector length and tile size influence the performance of each approach. The experimental results show that tile size is the most influential factor to Tiled streaming while motion vector and packet size have significant effects on the effectiveness of Monolithic streaming. Hybrid, which is a combination of Tiled streaming and Monolithic streaming, is influenced by tile size and packet size.

Subject Descriptors:
H.5.1 Multimedia Information System
H.4.3 Communication Application

Keywords:
Zoom, Video Streaming, Region-of-Interest, MPEG-4

Implementation Software and Hardware:
C, Xvid, FFmpeg

Acknowledgement

I would like to express my gratitude to my supervisor, Prof. Wei Tsang Ooi, who has always been following my progress for the past two years. Under his guidance, I always have motivation to work hard on this project.

I would also like to thank Zoomable Video Project Team, Ravi and Axel, who have made significantly contributions to the progress of this project.

Lastly, I would like to thank my family for having supported and encouraged me all the time.

List of Figures

Table of Contents

Chapter 1

Introduction

The availability of High-definition (HD) cameras has allowed us to capture videos with very high resolutions. The presence of free video-sharing sites such as YouTube, Google video in recent years has increased the users need of sharing video content over the Internet. Besides, the rapid advances in hardware development for mobile devices recently have enabled us to easily watch videos on these portable devices. However, a number of factors have prevented us from watching the original high-resolution videos such as bandwidth bottleneck, limited display size of mobile devices, etc. In other words, the high-resolution videos are not available to us; instead, videos with lower resolutions are given. As such, users may not clearly see the details of their region-of-interest (ROI). This suggests that zooming is possibly a useful operation that a video playing system should support. With zooming, users are allowed to select a ROI and retrieve the higher resolution version of the ROI.

Zooming is a useful operation in many applications. Let us take lecture webcasting as an example. To save network bandwidth as well as to reduce the buffering time, a reduced resolution version of the lecture is sent. Hence, a student may not see what is written on the board. This is getting worse if the lecture is watched on a mobile device. However, with zooming, a student may zoom into the writing on the board and watch it more clearly. Zooming is also helpful when the investigation involves watching and examining the recorded surveillance videos to find the suspects or to see some details. To study whether zooming is a useful operation and frequently used or not, a preliminary user study was conducted by our team. The record of a lecture video was made available to the students on a webpage sometime before the test. The interface on the web allows the students to select a ROI and zoom into to view it with higher resolution. The results have shown that zooming was used about 67 percent of time and the students tended to select a small ROI size to see the details with highest detail level (to see what the lecturer wrote on the board). The usefulness of zooming has motivated us to study the efficient approaches to ROI-based streaming.

1.1 Problem and Our Solutions

Zooming is not new and easy to implement if the video is stored on users local storage. The video player just needs to decode the necessary data of original video for decoding the ROI. The ROI is then scaled down to fit the display size. However, in this project, we are studying a more challenging problem that involves supporting zooming in video streaming in which high resolution source videos are stored at the server side. Due to a number of factors such as network bottleneck, limited display size of clients device, etc. only scaled-down version of source video is sent to clients. As such, clients may not clearly see some fine-grained details of video. Since the original video is not available at client side, zooming in this scaled-down version definitely does not help. Hence, the client should request the high quality version of this ROI from the server. Upon receiving the ROI request from client, the server will look up into the source video and send the ROI and scale it down to fit the clients display size if necessary.

One of the simplest ways to support ROI-based streaming is to encode the ROI as an independent video stream upon receiving ROI request from a client and send this new stream to this client. However, this solution is not scalable if the server is to support a large number of clients or clients frequently change the ROIs. Another way is to pre-encode some ROIs that are most likely viewed by clients. This method only requires encoding of ROIs once and the encoding can be done off-line; however, it is lack of flexibility as users cannot freely select the ROIs and their choices are restricted to ROIs that are available at the server side. The drawbacks of these solutions have motivated us to study for a better method of ROI-based streaming and decoding.

In this project, we are introducing a more scalable and flexible method for ROI decoding called monolithic streaming. Our method can support a number of clients and allows them to freely select the ROIs without any restrictions on ROI sizes or ROI positions. Ideally, to minimize the overhead in the bandwidth, only data of the ROI should be sent. However, due to the dependencies between the ROI and other regions of video frames, sending only the ROI may make it un-decodable. Our approach will analyze the dependencies among macroblocks of video and use dependency lists to keep track these dependencies. Since all dependencies are known and available to the server, the server will send the ROI together with data on which the ROI depends.

Another promising approach to ROI decoding is also proposed - hybrid method, which is the combination of monolithic streaming and tiled streaming, a ROI streaming method similar to that proposed by Feng (Feng, Dang, Kassebaum and Bauman, 2008).

1.2 Contributions

In this section, we will summarize the contributions we have made as well as some results we have achieved in this project. We have proposed a novel ROI-based streaming approach that can analyzes the dependencies among regions of video stream and efficiently send necessary data for decode a ROI in order to save network bandwidth. We aimed to make our solution more scalable by improving the look-up time (the amount of time the server needs to decide which data are necessary for decoding a ROI). We have also introduced an efficient and compact way to store the dependency information for each video stream.

Finally, we evaluate the performance of various ROI decoding methods (monolithic streaming, tiled streaming and hybrid) in terms of compression efficiency and bandwidth efficiency to further understand how encoding parameters (packet size, motion vector length and tile size) affect these methods. Comparisons of efficiency among the method are also investigated to see how well our methods perform when compared to others.

1.3 Report Organization

The rest of the report is organized as follows. Chapter 2 briefly discusses about the research work related to ROI cropping and streaming. Chapter 3 provides some necessary background for this project. Chapter 4 and 5 provide the details about our solutions and other ROI-decoding methods. Chapter 6 discusses about experiment results and Chapter 7 provides summary of our findings and possible directions for our future work

Chapter 2

Related Work

Supporting selection of a Region-of-Interest (ROI) and interaction with ROI has been one of useful operations implemented in many applications. It has also been the subject of research in many computing fields.

A number of research work are interested in automatic detection and tracking of ROI in videos. Many of the approaches are mainly based on the analysis of video content or the amount of motion in video. Fan studied zooming into video frames on mobile devices with limited display (Fan, Xie, Zhou and Ma, 2003). They introduced a novel approach for automatic detection of ROI by using automatic attention-based modeling which takes into account a number of objects. This model also aimed to minimize the amount of interaction between the users and the devices. Another interesting application involving ROI detection and tracking is the prediction of ROI. Mavlankar studied the schemes to predict ROI in high resolution videos that are streamed over the network (Mavlankar, Varodayan and Girod, 2007). The prediction strategies aimed to produce a good prediction of ROI that could satisfy the expectation.

In the context of video coding, the different approaches to efficiently support cropping of ROI in high-resolution videos have been studied. Feng proposed a way of encoding video in a constrained manner to create compression complaints stream such that the compressed video can efficiently supporting cropping in very high-resolution videos, while adding some overhead in compressed file size (Feng et al, 2008). Rehan introduced an efficient technique for frame accurate cropping of MPEG video without affecting the quality of cropped video frames (Rehan and Agathoklis, 2007). The idea behind this technique is to remove the temporal dependencies between cropped video frames and other frames (before or after the cropping points) by transcoding the first frame in the cropped set of frames as I- frame. Some other frames of the GOP need re-encoding in order to re-enforce their dependency on this new I- frame. Bae exploited one of the most striking error resilience tools of recently widely adopted H.264/AVC, Flexible Macroblock Ordering (FMO) to support cropping of multiple ROIs (Bae et al, 2006). FMO allows freely assigning each macroblock to a slice group. In this approach, each ROI is independently encoded as a slice group. However, this approach is lack of flexibility since only a limited set of fixed ROIs are determined and encoded in the bitstream. Consequently, users are not allowed to freely select the ROI size or ROI position. Another work that exploited FMO for streaming of ROI over the network is discussed in (Mavlankar, Baccichet, Varodayan and Girod, 2007). Their study was to identify which the optimal slice size for ROI-based streaming over the network should be. This optimal slice size was achieved through theoretical estimation and then validated through experiments with HD videos of various resolutions.

In this project, we introduce a novel approach for ROI-based streaming which is efficient and flexible. It allows users to freely select the ROIs and can be scalable to a large number of users. Our approach is also aiming to achieve good bandwidth efficiency in order to save network bandwidth.

Chapter 3

Video Compression Background

3.1 Video Frame

From the traditional view, a video sequence is a collection of frames. However, MPEG-4 treats video sequences a collection of video objects and a video object (VO) consists of several video object plane (VOP) which is an instance of VO at a certain time t. Uncompressed video often requires large bitrate and space for transmission and storage. As a result, video compression is necessary to reduce the video size. It comprises of a encoder which produces the original video to a compressed bitstream and a decoder which reconstructs from a bitstream an approximation of uncompressed video. The encoder compresses the video by removing temporal redundancy among video frames and spatial redundancy within a frame. Temporal redundancy refers to the fact that there are similarities between consecutive frames. Hence, the current frame can be predicted from previous frames or future frames and only residual errors (by subtracting reference frame data from the current frame data) are coded. Based on temporal redundancy, typically, video frames can be classified as I- frame, P- frame and B- frame

- I- frame: contains only data of its frame and no reference to data of previous or future frame
- P-frame: Predictive video frame, only contains residual errors and has previous I- or P-frames as its reference frame

- B-frame: Bidirectional frame, similar to P-frame but may have previous or future I- / P-frames as its reference frame.

3.2 Macroblock

A macroblock, a region of 16x16 pixel in a video frame is the basic unit of encoding and decoding process. Encoding comprises the following stages: motion compensated prediction, Discrete Cosine Transform (DCT), Quantization and Entropy Encoding. Motion compensated prediction involves motion estimation, which searchess for a 16x16 region in reference frame that closely matches the macroblock in current frame and motion compensation, which subtracts the reference region data from current macroblock to produce residual errors. After motion compensated prediction stage, a set of residual errors for current macroblock and a motion vector, which is position of reference area relative to position of current macroblock are obtained. Video size is further reduced by processing the residual errors and motion vector processed during the DCT, Quantization and Entropy Encoding stages. However, we are not giving the details on how each of these stages works.

Sometimes the change from previous frame to current frame is significant and hence the residual errors will be large. The encoder may choose encoding the macroblock without motion compensated prediction. Therefore, a macroblock can be coded in intra-mode (coded without motion compensation prediction) or in inter-mode (coded with motion compensation prediction). Macroblocks in an I-frame are intra-mode while macroblocks in a P-frame can be inter-mode or intra-mode. For B-frame, a macroblock can be in forward mode (motion prediction from previous I- or P-frame), backward mode (motion prediction from future I- or P-frame) or interpolative mode (prediction from previous and future I- or P-frames)

3.3 Packet-based Periodic Resynchronization

Since the compressed bitstream is variable-length coded, it is very sensitive to errors when transmitted over the network. Corruption of one or more bits may lead to loss of resynchronization between the encoder and decoder and therefore, it will result in incorrect decoding the rest of the bitstream. To deal with this problem, MPEG-4 Part 2 has introduced some error-resilience tools, one of which is packet-based resynchronization (resync). With packet-based resync, the encoder will insert into the bitstream resync markers after every fixed number of macroblocks or after every K bits in the bitstream. When errors are detected in the bitstream, the decoder will search for the next resync marker and hence, decoding the rest of bitstream is not affected. The encoder can also divide a video frame into multiple video packets, each of which consists of a video packet header followed by a number of consecutive macroblocks. A video packet header has a resync marker inside it. The dependencies among video packets are also removed so that transmission errors in a video packet will not affect the decoding of other packets.

We are interested in the packet-based resynchronization since it provides a robust transmission of bitstream over the error-proned network. And furthermore, we can exploit the lack of independency among the video packets to save bandwidth and decoding time for our project.

Chapter 4

Monolithic Stream

4.1 Classification of Dependency

In this section, we classify different types of dependency that are introduced by most contemporary encoders in general and MPEG-4 in particular.

4.1.1 Motion Vector Dependency

An encoder processes each video frame in unit of macroblock. A macroblock that is coded in INTER-mode refers to some macroblocks in previous I- or P- frame as its reference macroblocks. Therefore, it directly depends on the reference macroblock and prediction errors to reconstruct its image data. We name this the motion vector dependency, which is introduced during the motion estimation/compensation stage of encoding process.

Each reference macroblock may in turn depend on another set of macroblocks. Consequently, for a macroblock to be decodable, its reference macroblocks and those that these reference macroblocks depend on need to be decoded as well. We call the macroblock that some macroblock m depends on parent macroblock of m. Figure 4.1 is an example that illustrates how the dependencies propagate over frames. The macroblock m of frame F is referring to a region that spans over 4 macroblocks of frame F-1. Each of these macroblocks in turn depends on 4 macroblocks in frame F-2. The dependency keeps spreading throughout the frame and the set of macroblocks needed to decode a macroblock may become very large in the worst case.

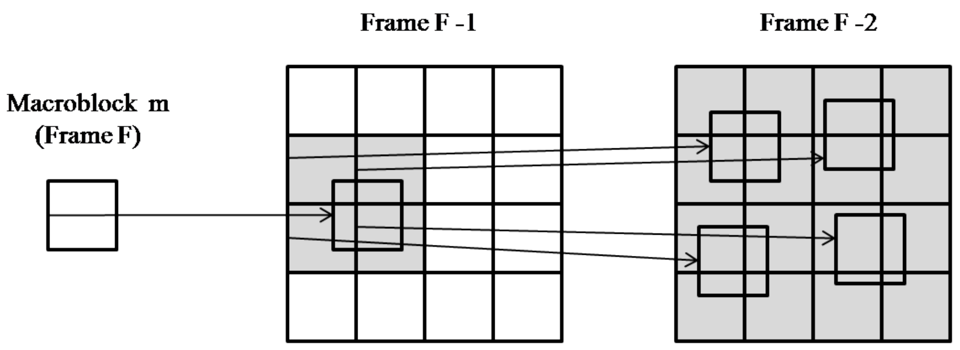


Figure 4.1: Motion Vector Dependency in Monolithic Stream

We also find that the amount of dependency can be further reduced if more analysis on a macroblock and the region on which it depends. The optimization can be achieved based on the following observations. A macroblock m may depend on m for part of m rather than for the whole m. Not decoding the other part (pixel region) of m does not affect data correctness of m since that region is not referred to by m. Hence, to decode part of m, only subset of parent macroblocks of m are required instead of the whole set. These observations will help us to reduce the number of macroblocks necessary for decoding some macroblock and prevent dependencies from unnecessarily spreading. Figure 4.2 shows an example where optimization is applied. Only four macroblocks of frame F-2 are now necessary for decoding macroblock m if which part of a macroblock actually needs decoding is taken into account.

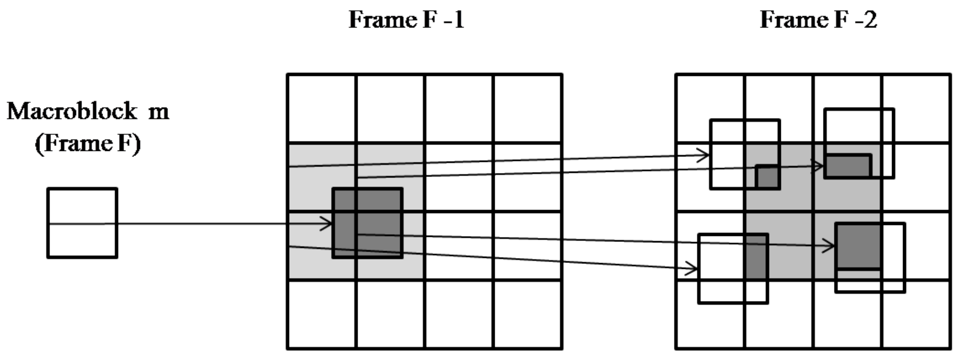


Figure 4.2: Reducing Motion Vector Dependency in Monolithic Stream

4.1.2 VLC Dependency

Since the compressed bitstream is variable-length coded, there are dependencies among macroblocks at the bitstream level. In other words, to get bits of some macroblock m, the bits of macroblocks that come before this macroblock should be properly parsed. They are necessary for the decoder at client side to maintain the syntax of bitstream and extract the bits of m. We call this VLC dependency. It is important to note that the analysis on motion vector dependency of the macroblocks sent due to VLC dependency is not required.

To reduce the VLC dependency among macroblocks in a video frame, we use the packet-based periodic resynchronization mechanism introduced by MPEG-4 Part 2. A video packet is a group of consecutive macroblocks that are independently coded from other video packets in the same frame. Each video packet starts with a resynchronization marker (or packet header) which marks the beginning of a packet. As a result, not sending the bits of packets coming before some packet S does not affect the parsing of bitstream for packet S. Nevertheless, there are still VLC dependencies among macroblocks within the same packet. If a macroblock m that needs to be decoded, all macroblocks before m and in the same packet with m are also sent due to VLC dependency. The remaining bits after m in the packet can be truncated and would not be sent. Truncation of these bits does not affect the parsing of bitstream since the encoder can resynchronize the bitstream by looking for the next packet header of video frame.

Using packet-based resynchronization can help reduce VLC dependencies and hence, improve bandwidth efficiency. The size of packet is one of the most influential factors to data rates of monolithic streams. We evaluate the effect of packet size on monolithic streams in Section 6.2.4.

4.2 Dependency List

In order to keep track the dependency information among the macroblocks, we are using a data structure called dependency list. A dependency list of a macroblock is a list which records positions (x,y) of macroblocks that depend on it. In other words, for every macroblock m, we maintain a list L(m) = {(x, y) — there exists a macroblock n: n at position (x, y) and n depends on m}. Given a ROI and a macroblock m, m should be decoded if and only if its dependency list L(m) has at least a macroblock position that falls inside the ROI. We define look-up time as the amount of time that the server needs to determine whether a macroblock should be decoded or not, given a ROI.

4.3 Look-up Time of Dependency Lists

We maintain the dependency list as a sorted list. For every macroblock in a frame, the server needs to check whether this macroblock is necessary for decoding a ROI. Clearly, if the macroblock is inside the ROI, it should be sent to a client. Otherwise, the server has to scan through the dependency list of this macroblock and check whether there is any position of the list falling within the ROI. We refer to this as the naive checking algorithm.

We have observed that this checking is carried out for all macroblocks in every frame regardless of where and how large the ROI is. Moreover, for macroblocks that are unnecessary for the decoding of ROI, the server still has to traverse through each element of dependency lists to ensure that no positions are inside the ROI. This will unnecessarily increase the look-up time for every frame. As a result, it is desirable to improve the look-up time for macroblocks unnecessary for ROI decoding. In other words, we wish to quickly know a macroblock is not sent without traversing through its dependency list.

Positions in a dependency list form an irregular shape on 2-D plane, which is composed of smaller square macroblock of size 1x1. Checking whether a dependency list overlaps with a ROI is the same as checking whether the shape formed by this dependency list intersects with the ROI. In order to make the checking of intersection faster, we compute the bounding box for each dependency list. Bounding box is the smallest rectangle that encloses the shape formed by a dependency list. As a bounding box and a ROI are both rectangular shapes, checking for intersection is much easier and faster. If bounding box of a macroblock does not intersect with the ROI, this macroblock is not sent. Nevertheless, if there is intersection between the bounding box and the ROI, scanning through the dependency list as before is required.

Using bounding box can help improve look-up time by not traversing through the list if there is no intersection between the bounding box and the ROI. However, the server still needs to start from the beginning macroblock of a frame and go through every macroblock to decide which macroblocks to send, regardless of where the ROI is. We can still further improve the look-up time in a frame by refining the area to search for macroblocks to send to a smaller one.

For each macroblock m and its dependency list L(m), we define "MAX_RIGHT(m)" as the distance from m to the element of L(m) that is farthest away from m and on the right side of m. We similarly define MAX_LEFT(m), MAX_TOP(m) and MAX_BOTTOM(m) to keep track the farthest distance from m to an element of L(m) in other directions. Figure 4.3 is an example that illustrates these four values of a given macroblock m.

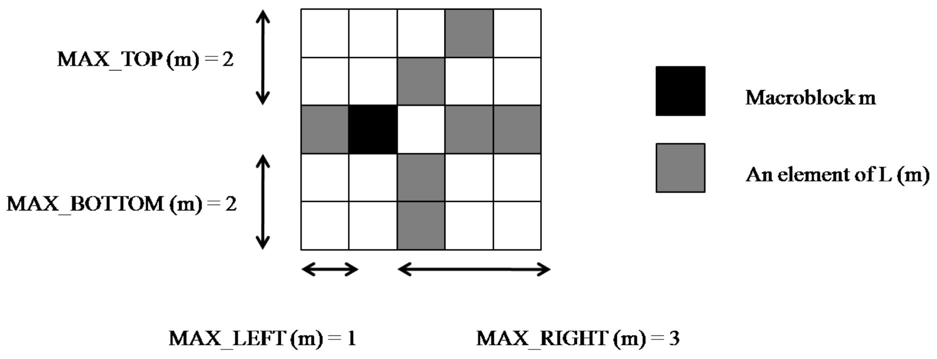


Figure 4.3: Farthest Distance Values of a Macroblock

For each frame F, we define MAX_RIGHT(F) as MAX{MAX_RIGHT(m)} for all m in F. Similarly, we define MAX_LEFT(F), MAX_TOP(F) and MAX_BOTTOM(F). These values will be used to refine the searching area (extended around ROI) for possible candidate macroblocks to send. Figure 4.4 shows how the searching area is extended around ROI.

Our experiment results show that this improved checking algorithm (by using bounding box and refining searching area) is ten times faster than the naive checking algorithm when decoding a ROI of large size (e.g. 60x30 macroblock) and up to thirty times faster than the naive checking algorithm when decoding a ROI of small size (e.g. 15x15 macroblock).

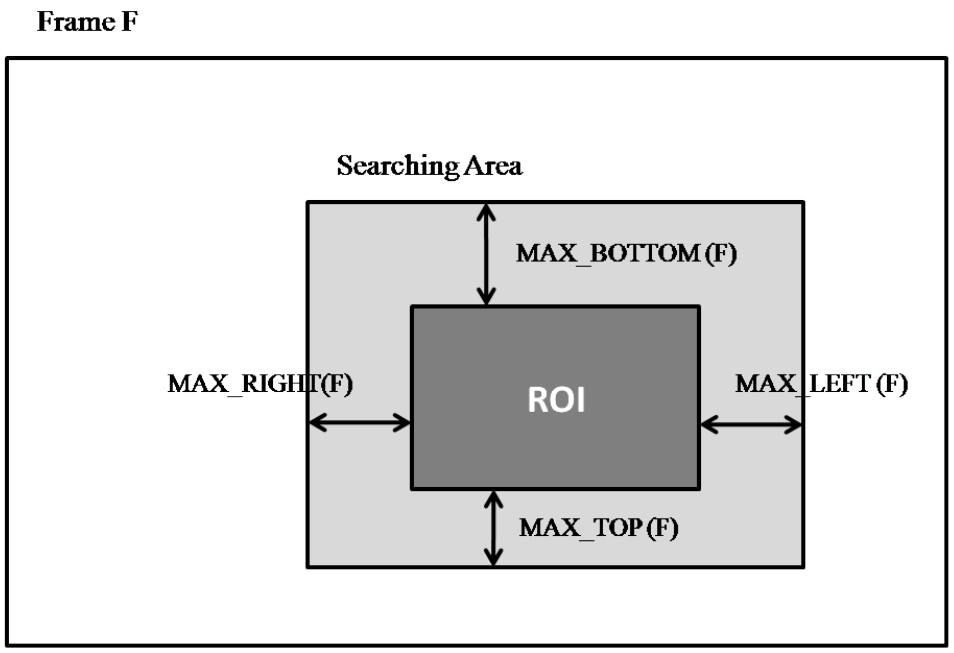


Figure 4.4: Refined Searching Area around ROI

4.4 Storage of Dependency Lists

The server needs to maintain the dependency lists for all macroblocks in video frames. This may require a significant amount of space for the storage of dependency lists if every position (x, y) in the list takes up at least two bytes. We are proposing a compact way to represent the dependency list for each macroblock. In this method, we are using a bitmap to store the dependency list of a macroblock. A bitmap is an array of bits 0 and 1 where 1 indicates that a position should be in the list and 0 means otherwise.

To get the size of the bitmap to store the dependency list of a macroblock, we first need to find the smallest square box that encloses all positions in the list such that this macroblock is located at the center of the box. The bitmap to represent the list has the same size as the square box. The position in the bitmap of the lists element is the same as its position in the square box. It is not necessary to have a bit for the macroblock which is the owner of the list since it is always 1 and located in the middle of the bit-string. Hence, the number of bytes to store the bit string is: (Area of Square Box − 1) / 8. Note that the side of the square box is always an odd number and hence, the term (Area of Square Box − 1) is always divisible by 8.

Figure 4.5 illustrates the steps to represent dependency list as a bit-string.

The reconstruction of dependency list from the bit-string can be easily done since we know the position of macroblock to which the dependency list belongs to, and the number of bits in the bit-string which is the area of the square box. We observe that this compact representation of dependency list requires about five times less storage space than using two bytes for each element of a list.

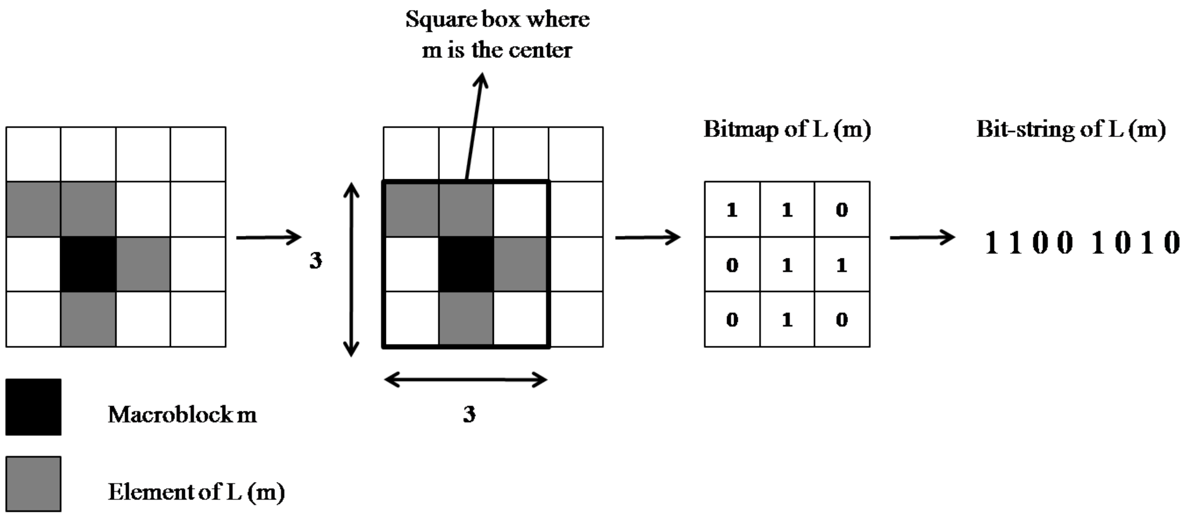


Figure 4.5: Representation of Dependency List as a Bit-string

Chapter 5

Other ROI Decoding Methods

5.1 Baseline

Baseline is one of the simplest approaches to ROI decoding. In this method, the ROI will be encoded as an independent stream and the entire stream will be sent upon receiving ROI request from a user. As such, this method will ideally have less bandwidth wastage and possibly require the least data sent to decode ROI among ROI decoding methods. Due to this fact, this method will be used as the benchmark to evaluate the performance of various ROI decoding methods discussed in following sections.

This method requires two basic steps: cropping and encoding. In the cropping step, given the ROI position (x,y) and ROI size wxh, sub-images of size wxh at position (x,y) of each image from raw video sequence will be grouped together to create a new raw video sequence for the ROI. This new raw video will be encoded by an encoder to produce an independent stream for the ROI. While encoding step may be fast, cropping step takes much more time. Hence, due to its computational inefficiency, baseline method is not often used for real-time ROI decoding. However, it can be employed to pre-encode some popular ROIs which are frequently selected by a number of users since it may lead to least bandwidth wastage. The pre-encoding of ROI helps alleviate the processing time of servers when these ROIs are selected.

5.2 Tiled Streams

Tile streaming is another approach to decode the ROI. This method is motivated by Web-based map service Google-Maps, where maps are divided into grid of smaller images and only images that overlap with ROI will be sent to the user for being rendered. We are employing the similar technique for ROI decoding in video. Accordingly, each image of raw video sequence will be divided into grid of tiles. The tiles of the same (x,y) coordination from all images will be grouped together and encoded into an independent tiled video stream. Upon receiving ROI request from a user, only tiled streams that overlap with the ROI will be streamed to the user.

Clearly, tiled streaming introduces the simplicity at the server side since the server can easily determine the minimal set of tiles to be sent upon receiving the RoI request from clients. Moreover, this method is suitable for publish-subscribe paradigm. In this paradigm, the server needs to multicast each tile to a channel while the client can subscribe to the channels of tiles that intersect with RoI. Nevertheless, tiled streaming would probably pose some difficulties in playback process at the client side. Since a full frame is composed of smaller frames from multiple tiled streams, the video player at client side should be modified so that it can combine frames from different tiles and correctly play back the full video frame. This may result in the high complexity in synchronization and buffering mechanisms of the decoder. To avoid such complexity, Feng et al (2008) has proposed the method to encode a single video stream in which characteristics of tiled streams are imposed. Specifically, motion estimation and entropy coding for a macroblock are confined to be carried out only within the tile. As a result, these constraints have in effect created multiple independently encoded tiled streams within a single stream.

One of the drawbacks of tile streaming approach is the reducing in compression efficiency. Since video frames are broken up into tiles and independently encoded, each tile stream may not achieve the best compression efficiency since the searching area for motion estimation is limited to the tile rather than the whole video frame. Moreover, there would be more overhead in the number of bits for video header and frame header since each full video frame is consisting of multiple independent frames from all tiles. Hence, this results in the fact that more storage for tiled streams is required at the server side. Besides, tiled streaming also incurs some bandwidth wastage. Since the ROI may not be aligned with tile boundaries, the tiles which partially intersect with ROI are to be fully sent. Hence, some data outside the ROI which are unnecessary for decoding and displaying the ROI are still transmitted and this will waste some bandwidth. Tile size is an important factor that hast influence on the performance of tiled streaming in terms of bandwidth and compression efficiency. The smaller the tile size is, the less redundant data are sent but more space is required to store the tiles. The larger the tile size is, the more data are unnecessarily sent but better compression efficiency is achieved. The appropriate tile size should be chosen to achieve a good trade-off between compression and bandwidth efficiency.

5.3 Hybrid

Hybrid method is a combination of Monolithic Streaming method and Tiled Streaming method. It employs the advantages of both methods to achieve the good bandwidth efficiency. Similar to Tiled Streaming method, the original video is encoded as multiple independent tiled streams. When user requests a ROI, the server will send all the tiled streams that completely fall inside the ROI. For tiled streams that partially overlap with the ROI, instead of sending the whole streams as Tiled Streaming method, Monolithic Streaming method is applied for these streams and only necessary data will be sent. As such, the amount of redundant data sent to user is reduced and hence, there is less bandwidth wastage.

Chapter 6

Experimental Evaluation

6.1 Experimental Setup

We are using two standard HD 1080p video sequences. The first video sequence, Rush_hour, has 498 frames is of type of slow motion. The second video sequence, Tractor, consists of 688 frames and has significant amount of fast motion. For all encoded streams, we fix the quantization scale as 2 for I, P and B frames to make sure that video streams used in our experiments are all of the same high quality. We are also using closed GOP (Group-of-Picture) size of 7 with the frame pattern IBBPBBP for all video streams. The set of motion vectors used in the experiments is 4, 8, 16, 24, 32, 40, 48, 64, 72 (pixel unit) and the set of packet sizes is 64, 128, 256, 512, 1460 (byte unit). Note that the motion vector value is the maximum possible motion vector that can be used in the motion estimation stage of encoding. A shorter motion vector length than the one specified for the encoder is used if the best matching macroblock is found by the former one. All video streams will be encoded using the standard FFMPEG encoder with the video codec set to MPEG-4 which implements MPEG-4 Part 2.

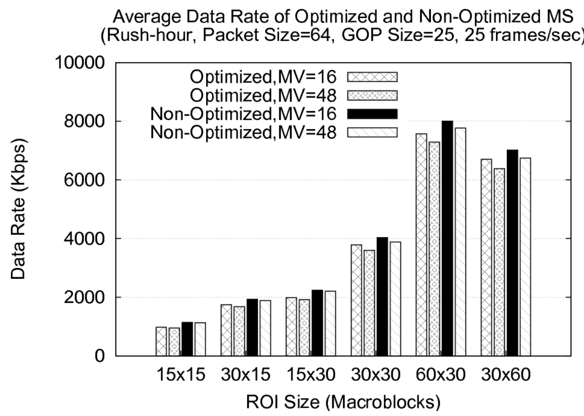
For tiled streaming method, we are using three different tile sizes: 4x4, 8x8 and 16x16 (macroblock unit). The same sets of encoding parameters as above are also used for TS method. For MS method, we are using optimized version for ROI decoding in most of the experiments. The non-optimized version is used when we compare performance between optimized and non-optimized versions.

To simulate various ROI requests from clients, we are choosing ROI widths (macroblock unit) from the set 15, 30, 45, 60, 90, 120 and ROI heights (macroblock unit) from the set 15, 30, 45, 60. All combinations of values from these two sets will result in different ROI sizes. These ROI sizes will be used for ROI decoding methods to evaluate their bandwidth efficiency. Each ROI size will be conducted at five random positions in a frame and the average is taken for that ROI size. We will only show results for 6 ROI sizes (15x15, 30x15, 15x30, 30x30, 60x30, 30x60) in the graphs for conciseness. The other ROI sizes present the similar trends.

6.2 Results

6.2.1 Optimized Monolithic Stream

In this experiment, we compare the bandwidth efficiency of monolithic streams with and without optimization of motion vector dependency. We are using the GOP size of 25 to clearly see the effect of optimized monolithic streams. Figure 6.1 shows data rates of monolithic streams with the packet size of 64 bytes. We observe that the optimized monolithic streams can help reduce the data rate hundreds of kbps. The reducing becomes more significant as the ROI size increases. We are using the optimized version of monolithic streams for other experiments in later sections.



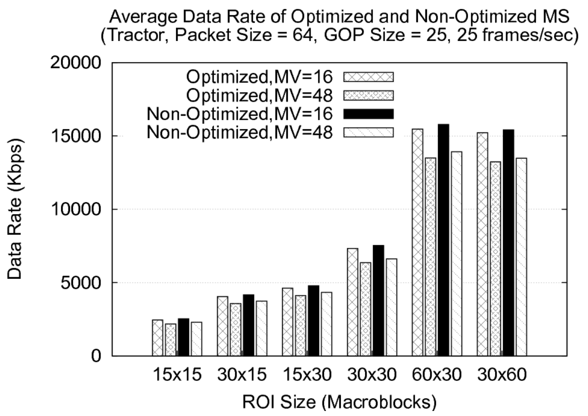


Figure 6.1: Optimized Monolithic Stream

6.2.2 Bandwidth and Compression Efficiency - Tiled Streams

In this experiment, we are studying the influence of tile size on the tiled streaming method in terms of compression efficiency and bandwidth efficiency. Figure 6.2 shows the file size of tiled streams encoded with different tile sizes for rushhour and tractor videos. The experimental results show that larger tile size will lead to better compression efficiency. This is due to the fact that large tile size would result in higher chance to find a better matching macroblocks during motion estimation. As such, the residual errors would be small and require less space to store.

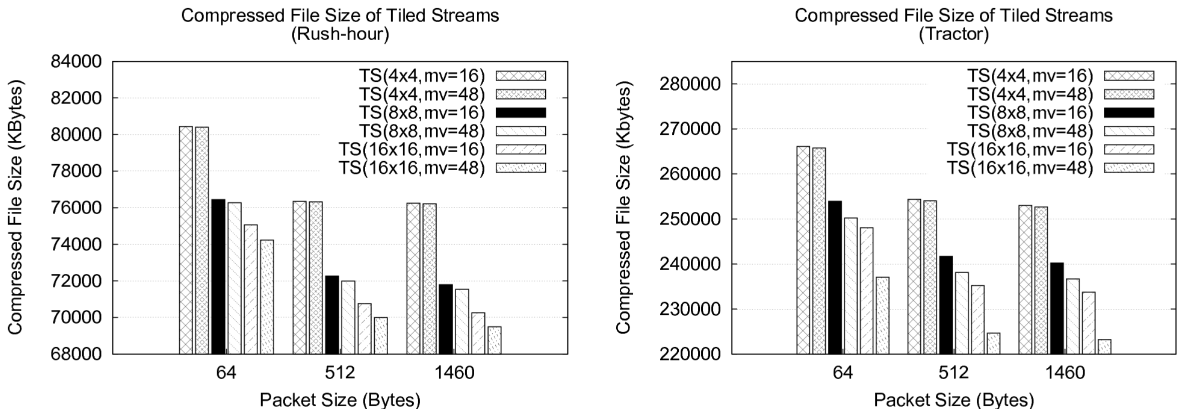


Figure 6.2: Compression Efficiency of Tiled Streams

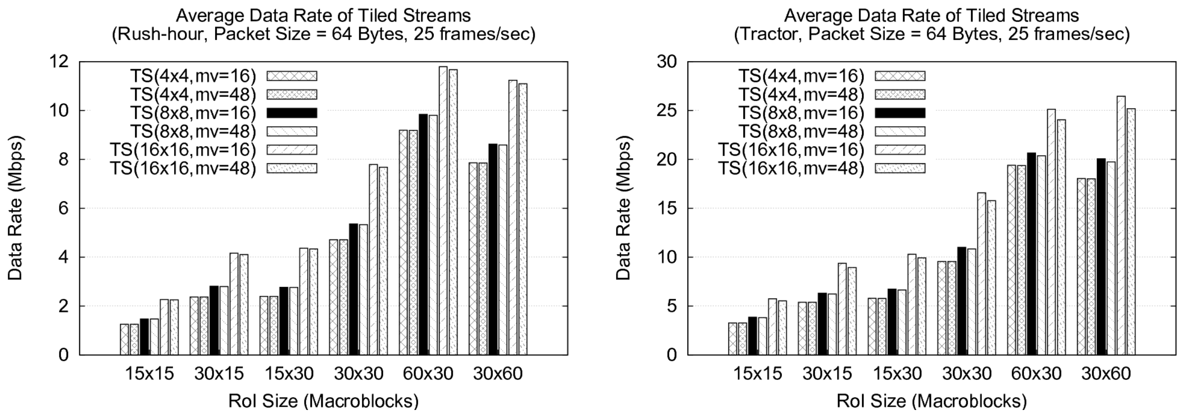


Figure 6.3: Bandwidth Efficiency of Tiled Streams

While increasing tile size gives better compressed file size, it may also lead to more wasted bits to send since tiles that partially overlap with the ROI still have to be fully sent. To see whether the savings in file size can help hide the effect of wastage when large tile size is used, we run the experiment to evaluate the bandwidth efficiency of tiled streams for various tiled sizes and ROIs and the results are shown in Figure 6.3 We observe that the larger the tile size is, the larger amount of data is sent. Thus, the wasted bits are dominating the saving bits in file size when tile size increases. The tile size of 4x4 gives the best bandwidth efficiency while the tile size of 16x16 requires much more data sent.

Using longer motion vector and larger packet length results in better compressed file size and average data rate. Using longer motion vector allows higher possibility for a better motion estimation/compensation while larger packet length reduces the number of packets and hence, reduces the total number of bits used for packet headers. However, the effect of motion vector and packet length on compression efficiency and bandwidth efficiency is not as significant as tile size. The improvement in file size due to longer motion vector (mv=48) is noticeable when the tile size of 16x16, where the searching area is large enough for motion vector 48. To summarize, tile size is the factor that has most influence on the effectiveness of tiled streaming method.

6.2.3 Compression Efficiency - Monolithic Stream

In this experiment, we evaluate the effect of motion vector and packet size on the compression efficiency of monolithic streaming. Figure 6.4 shows the file sizes of monolithic streams encoded with various motion vector lengths and packet sizes.

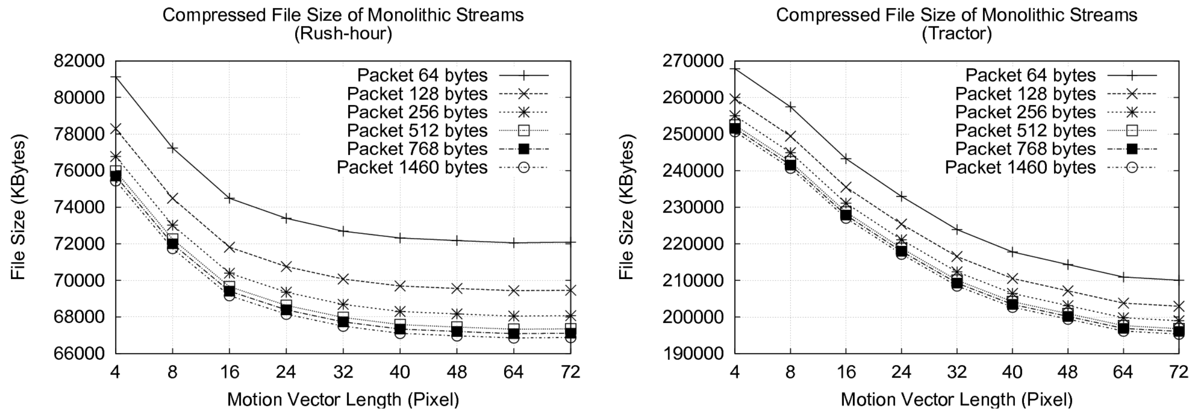


Figure 6.4: Compression Efficiency of Monolithic Stream

We can observe that increasing motion vector length and packet size will yield better compression efficiency. Allowing the encoder to use longer motion vector may result in better motion estimation while larger packet size will reduce the total number of bits used for headers of packets. However, the decrease in compressed file size due to increasing of motion vector length is more significant than the decrease due to increasing of packet size. This shows that motion vector is more influential to the compression of monolithic streams than packet size. For tractor video, better compression can still be achieved when motion vector of 48 is used while compressed file size almost remains constant beyond motion vector of 32 for rushhour video. Since there is less motion in rushhour video, increasing motion vector length beyond 32 may not help reduce file size further at all.

Figure 6.5 shows the comparison of file size between monolithic streams and tiled streams for different motion vector lengths and packet sizes. Monolithic streams achieve better compression efficiency than tiled streams and the influence of motion vector in monolithic streams is more significant. As mentioned in the previous section, increasing motion vector length does not improve the compression ratio of tiled streams much since the motion estimation and compensation is constrained within the tile.

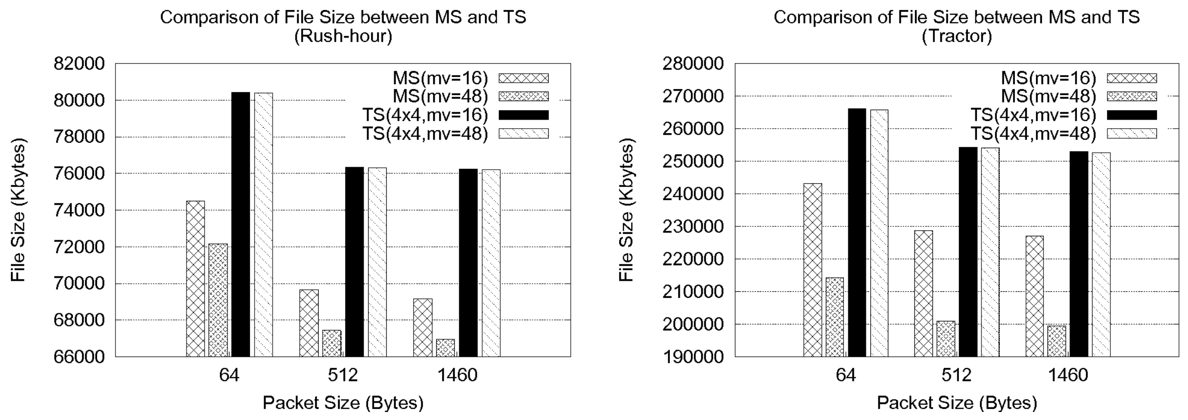


Figure 6.5: Compression Comparison between Monolithic and Tiled Streams

6.2.4 Bandwidth Efficiency - Monolithic versus Tiled Streams

In this section, we compare the bandwidth efficiency between two methods. While it is clear that monolithic streaming gives better compression efficiency than tiled streaming, we do not know which method achieve better bandwidth efficiency. We also investigate how changing the encoding parameters (motion vector and packet size) will affect the data rate of monolithic streams.

Figure 6.6 shows the data rates of both methods with small packet size being used (64 bytes). We can observe that monolithic streams require lower data rate than tiled streams and the difference in data rate is getting more significant when the ROI size increases. As the ROI size increases, more tiles that partially overlap with the ROI are sent and hence, introduce more redundant macroblocks being transmitted. The difference is also more significant in tractor video since there is much motion in this video and more encoded data are needed for macroblocks.

The data rate of monolithic streams is significantly increasing if packet size of 1460 bytes is used as shown in Figure 6.7. The data rate of tiled streams is now much lower than monolithic streams. Using large packet significantly degrades the bandwidth efficiency of monolithic streams since it results in more VLC dependency data to be sent.

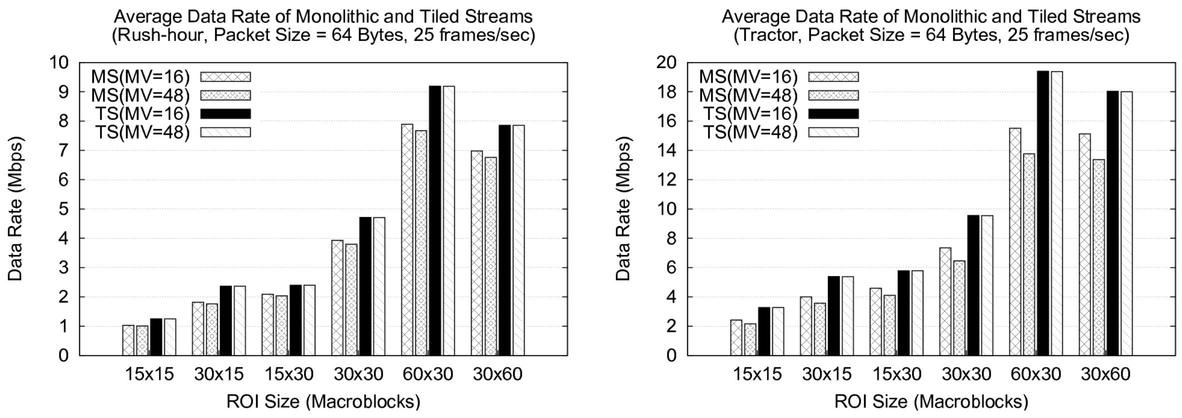


Figure 6.6: Bandwidth Comparison between Monolithic and Tiled Streams (Packet Size = 64 Bytes)

While both longer motion vector and larger packet size can reduce the file size, only using longer motion vector helps improve bandwidth efficiency. Although long motion vector may introduce more motion vector dependencies, its improvement in compression efficiency can still help achieve good bandwidth efficiency. Besides, it is better for monolithic streams to be encoded with small packet sizes so that lower data rates can be achieved for ROI decoding while a slightly increase in file size is acceptable.

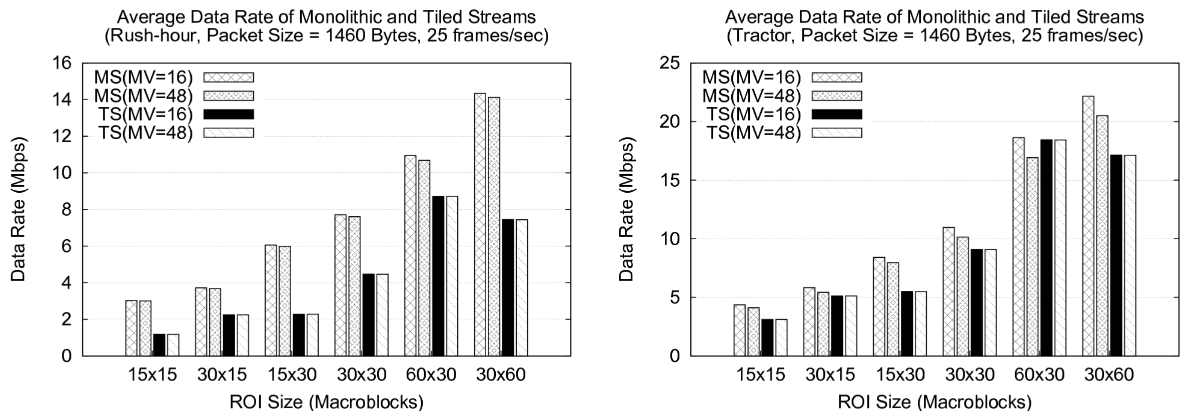


Figure 6.7: Bandwidth Efficiency of Tiled Streams (Packet Size = 1460 Bytes)

6.2.5 Bandwidth Efficiency - Hybrid Method

In this section, we want to understand how tile size and packet size affect the bandwidth efficiency of hybrid method. While it is clear that large tile size implies that more redundant data will be sent for TS method, this fact may not be true for hybrid method as MS is employed to help reduce the amount of unnecessary data sent for partially overlapped tiles. Besides, since packet is one of the important factors that significantly affect the bandwidth efficiency of MS method, we need to evaluate its role in the performance of hybrid method. Since longer motion vector gives better bandwidth efficiency for both tiled streams and monolithic streams, we can expect the same effect on hybrid method. Thus, we are using the motion vector 48 in this experiment.

Figure 6.8 shows the data rates of hybrid method when different tile sizes are used and the tiled streams are encoded with packet size of 64 bytes. We can see that the data rates required for tiled streams with tile size of 4x4 are slightly higher than streams with other tile sizes. The data rates required for tiled streams with tile sizes of 8x8 and 16x16 are roughly the same. This fact shows that hybrid method has effectively reduced the amount of redundant data sent for tiled streams of large tile sizes while it still makes use of good compression efficiency achieved by large tile sizes.

Figure 6.9 shows a different scenario where the packet size of 1460 bytes is used. The data rates are increasing when tile sizes increase. However, the difference in data rates between tiles

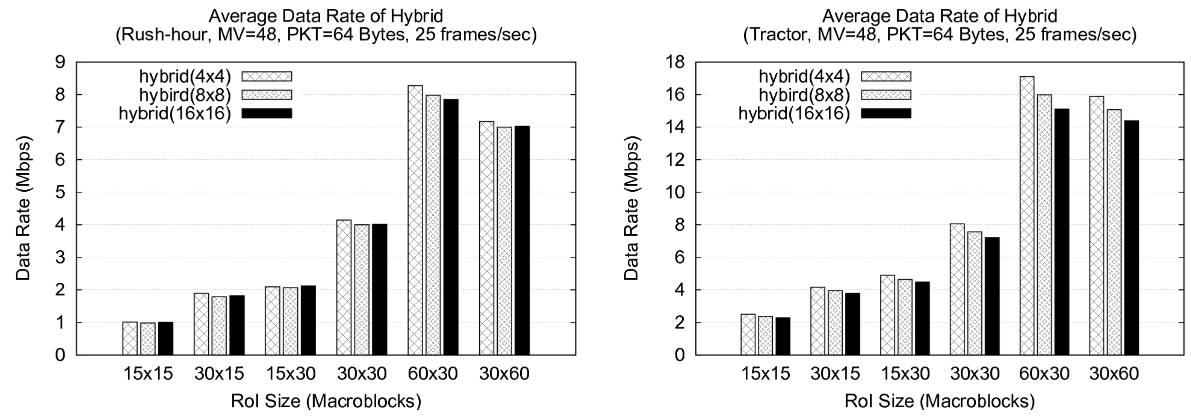


Figure 6.8: Bandwidth Efficiency of Hybrid Method (Packet Size = 64 Bytes)

size of 16x16 and other tile sizes is less significant than that introduced in tiled streams. This is as expected since monolithic streaming is employed to reduce the amount of redundant data being transmitted.

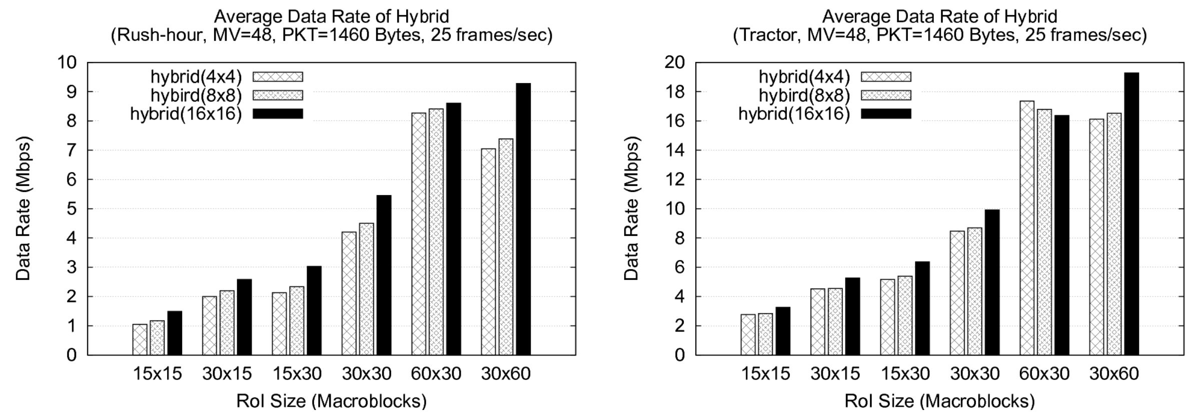


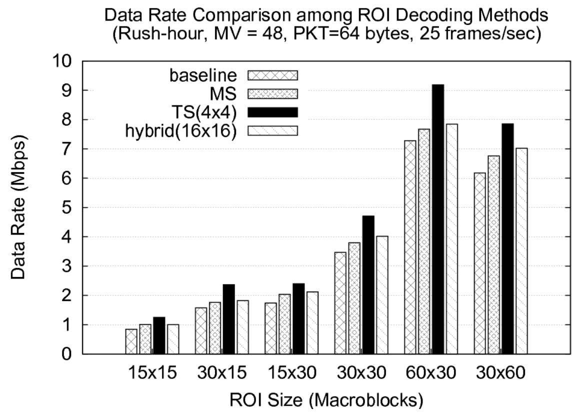
Figure 6.9: Bandwidth Efficiency of Hybrid Method (Packet Size = 1460 Bytes)

While tile size is a major factor that affects the performance of TS method, its influence on hybrid method is controlled by packet size. Using large packet size leads to more data transmitted for streams of the same tile size. The increase in data rates due to large packet sizes is significant for large tile size and this makes large tile size not able to achieve as good bandwidth efficiency as smaller tile sizes. Nevertheless, using small packet size helps streams of large tile size achieve as good as or even better bandwidth efficiency than streams of smaller tile sizes. And the effect of tile size on bandwidth efficiency of hybrid method in this case is opposed to tiled streams.

6.2.6 Comparison among ROI decoding methods

This section basically gives an overall comparison among the methods discussed in the previous sections. This will help us have an overview of the bandwidth efficiency of each method relative to others under different encoding parameters. In theory, baseline method would achieve the least bandwidth wastage since each ROI is encoded as an independent stream and no data outside the ROI is sent. However, it is not clear whether the baseline method results in the least amount of sent data among the methods since the search range of motion estimation is constrained within the ROI. MS method performs well when small packet size is used for encoding. Hybrid method can result in good bandwidth efficiency for either large or small packet sizes, depending on which tile size is used. TS method only performs well if small tile size is used.

In this experiment, two different scenarios are presented. The first scenario involves using encoding parameters PKT64, MV48 which result in good performance for MS method and hybrid method with tile size 16x16. The second scenario uses encoding parameters PKT 1460, MV48 which make the MS method and hybrid method with tile size 16x16 poorly perform (but hybrid with tile size 4x4 well performs in this case). Figure 6.10 and Figure 6.11 show the experiment results for rushhour and tractor videos under two different scenarios.



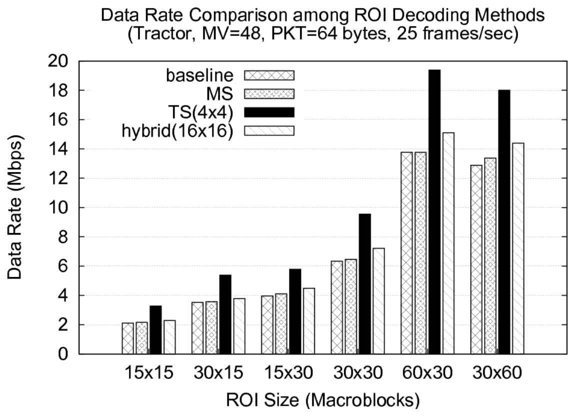
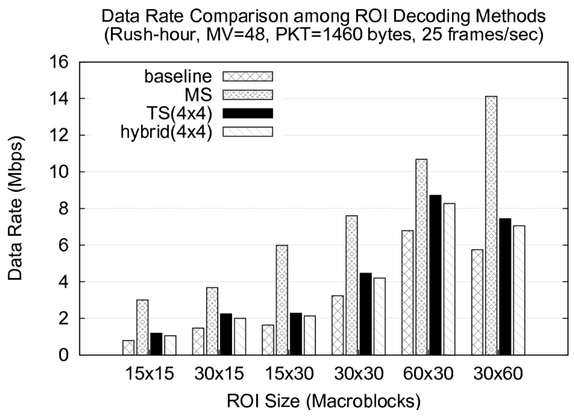


Figure 6.10: Bandwidth Efficiency Comparison among Methods (Packet Size = 64 Bytes)

Under any scenarios, baseline method has the best bandwidth efficiency among the methods in most cases. We can observe that in the first scenario, the amount of sent data for MS



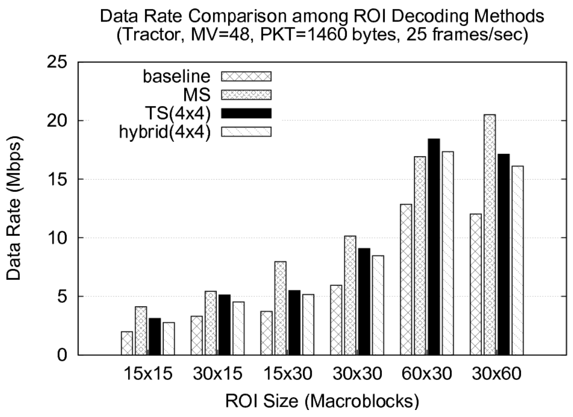


Figure 6.11: Bandwidth Efficiency Comparison among Methods (Packet Size = 1460 Bytes)

method is just slightly higher than baseline method and much better than hybrid method. This suggests that ROI decoding using MS method for video streams coded with small packet size can achieve very good bandwidth efficiency and the amount of bandwidth wastage is not significant. In the second scenario, we observe the poor performance of MS method and this is as expected. Except for baseline, hybrid with tile size 4x4 achieves the best bandwidth efficiency among the methods. However, the amount of sent data by hybrid4x4 is still much more than baseline method in tractor video.

Chapter 7

Conclusion

7.1 Summary

In this section, we will summarize the results we have achieved in this project. We have explored various ROI streaming approaches (naive approach baseline, monolithic streaming, tiled streaming and hybrid) and investigate how encoding parameters (motion vector length, packet size, and tile size) influence the performance of these methods in terms of bandwidth and compression efficiency.

Tile size is the most influential factor to tiled streams. Large tile sizes lead to better compression efficiency but at the same time, they will result in more redundant data to be sent. Small tile sizes increase the file size but can give good bandwidth efficiency. Long motion vector and large packet size help improve compression efficiency of monolithic stream. While long motion vector results in lower data rate for monolithic stream, large packet size will significantly increase the data rate of monolithic stream, mainly due to the increasing in VLC dependency. Small packet size is leading to very good bandwidth efficiency for monolithic stream and hence, is recommended.

Hybrid method incorporates the monolithic streaming to reduce the amount of redundant data introduced in tiled streaming, especially when large tile is used. Our experiment shows that the data rates of hybrid method are significantly better than tiled streams. It is interesting to note that if small packet size is used, large tile size (e.g. 16x16) even achieves better bandwidth efficiency than small tile size (e.g. 4x4). However, for large packet size, data rates of hybrid with tile size 16x16 are still higher than hybrid with smaller tiles sizes.

The bandwidth comparison among the methods has shown that monolithic stream performs very well if video stream is encoded with small packet size. Its data rate is close to the data rate of baseline method and even lower than hybrids data rate. For either small or large packet sizes, hybrid method can still achieve good bandwidth efficiency if appropriate tile size is chosen (depending on packet size).

7.2 Future Work

In this project, we have introduced the new approach for ROI streaming called monolithic stream. We have also come up with the data structure, the dependency list to keep track the dependencies among macroblocks and how to achieve fast look-up time which is very important if the server is to support a large number of clients. The experiment has shown that our method can achieve good bandwidth efficiency if small packet size is used. However, its performance is significantly degrading for large packet sizes which introduce more VLC dependency data to be sent. Hence, it is desirable to make our approach sustainable to large packet sizes. A promising solution is to do VLC trans-coding in order to remove VLC dependency between decoded macroblocks and other (un-decoded) macroblocks in a packet. As such, no VLC dependency data should be transmitted. Specifically, the packet header and data of decoded macroblocks in the packet will be grouped together and variable-length coded (the modification of packet header may be required). VLC trans-coding might not be difficult if it is done off-line. However, in the context of streaming, VLC trans-coding must be done in real-time and thereby would be very challenging

References

Bae, T.M, Thang, T.C, Kim, D.Y, Ro, Y.M, Kang, J.W, Kim, J.G. Multiple Region-of-Interest Support in Scalable Video Coding. *ETRI Journal*, 28(2), 2006.

Fan, X., Xie, X., Zhou, H. and Ma, W. Looking into video frames on small displays. In *MULTIMEDIA '03: Proceedings of the eleventh ACM international conference on Multimedia*, pages 247–250, New York, NY, USA, 2003. ACM.

Feng, W., Dang, T., Kassebaum, J. and Bauman, T. Supporting region-of-interest cropping through constrained compression. In *MULTIMEDIA '08: Proceedings of the 16th ACM international conference on Multimedia*, pages 745–748, Vancouver, British Columbia, Canada, 2008.

Huang, J., Feng, W. and Walpole, J. An Experimental Analysis of DCT-based Approaches for Fine-grained Multiresolution Video. *Multimedia Systems*, 11(6):513–531, June 2006.

Liu, F. and Gleicher, M. Video retargeting: automating pan and scan. In *MULTIMEDIA '06: Proceedings of the 14th annual ACM international conference on Multimedia*, pages 241–250, New York, NY, USA, 2006. ACM.

Mavlankar, A., Baccichet, P., Varodayan, D. and Girod, B. Optimal slice size for streaming regions of high resolution video with virtual pan/tilt/zoom functionality. In *Proc. 15th European Signal Processing Conference, EUSIPCO'07*, pages 1275–1279, 2007.

Mavlankar, A., Varodayan, D., and Girod, B. Region-of-Interest Prediction for Interactively Streaming Regions of High Resolution Video. In *Proc. International Packet Video Workshop, PV2007*, pages 68–77, Lausanne, Switzerland, Nov. 2007.

Pea, R., Mills, M., Rosen, J., Dauber, K., Effelsberg, W., and Hoffert, E. The Diver Project: Interactive Digital Video Repurposing. *IEEE Multimedia*, 11(1):54–61, Jan-March, 2004.

Rehan, M. and Agathoklis, P. Frame-accurate video cropping in compressed mpeg domain. In *Proc: PacRim 2007*, pages 573–576, 2007.

Segall, C.A and Sullivan, G.J. Spatial Scalability within the H.264/AVC Scalable Video Coding Extension. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1121–1135, Sept,2007.