Supporting Zoomable Video Streams with Dynamic Region-of-Interest Cropping

Ngo Quang Minh Khiem, Guntur Ravindra, Axel Carlier, and Wei Tsang Ooi
Department of Computer Science
National University of Singapore
Singapore 117417
Email:{nqmkhiem,ravindra,carlier,ooiwt}@comp.nus.edu.sg

ABSTRACT

Streaming of an arbitrary region of interest (RoI) from a high resolution video is essential to supporting zooming and panning within a video stream. This paper explores two methods for RoI-based streaming, referring to them as *tiled streaming* and *monolithic streaming*. *Tiled streaming* partitions video frames into grid of tiles and encodes each tile as an independently decodable stream. *Monolithic streaming* applies to video encoded using off-the-shelf encoder, and relies on a pre-computed dependency information to send the necessary bits for the RoI. We evaluated these two methods in terms of bandwidth efficiency, storage requirement, and computational costs under different video encoding parameters. Experimental results show that bandwidth efficiency of tiled streams for RoI-based streaming reduces when tile size increases, despite improvement in compression efficiency. In the case of monolithic streams, use of a larger motion vector range coupled with careful run-time optimization can still improve the bandwidth efficiency, despite an increase in motion vector dependency.

Categories and Subject Descriptors: H.5.1 [Multimedia Information Systems]: Video;H.4.3 [Communications Application]

General Terms: Design, Performance.

Keywords: Zoom and Pan, Video Streaming, Region-of-Interest, Video Cropping.

1. INTRODUCTION

As video capturing devices become increasingly capable of capture at higher resolutions, we see an opposing trend in reduction of display sizes, mainly driven by the proliferation of mobile devices including portable media players, mobile phones, and PDAs. Even as screen resolution on mobile devices improves, due to the physical limitation of the size of the device and human inability in differentiating details beyond a certain pixel density, display resolution on mobile devices will not be able to catch up with the resolution of capture devices. In other words, more bits are being captured than being displayed. This observation has driven us to rethink the possible user interactions with video. In particular, we believe that zooming into and panning around a video become important operations.

As opposed to currently most adopted trick play operations – fast-forward, rewind, fast/slow-motion playback – which operate in the temporal dimension, zooming and panning operate in the spatial dimension. Zooming allows users to view a cropped region of interest (RoI) at a high resolution, in effect, magnifying the RoI, while panning allows user to change the coordinates of the magnified RoI while keeping the size of the RoI fixed.

The zoom/pan operation is useful in many video applications, including sports, surveillance, and education. Consider an example in educational video. When watching a video lecture on a hand-held device with a small display, one can see the lecturer and the whiteboard but may not be able to read what is written on the board. One could zoom into the region around the written matter for a clearer view (Figure 1) and pan to view another area on the board as the lecture proceeds. Another example is viewing of surveillance video. One might want to zoom into an area in a scene to examine the detail more clearly (e.g., faces of suspects, license plate numbers), or pan to track a suspicious person around.

Supporting zooming and panning during local playback is relatively easy to achieve. The decoder only needs to decode bits necessary to display the RoI and scale down each frame to fit the display size. Our research is motivated by a more challenging problem of supporting zooming and panning in streaming video, where the high definition video is stored on a server and is streamed to one or more client(s) for viewing. To reduce the bandwidth consumed, the high resolution video is usually not sent directly to the clients, instead, the video is scaled down to a size appropriate for display at the client before streaming. The lack of original high resolution video at the client leads to a very different solution for zooming/panning than local playback. The client would need to send the request for a RoI to the server. The server would then send the RoI, scale down to the appropriate resolution, for display at client.

A naive solution would be for the server to transcode the requested RoI as a new video and send it as a stream as usual. Such a solution, however, incurs computation cost, and is not scalable to larger number of users, where different users can freely zoom and pan to different RoIs. Another naive and more restrictive solution is to restrict the possi-

MMSys'10, February 22–23, 2010, Phoenix, Arizona, USA.

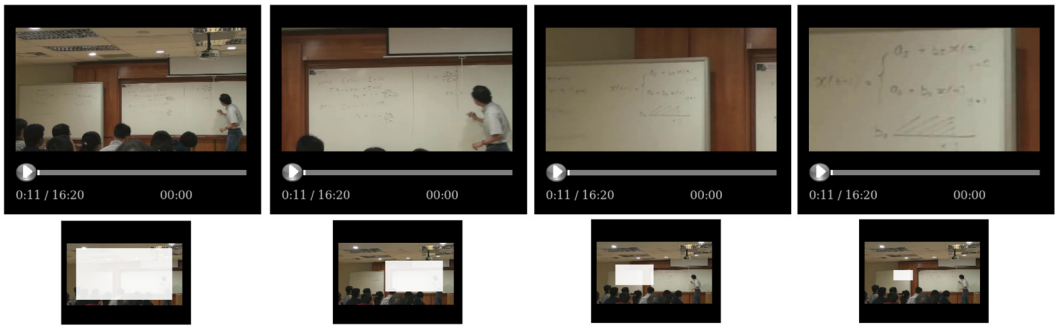


Figure 1: Example of Zooming. Users can zoom to view different level of details within the video. The images on top shows the video player, while the image below shows a thumbnail of the video with the zoomed-in RoI highlighted in white.

bilities of RoIs that the user can request and store a copy of video for each possible RoI on the server. This solution incurs a one-time computational cost at the expense of more storage and restrictive user interaction.

Our research aims to investigate into new approaches to encode, store, and stream a pre-recorded video that allow users to *freely* zoom and pan into any RoI in a video stream, and is *scalable* to large number of users by being *efficient in terms of bandwidth*, *computation cost*, and *storage*. To support zoom and pan, two operations are needed: scaling and cropping. As a first step towards this goal, we investigate into how to support dynamic cropping of RoI on video streams. Solutions to scaling are well known and can be supported by storing different versions of video at different zoom levels (i.e., resolution)[1].

This paper explores two methods to encode, store, and stream video that support dynamic selection of RoI for video cropping. The first method, *monolithic streaming*, uses videos encoded with off-the-shelf video encoder. The server precomputes a dependency graph of macroblocks. When users request for an RoI, macroblocks falling within the RoI are streamed along with all dependent macroblocks that are outside the RoI. The second method, *tiled streaming*, divides each frame into a grid of *tiles,* and encodes each grid as a video. Equivalently, one can modify the video encoder to restrict the motion vector and video slices to fall within a tile. When users request for a RoI, tiles that overlap with the RoI are sent. We based our work on MPEG-4 simple profile, but the techniques and principles apply for any DCT-based video codec.

Experimental evaluation shows that motion vector length and slice length determine the effectiveness of the two methods. For the same RoI configuration, monolithic streaming achieves lower data rates than tiled streaming when slice size is small and longer motion vectors are used. However, the performance of monolithic streams can deteriorate significantly for larger slice size. We also found that parameters that lead to better compression (longer motion vector, bigger tile sizes) do not necessarily translate to bandwidth savings when only an RoI is streamed.

The rest of the paper is organized as follows: Section 2 highlights some of the work required by, or related to, RoI-based streaming. Section 3 reports on a pilot study conducted to verify the need and usefulness of supporting zoom and pan, with arbitrary RoI cropping in a video stream. Section 4 describes the tiled streaming method, and is followed by a description of the monolithic streaming method in Section 5. Section 6 discusses an experimental evaluation of the two methods, followed by a concluding discussion in Section 7.

[1]The number of zoom levels is usually much less than the number of possible RoIs

2. RELATED WORK

Selecting a region of interest (RoI) to zoom into is a common human activity while dealing with images and maps. In the context of video, several projects/tools have proposed using zooming and scrolling to focus on a RoI within a high-resolution video. Techniques for automatically determining a RoI and adapting standard-based video encoding techniques for RoI based applications have been explored.

The Diver project [12] and virtual camera control [14] aim at providing a digital video re-purposing system that allows users to interact with pre-recorded video. One of the features is the ability to select a region of interest so as to spatially crop the stored video. Video re-targeting [9], addressed the problem of adapting video frames to a target display by way of spatial cropping and scaling. A candidate set of cropping windows is analyzed and an optimal cropping window that minimizes a distortion function is chosen to crop the video before it is scaled. The cropping window can change with motion in the video and is therefore adaptive.

In the context of RoI detection and tracking for stored video playback, there have been attempts to model the RoI [5] based on amount of motion. Such models could help determine the RoI, without human interaction with a display device. Multi-scale cropping [4] dealt with automated tracking of multiple RoIs defined by motion change. The aim was to minimize the number of RoI trajectories within the video while covering all the RoIs. Automatic detection and tracking of RoI using object detection in wide angled panoramic video [15] is another technique to realize automated virtual camera control. A single object (RoI) is detected and tracked across multiple synchronized cameras covering a class room lecture. Mavlankar et al. [11], address the issue of tracking an object within a RoI and its application in predicting a new RoI during explicit user interaction.

The previous work described above has established useful applications of supporting zooming and panning using the notion of a region-of-interest. Their research focuses on user interaction or automating the selection of RoI based on content analysis.

Current video standards do not support arbitrary, interactive cropping, and scaling required for RoI-based streaming. Recent H.264/AVC scalable extension [8] supports spatially scalable coding with arbitrary cropping in its Scalable High Profile [3]. The coded video, however, supports pre-determined spatial resolutions and cropping only, as it is designed for applications such as "pan and scan" when converting wide-screen DVD for 4:3 display and does not support interactions.

A recent work on fine-grained multi-resolution video [7] explored how to support a wide range of spatial resolutions. A follow-up work [6] investigated into supporting RoI cropping with constrained compression of MPEG-2 video. The latter method is the same as tiled streams explored in this paper.

Techniques for cropping by limiting temporal dependency [13] within a compressed video was addressed by Rehan and Agathoklis. They proposed a technique where by the first frame in the cropped set is encoded into an I-frame, with all predicted frames re-encoded with the new I-frame as the reference. As re-encoding works only when the RoI is known apriori, this method lacks flexibility. Exploiting flexible macroblock ordering (FMO) for region of interest cropping [2] is another approach suited for H.264 SVC. The RoI is encoded as a slice group and dependent/overlapping parts are encoded as another slice group. Although FMO disables inter-slice group decoding dependency within a picture, the existing temporal dependency still persists. The proposed approach is static in the number of RoIs which is fixed at the time of encoding the video. Such an approach has an inherent drawback of not allowing users to choose the RoI or change the RoI during playback. A similar approach, which takes advantage of FMO [1], solves the issue of bandwidth awareness. Network feedback is used to change the encoding parameters for a RoI, automatically determined based on motion characteristics. The RoI is encoded as a slice group using a lower quantization scale when compared to regions outside the RoI. Mavlankar et al [10], analyze the optimal slice size for RoI-based streaming with virtual PAN/ZOOM functionality. They make use of the fact that H.264 AVC allows creation of slices of arbitrary configurations due to FMO. They theoretically estimate and empirically validate the optimal slice sizes for HD videos of different resolutions.

Few attempts have been made to address interactive RoI based streaming of encoded video. We address the issue of packetization of slices, influence of motion vector, and methods to encode/pre-process compressed MPEG4 video so as to achieve streaming for user-specified RoI. In the following sections, we describe the *tiled stream* method and the *monolithic stream* method. The former is a new way of encoding a raw video while the latter uses a pre-processing stage before streaming an already encoded video.

3. PRELIMINARY USER STUDY

To study the usefulness of zooming and panning a video, we conducted a preliminary user study. We recorded High Definition (1920x1080) videos of two class room lectures attended by 81 students. The camera was mounted statically with a fixed view at the center of the class so as to have a full view of the white-board. Movement of the instructor was not explicitly tracked.

One week before the mid-term test of the class, the recorded videos were made available on a website. The web page only displays a video at a size of 320x180. At the default view, the user sees a scaled down version of the whole video (without cropping). We provide a user interface that allows users to perform zoom and pan operations on the videos through both keyboard and mouse. The interface supports six different levels of zooming (from 0 to 5). Level 0 is the default view and the level 5 is the most detailed one (highest zoom) equivalent to an RoI of 320x180 in the original HD video. Over a period of one week, students' interaction with these videos was logged. Figure 1 shows the video used in this user study.

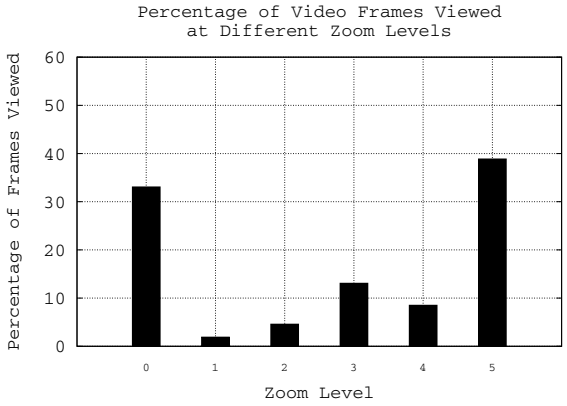


Figure 2: User Interaction with Zooming

Figure 2 depicts our observations on the most popular video viewed by 58 users. The figure shows the percentage of video frames viewed by the 58 users at different zoom levels. We note that the zoom operation was used for about 67% of the time and the maximum level of zooming (level 5), i.e, the smaller RoI size with maximum detail, seems to be preferred. This is understandable, to see the white-board, one needs to zoom. But it is not necessary to stay at this maximum detail level once the written matter has been viewed. Some of the students tend to listen to the audio with occasional zooming, perhaps because they have taken notes during the lecture and only refer to the board when necessary. Such traits could be the cause for zoom level staying at level 0 for 33% of the time. It is possible for zooming to take place only when something on the board called for attention.

Figure 3 shows the areas (RoI) viewed by different users. Each pixel location was tracked and assigned a score equal to the number of frames in which it appeared within a particular RoI. The set of scores was normalized and mapped to the range of 0 to 255 to create a grey-scale range. Pixel locations were assigned this normalized score and plotted as an image. The darker areas represent relatively least viewed regions and the lighter ones represent the most viewed regions in the video. It can be observed that the RoI is wide spread, indicating that it is necessary to have a system that can adapt to dynamic changes in the position of the RoI. Nevertheless, we also observe that certain areas are more

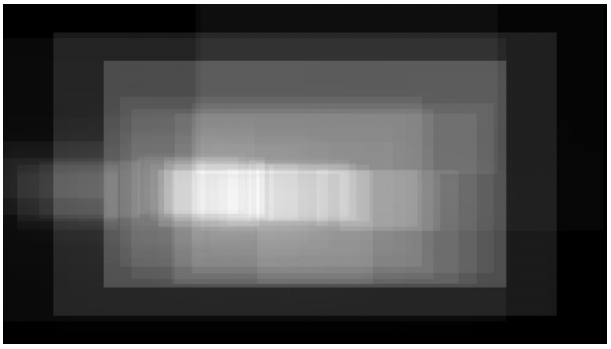


Figure 3: Heat Map of RoI Requested by Users

preferred regions of interest (bright areas along the center). Manual inspection showed that these regions cover a long white-board containing written matter and also regions where the instructor is seen moving about. Based on these observations, one can conclude that the RoI is dependent on video content and also on user preferences.

4. TILED STREAMS

The first method we explore in this paper is inspired by how Web-based map services (e.g., Google Map) support zooming and panning within a map. A commonly used technique for displaying on-line maps is to divide a large map into grids of smaller images. Images that overlap with the RoI of the user are sent to the browser for display. We employ a similar technique for zooming and panning within a high resolution video.

We call this first method *tiled streaming*. Video frames are broken into a grid of tiles in the pixel domain (Figure 4). For convenience, we use tiles that are aligned with macroblock boundary. One can view the video as a three dimensional matrix of tiles. Tiles in the same x-y position in the matrix are temporally grouped and encoded independently using a standard encoder to create a *tiled stream*. These streams are indexed by the spatial region they cover. For a given RoI, a minimal set of tiled streams covering the RoI is streamed by looking up the index. New tiles may be included into the stream or tiles may be dropped when the RoI changes. This technique is similar to that proposed by Feng et al. [6].

As streaming tiles is not a conventional approach to video streaming, a modified video player is needed to playback tiled streams. The server sends a tile header (similar to file header) for each tile so that the corresponding tile could be decoded when streamed. The video player needs to buffer the tiled streams and synchronize between them during playback.

The complication of buffering and synchronizing between multiple streams may be avoided by encoding the tiles into a single video stream, as proposed by Feng et al. [6]. By modifying the video encoder such that motion vectors are confined to a tile area and entropy coded bit strings are also localized to a tile, we can ensure that each tile is still independently decodable while avoiding the complications of synchronization.

The main advantage of tiled streams is its simplicity at the server. A server, upon receiving the RoI request from a client, can easily determine the set of tiles that overlaps with the requested RoI before transmission. This solution is also admissible to a publish-subscribe paradigm. The server can multicast each tile to one channel, while a client that wishes to view a particular RoI only needs to subscribe to the channel serving tiles that overlap with its RoI.

Tiled streaming, however, has several potential drawbacks. First, either a customized video player is needed to combine the tiled streams, or a customized video encoder is needed to generate a single stream where each tile is independently decodable. Second, since there is a constraint on motion vector length, the compression efficiency is reduced, leading to higher storage requirement on the server and higher bandwidth for streaming the same RoI. Finally, there will likely be wastage of bandwidth as not all bits transmitted are necessary for decoding and display of the RoI. Using tiled streaming, the whole tile needs to be streamed even when there is a small overlap with the RoI. Since RoI does not necessarily align with the tiles, redundant regions outside of RoI will be transmitted as well. Given a fixed tile area (A_T), tile width (w_T), tile height (h_T), RoI area (A_R), RoI width (w_R) and RoI height (h_R), it can be shown that in the best case the region selected is $\left\lceil \frac{w_R}{w_T} \right\rceil \left\lceil \frac{h_R}{h_T} \right\rceil \frac{A_T}{A_R}$ times the actual RoI. If the RoI width and height are integral multiples of the tile width and height, then tiled streaming would result in transmission of a region equal to the dimension of the RoI. In the worst case, the region selected would be $\left(\left\lceil \frac{w_R}{w_T} \right\rceil + 1\right)\left(\left\lceil \frac{h_R}{h_T} \right\rceil + 1\right) \frac{A_T}{A_R}$ times the size of the actual RoI.

There is a trade-off between wastage and storage as tile size changes. In the extreme case, each macroblock is a tile, leading to lowest compression efficiency but fewest redundant regions. As tile size increases, compression efficiency increases, but more redundant regions will be streamed. We evaluate the effect of using different tile sizes in Section 6 of this paper.

5. MONOLITHIC STREAM

Supporting RoI decoding using tiled streams can lead to transmission of redundant bits to clients – bits that do not contribute to decoding of pixels within RoI at all. To overcome this issue, we explore a second method in this paper, called *monolithic stream*, that transmits only bits that are required for decoding of RoI. This method uses a video stream encoded with a standard encoder. The server, however, needs to first analyze the dependencies among macroblocks (Figure 5) within the video and construct a data structure that allows the following query: given the current RoI and a macroblock m, is m needed by the client to decode a macroblock that falls within the RoI? Clearly, if m falls within the RoI the query returns yes. Otherwise, the query will return yes if and only if there exists a macroblock m' that falls within the RoI and depends either directly or indirectly on m. In other words, the bits in m are needed to parse and decode m' properly. This analysis of video is done off-line and is a one-time operation. During transmission, the server parses through each macroblock m and puts m into a network packet for transmission if the above query is true. As a result, the server only transmits necessary bits to the clients.

The client would need a robust, but otherwise standard, video decoder and player to display the RoI. Since not all bits from the original video are sent, regions outside of the RoI are either not fully decoded, or decoded but with in-

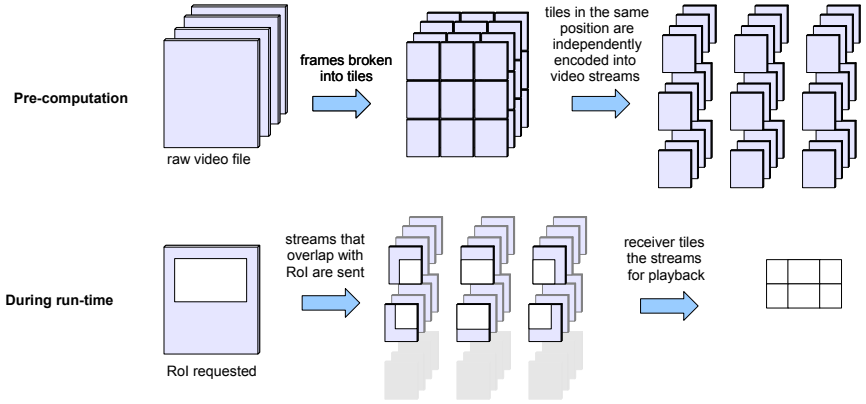


Figure 4: Tiled Streams

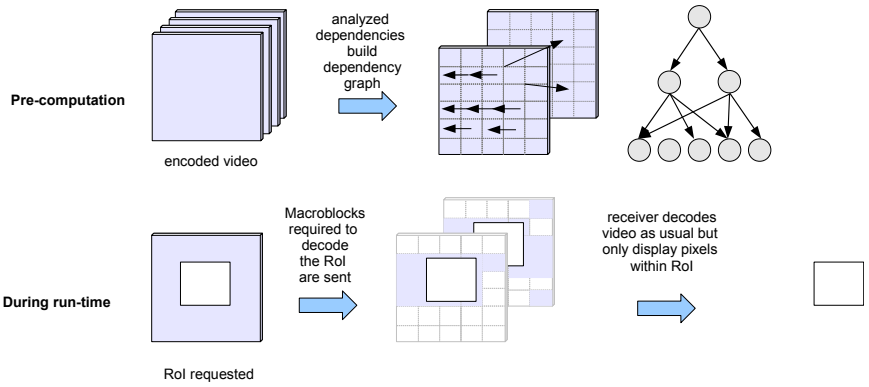


Figure 5: Monolithic Stream

correct pixel values. As we only display the RoI, the error values do not affect the video playback.

Note that bits outside of the RoI are likely to be transmitted to the client, but unlike tiled streams, these bits *do* contribute to the decoding of the RoI due to dependencies. Nevertheless, the bits outside of RoI still constitute a bandwidth overhead and it is desirable to reduce these bits as much as possible by reducing the dependencies among the macroblocks. The bandwidth overhead is affected by various encoding parameters. Encoding the video with more I-frames, restricting the length of motion vector, and reducing the size of the slices should help reduce the amount of dependency. On the other hand, reducing the amount of dependencies decreases compression efficiency leading to increase in storage size and more bits per macroblock.

5.1 Dependency

We now elaborate on how we determine if a macroblock m' depends on another macroblock m. The dependency could be due to the following reasons in MPEG-4. First, m' is an inter-coded macroblock that refers to some pixels that fall in m. We call this the *motion vector dependency*. Figure 6 illustrates the motion vector dependency for three frames. A macroblock in Frame 3 needs to be decoded and has a motion vector (arrow shown in the figure) that points to a particular position in Frame 2, its reference frame. As the referenced position is not on a macroblock boundary, four macroblocks, labeled 5, 6, 9, and 10 are required in order to decode the macroblock in Frame 3. Each of these macroblocks might refer to a set of macroblocks in Frame 1. The dependency propagates through differentially coded frames and the number of dependent macroblocks increases exponentially in the worst case. We will show how this dependency can be optimized in a later section. Motion vector dependency can be found by tracing the motion vector recursively while making a pass through the video. If the client needs to decode a macroblock m' that depends on m due to motion vector dependency, then the server needs to send m, along with all macroblocks that m depends on. One can view the dependency relationship among the macroblocks as a directed acyclic graph. An example of the graph is shown on the right of Figure 6.

Second, since macroblock boundaries are not byte-aligned, macroblock m that comes before the bit-stream needs to be parsed before the video decoder can get to m' due to variable length coding (VLC). We call this the *VLC dependency*. Note that macroblocks that are sent due to VLC dependency only, need not be fully decoded. They are sent to allow parsing and maintaining the syntax of the bit-stream. If the client needs to decode a macroblock m' that depends

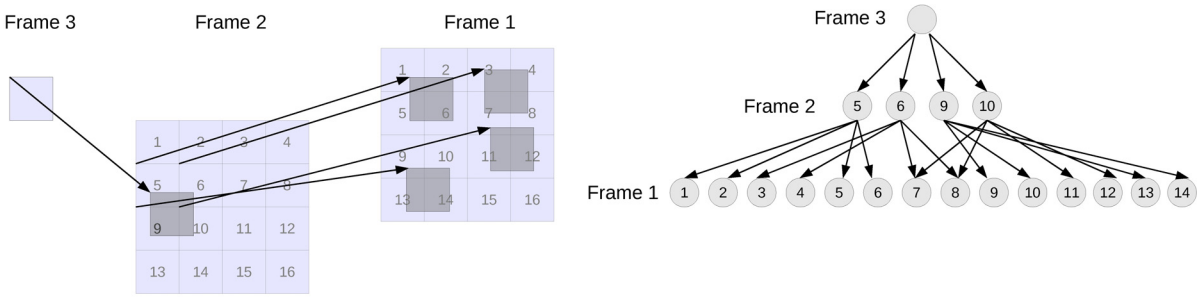


Figure 6: Motion Vector-based Macroblock Dependency

on m due to VLC dependency, then the server only needs to send m. Macroblocks that m depends on (through motion vector dependency) need not be sent.

Other dependency exists. For instance, due to prediction of DCT coefficient – the AC/DC values of m' could be predicted from m. Other video coding formats could introduce additional dependencies, but the principle of a monolithic RoI stream remains unchanged.

5.2 Dependency List

To allow the server to quickly determine if a macroblock needs to be sent for a given RoI, we maintain the following data structure called *dependency list*. For each macroblock m, the server pre-computes a set $D(m)$ containing the positions of all macroblocks that depend either directly or indirectly on m. In other words, $D(m) = \{(x, y)|$there exists a macroblock m' at position (x, y) that depends on $m\}$. $D(m)$ forms an irregular shape region that m contributes to in some frame. For each macroblock m, the server checks if $D(m)$ overlaps with the RoI. If so, then the server transmits m. The set $D(m)$ is maintained as a sorted list of positions. To determine if $D(m)$ overlaps with the RoI, the positions of the list for m are scanned to determine if any of them fall into the RoI. If there is at least one match, m is transmitted.

Figure 7 shows how a dependency list would look like for a specific example. Macroblock at position 2 in Frame 3 depends on a region spanning across macroblocks at positions 5 and 6 in Frame 2. Further, macroblocks at 5 and 6 in Frame 2 depend on a region overlapping macroblock positions 5, 6 and 7 in Frame 1. A partial dependency list for Frame 1 is shown at the extreme right in the figure. Each node in a square box corresponds to a macroblock in Frame 1. Each macroblock points to a list of circular nodes representing macroblock positions in other frames that depend on the macroblock. Hence if the RoI covers position 6, macroblocks at positions 7, 9, 10 and 11 in Frame 1 need to be transmitted (in addition to the macroblock at position 6).

Computational cost of searching through a list can be reduced by maintaining the dependency list as a 2D range search tree. Use of efficient data structures is out of the scope of this paper and hence not discussed further. Nevertheless, we observed that our current implementation, on an average, takes less than 2msec to look-up the dependency list for all macroblocks in each frame on a 2.8GHz processor machine.

5.3 Reducing Motion Vector Dependency

The amount of motion vector dependency can be reduced if additional analysis is done on the macroblocks and its motion vector when constructing the dependency list. Figure 8 shows an example of how this can be done. The figure shows the same set of macroblocks as in Figure 6. On Frame 1 in Figure 8, however, we highlight the exact pixels needed to decode the macroblock in Frame 3 properly. Only the macroblocks that contain these pixels (shown in darker color) need be decoded. Not decoding the rest of the pixels will result in incorrect pixel values in certain regions of Frame 2, but these errors would not affect the decoding of the macroblock in Frame 3 since the erroneous values are never referred to. The graph on the right of Figure 8 shows the dependency structure after removing unnecessary dependencies.

Note that, during run-time, when the RoI is given, we can apply the same principle and reduce the amount of motion vector dependency. If the RoI is not aligned with macroblocks, there will be pixels at the bordering macroblocks that will not be displayed, and hence errors are not visible.

5.4 Reducing VLC Dependency

Most video encoders break the VLC dependency among macroblocks by introducing resynchronization markers. These markers signal the beginning of a slice, which is the smallest independently decodable unit in a video frame (assuming the reference frame is decoded). Slice has an adverse effect on the monolithic stream method. Its impact can be better understood by referring to Figure 9. Different shaded areas are slices resulting from entropy coded bit strings. The square segments are macroblocks. Although slice size (in terms of bytes) is fixed, each slice may contain different number of macroblocks. Let us say an entropy coded slice consists of five macroblocks. Further let it be that macroblocks 3-5 are within the RoI. After the macroblocks in the slice are entropy coded and assigned a variable length coded string, let bits 100-200 correspond to macroblocks 3-5. Then bits 1-99 have to be transmitted so that the variable length code for bits 100-200 (macroblocks 3-5) can be decoded. This means macroblocks 1-2 will be transmitted although they are outside the RoI. Hence slice results in an inevitable transmission overhead. This overhead can be reduced by reducing the slice size but would result in lower compression via VLC.

It is useful to think of each slice as consisting of three segments. Suppose a macroblock m is the first macroblock within a slice that needs to be decoded, and m' is the last

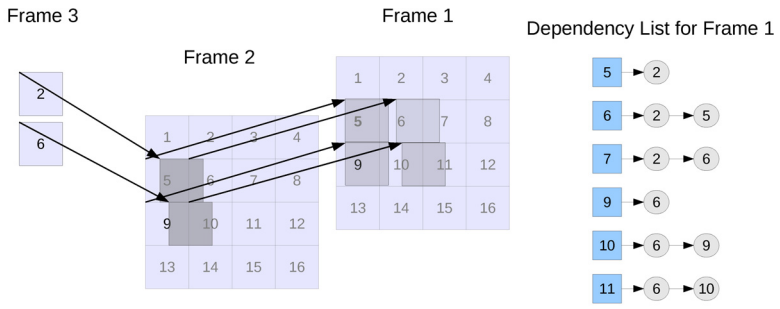


Figure 7: Maintaining Dependency List for a Frame

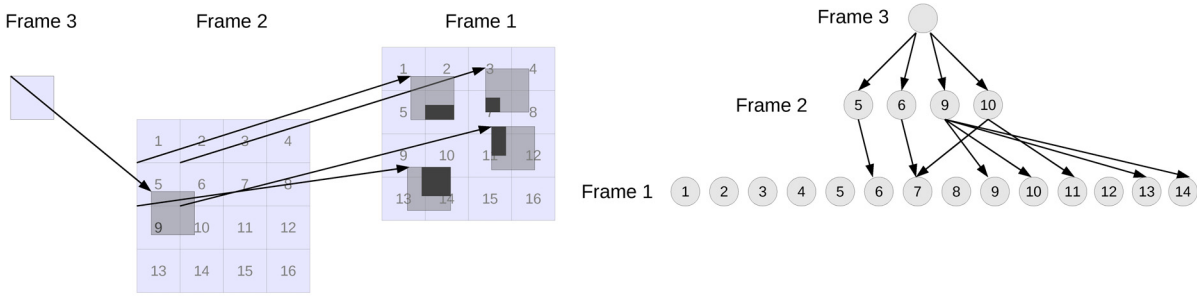


Figure 8: Run-time Optimization on Dependency

macroblock within the same slice that needs to be decoded. The first segment consists of bits from the beginning of this slice until m. This segment needs to sent, since they are needed to maintain the syntax of the stream and to "get to" m, but they need not be decoded. The second segment consists of bits between m and m'. The third segment consists of bits after m'. Bits in the last segment are neither needed for parsing nor for decoding. Thus, we can truncate the slice by not sending these bits. Robust decoders have the ability to synchronize to the next slice in case the slice header is not updated after truncation. However, updating the header fields of a slice is needed for bit-stream compliance.

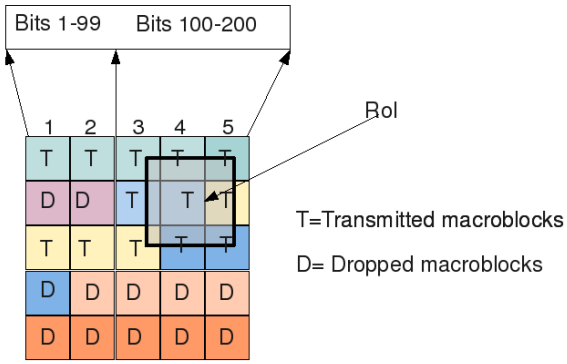


Figure 9: Slice Structure Impacts Transmission Overhead

6. EXPERIMENTAL EVALUATION

In this section, we outline various studies conducted to compare the performance of monolithic streaming and tiled streaming. The aim of the study is to understand the influence of slice size, motion vector length, and amount of motion in the video on the performance of the two methods based on the bandwidth required to send a RoI. We also evaluated the storage efficiency of the resulting video compressed with these methods.

6.1 Experimental Data and Conditions

(a) Rush-Hour (b) Tractor

Figure 10: Screen-shots of Test Video Sequences.

We evaluate the two methods above using two standard HD (1920 x 1080p) video files[2]. The first, called *Tractor,* consists of 688 frames and has dense motion vectors, while the second, called *Rush-Hour*, consists of 498 frames and has comparatively less motion. Figure 10 shows a screen shot of

[2]Available at
ftp://ftp.ldv.e-technik.tu-muenchen.de/dist/test_sequences/1080p

both videos. The two videos were encoded using full search for motion compensation so that the motion vector length can be as high as the specified maximum. We used FFMPEG[3] for all encoding operations with full search, quantization scale set to two, video codec being MPEG-4 simple scalable profile, and with a closed GoP structure of IBBPBBP. As a low quantization value was chosen, the encoded video rate is very high. The two videos were encoded for a combination of maximum motion vector length (in pels) chosen from $\{4, 8, 16, 24, 32, 40, 48, 64, 72\}$ and slice size (in bytes) chosen from $\{64, 128, 256, 512, 1460\}$. The specified motion vector length is the maximum possible motion vector that can be used while encoding. If the video has little motion, then motion vectors tend to be short.

As a result of different slice size and motion vector length combinations, there were 45 video samples for *Rush-Hour* and *Tractor*. Further, these 45 samples were encoded with a GoP length of 7 and 13. These 45 sample videos were then subjected to a RoI-based transmission for various RoI configurations. The width of the RoI (in macroblocks) was from the set $\{15, 30, 45, 60, 90, 120\}$ and the height (in macroblocks) from the set $\{15, 30, 45, 60\}$ macroblocks. As a result we had 24 RoI configurations, which can be categorized into three sets: RoIs where width equals the height (*square RoIs)*, RoIs where width is greater than the height (*wide RoIs*), and RoIs where the width is less than the height (*tall RoIs).* For each RoI configuration, we conducted our evaluation at five randomly chosen locations in the frame. The random locations were the same for both methods. When reporting the results, we only show six different RoI dimensions to avoid clutter. Similar trends were also observed for the remaining 18 RoI dimensions.

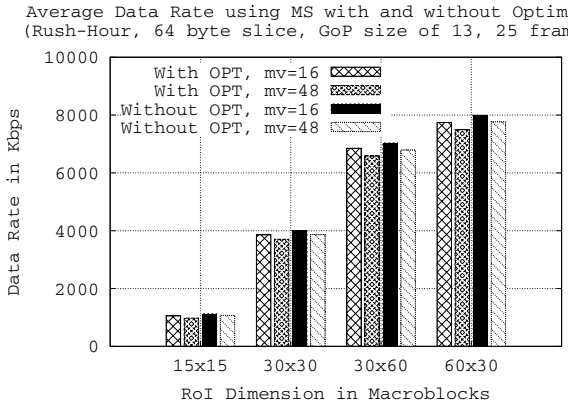
The main metric that we measure is the average data rate required when transmitting RoI of different dimensions. The average data rate is computed as the number of bits that would be transferred for a specific RoI dimension, averaged over five random locations, when the video is played out at 25 frames per second. It was observed that the PSNR of the luminance component of tiled stream and monolithic stream varied only in the second decimal place. Hence we do not compare the two streaming approaches based on PSNR readings.

The tile streaming method used the same FFMPEG encoder with the parameters described in the previous paragraphs. We evaluated three different tile sizes (4x4 macroblocks, 8x8 macroblocks, and 16x16 macroblocks). We always refer to tile stream using 4x4 macroblocks unless explicitly referred to the other tile sizes.

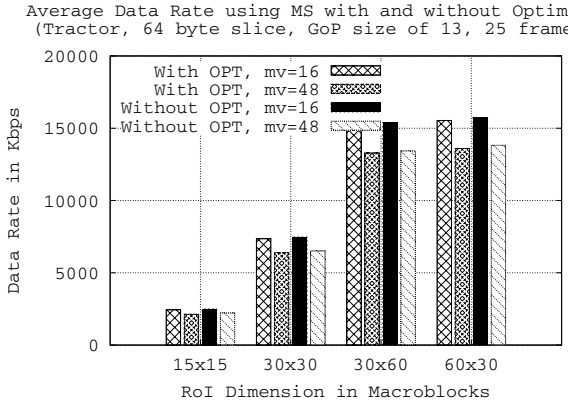
The monolithic streaming method has two variants, one which uses run-time optimization on the dependency and the other without this optimization step. We always refer to the optimized version unless explicitly stated otherwise.

6.2 Benefits of Motion Vector Dependency Optimization

We first evaluate the importance of run-time optimization of motion vector dependency (described in Section 5.3) when monolithic stream is used. Figure 11 shows the bandwidth required with and without the run-time dependency optimization for a slice of 64 bytes, using *Rush-Hour* and *Tractor* with a GoP size of 13. We can see that optimization helps in marginally reducing the data rate (of the order of a few hundred kilo bits per second). *Rush-Hour* benefits more from optimization than *Tractor,* although not significantly.



(a) Rush-Hour



(b) Tractor

Figure 11: Run-Time Dependency List Optimization

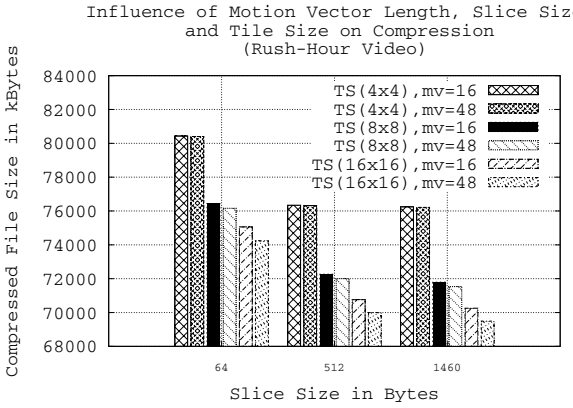
6.3 Influence of Tile Size

Our next experiment evaluates the effects of tile size on the effectiveness of tiled streaming. Figure 12 shows that compressed file size of tiled streams for tile size of 4x4, 8x8, and 16x16 macroblocks, using both *Rush-Hour* and *Tractor* as test video sequences. As the tile size increases, the compression ratio improves. This is expected since there are less constraints for motion compensation. As the tile size increases, tiled stream tends to display more of monolithic stream characteristics.

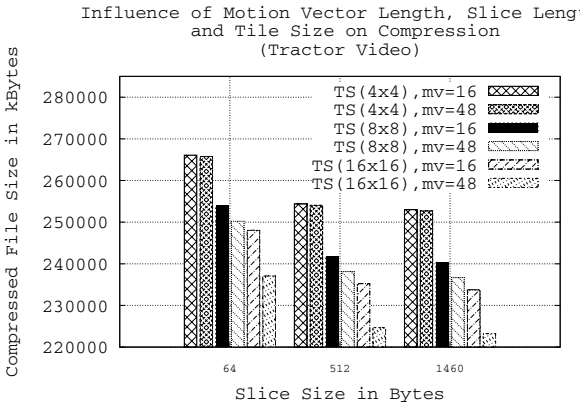
As tile size increases, we need fewer bits to code each macroblock. However, more macroblocks will be sent for the same RoI size. To see if this factor would offset the savings in improved compression, we plot Figure 13. The figure shows the average data rate required for different tile sizes and RoI sizes. We see that as tile size increases, more bits are sent. Thus, the unnecessary macroblocks sent in each tile have nullified the savings due to better compression.

For the rest of the experiments, we choose to use a tile size of 4x4, since this yielded the best average data rate.

[3] `www.ffmpeg.org`

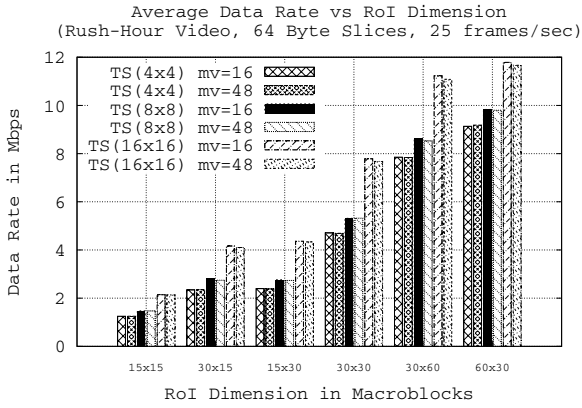


(a) Rush-Hour

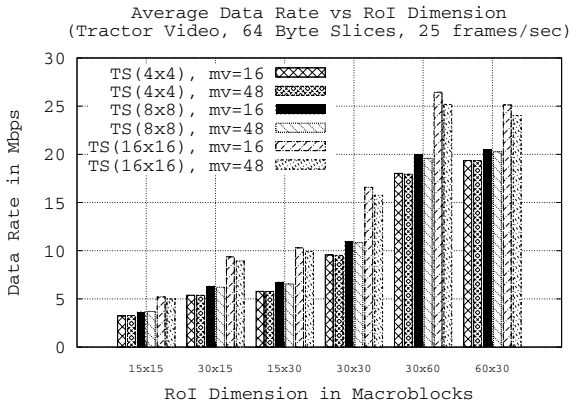


(b) Tractor

Figure 12: Compression Efficiency with Different Tile Sizes



(a) Rush-Hour



(b) Tractor

Figure 13: Data Rate with Different Tile Sizes

6.4 Influence of Motion Vector Size and Slice Size on Compressed File Size

We know that increasing motion vector length and slice size would improve compression of monolithic streams. To see which parameter is more influential, we plot Figure 14, showing how the compressed file size varies with motion vector length and slice size for *Rush-Hour* and *Tractor*. We can see that increasing motion vector length significantly reduces the file size, while larger slice sizes offer modest reduction in file size. Nevertheless, a video with 64 byte slice is significantly larger than a video with 128 byte slices.

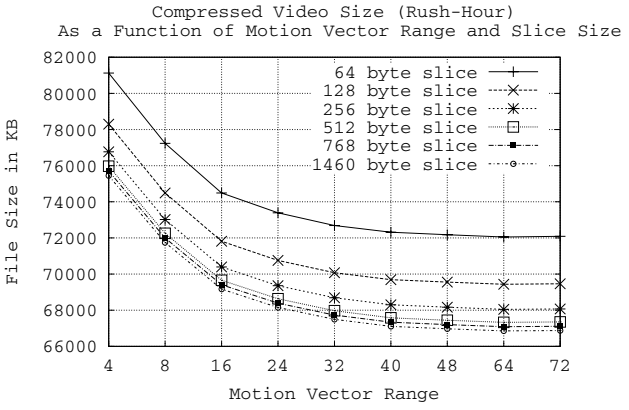
The *Tractor* video can be compressed better even for motion vectors as long as 48 (Figure 14b) where as for *Rush-Hour*, compression efficiency almost remains constant after a motion vector size of 40 due to lesser motion in the video.

Next, we compare the compression efficiency of using monolithic stream and tiled streams (calculated by summing up the sizes of individual encoded tiled streams). Figure 15 shows that the compressed file size for *Rush-Hour* and *Tractor*, using both methods with different motion vector length and slice sizes. We can see that, the reduction in file size when motion vector length increases is less significant for tiled streams. Since motion vectors are restricted to within a tile, allowing longer motion vectors does not help much and regions which could have been encoded by motion compensation are now limited. As a result, tiled stream has higher storage cost and needs to send more bits per macroblock.

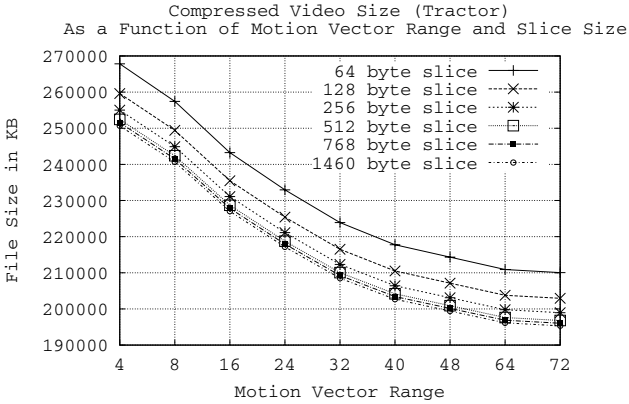
6.5 Monolithic versus Tiled Streams

Figure 16 shows the data rate achieved for *Rush-Hour* and *Tractor* (with slice size of 64 bytes). The graphs show that monolithic streams leads to lower data rate than tiled streams for both *Rush-Hour* and *Tractor*. For small RoIs (15x15), tiled streams require about 1 Mbps than monolithic stream. The differences increase as RoI size increases. As the size of the RoI increases, the perimeter of the RoI also increases. As a result, there are more number of tiles falling on the perimeter, increasing the number of redundant macroblocks sent. Since tiled streams uses more bits per macroblocks, more data is sent using tiled stream.

Figure 17 shows the effect on data rate when the slice size is increased to 1460 bytes. We see that tiled streams now achieves much lower rate than monolithic stream. Long slices result in significantly more VLC dependencies in the a monolithic stream. When a macroblock m is needed, all other macroblocks that come before m in the same slice should be sent in order for m to be decodable. Although using larger slices would reduce the compressed file size by a few hundred kilo-bytes (Figure 15), its impact on monolithic RoI streaming is adverse. Hence, it is better to encode

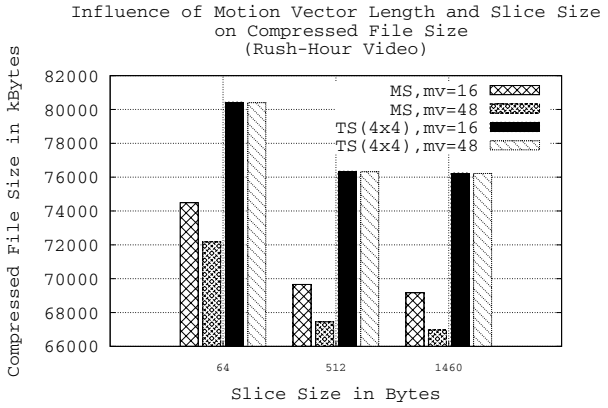


(a) Rush-Hour

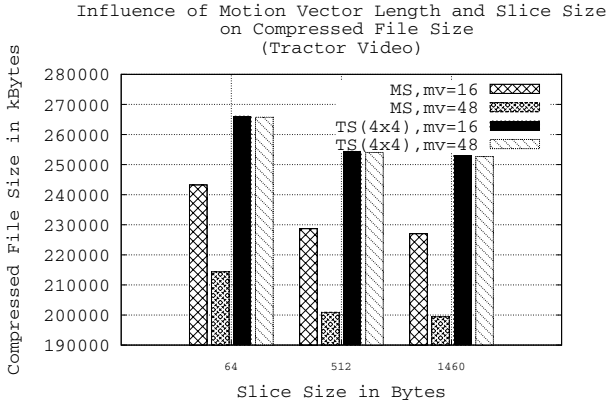


(b) Tractor

Figure 14: Compression Efficiency of Monolithic Stream



(a) Rush-Hour



(b) Tractor

Figure 15: Compression Efficiency of Tiled Streams and Monolithic Stream

a video with smaller slices, trading off a small increase in storage space for a much lower data rate.

Another observation we can make from the figures above is the effect of motion vector length. Allowing the encoder to use long motion vectors results in reduced data rate for any given RoI. Although use of longer motion vectors tends to increase the amount and spread of motion vector dependency, it also increases compression efficiency there by reducing data rate.

6.6 Summary

Our experiments show the trade-off of different encoding parameters and their effects on the bandwidth efficiency of monolithic streaming and tiled streaming. We summarize the most important observations below.

1. Larger slice size significantly increases the bandwidth overhead of monolithic streaming. This overhead is caused mainly by increase in VLC dependency rather than decrease in compression efficiency.

2. Video with high motion causes significant bandwidth overhead in tiled streams. This overhead is caused mainly by decrease in compression efficiency and is more significant in larger RoI.

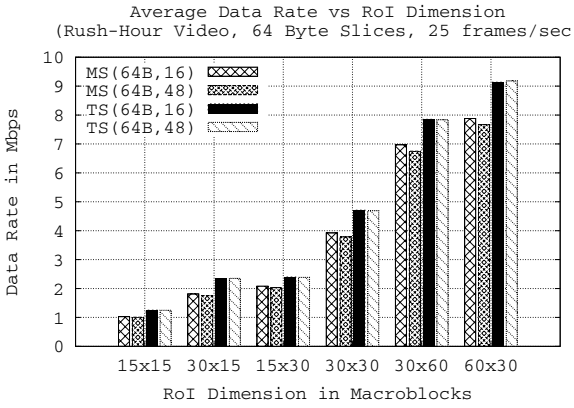
3. Allowing longer motion vector range significantly improves compression efficiency in monolithic stream, there by reducing the data rate despite increasing dependency in the video.

4. Using larger tiles significantly improves compression efficiency in tiled streams, but would still lead to higher bandwidth when streaming RoIs due to wasted transmission of bits that do not contribute to the decoding of RoI.

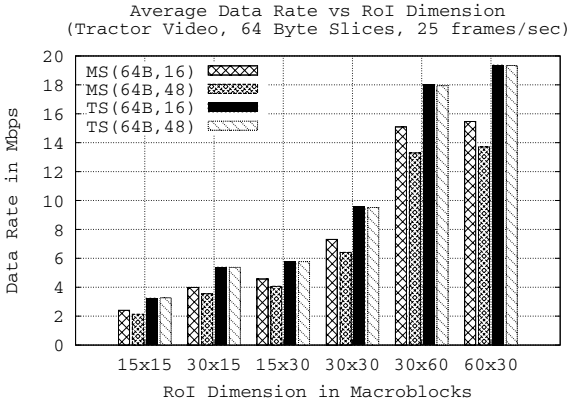
Our experiments point to monolithic stream as a more bandwidth efficient method to stream RoI compared to tiled streams in general, for different level of motion in video and for different RoI sizes, as long as careful selection of encoding parameters is exercised. As an additional advantage, monolithic stream has better compression efficiency, leading to smaller file sizes.

7. CONCLUSIONS AND FUTURE WORK

We presented two methods for RoI-based video transmission to support zooming and panning. The first, tiled streaming, breaks frames of a raw video stream into tiles and encodes individual tiles using a standard encoder. The requested RoI is met by sending tile streams that overlaps

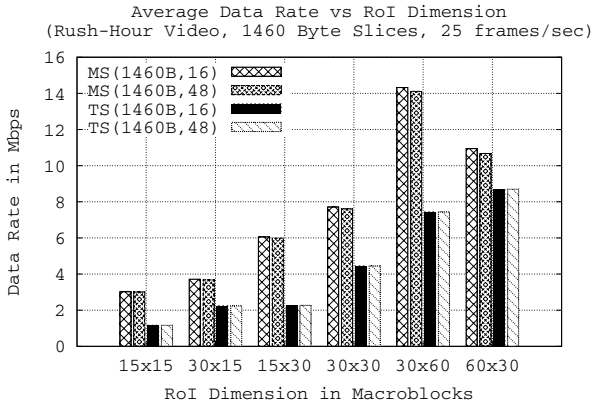


(a) Rush-Hour

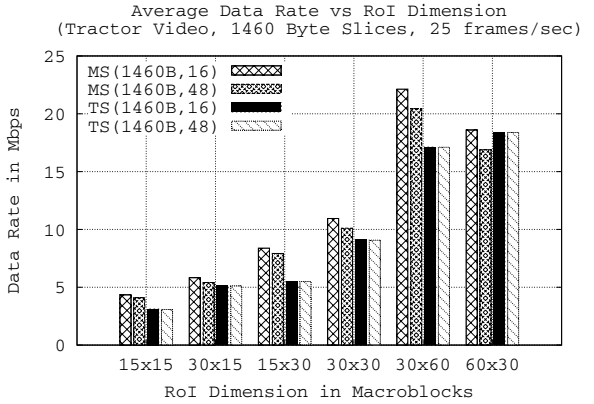


(b) Tractor

Figure 16: Bandwidth Efficiency (Slice Size: 64 bytes)



(a) Rush-Hour



(b) Tractor

Figure 17: Bandwidth Efficiency (Slice Size: 1460 bytes)

with the RoI. Monolithic stream uses a standard coded video for streaming. A pre-processing stage is used to process the video and build a macroblock dependency list. The requested RoI is met by sending bits that are needed to decode the macroblocks within the RoI, through consulting the dependency list. Our experimental results suggest that monolithic stream with proper choice of parameters achieves better bandwidth efficiency than tiled streams.

There are many possible directions this research can take. Our next step is to consider a combination of tiled stream and monolithic stream to stream a RoI. This hybrid streaming method would analyze each tile stream and build a dependency list. A tile stream that falls entirely within the RoI is sent as is. A tile stream that falls on the border of RoI would be sent as if it is a monolithic stream – only bits that contribute to the RoI decoding are sent. We expect that hybrid streaming would not gain much compared to tiled streaming when small tiles are used, but might be able to significantly reduce the redundant data sent for larger tiles. Compared to monolithic streaming, hybrid streaming restricts the propagation of motion vector dependency to within a tile and could lead to better bandwidth efficiency.

The monolithic stream method would benefit from the use of an efficient data structure for maintaining the dependency list. We plan to use a data structure that supports 2D range search to reduce the computational cost of look-up.

8. REFERENCES

[1] P. Baccichet, X. Zhu, and B. Girod. Network-Aware H.264/AVC roi coding for a multi-Camera wireless surveillance network. In *Proc. International Picture Coding Symposium*, 2006.

[2] T.M. Bae, T.C. Thang, D.Y. Kim, Y.M. Ro, J.W. Kang, and J.G. Kim. Multiple region-of-interest support in scalable video Coding. *ETRI Journal*, 28(2), 2006.

[3] C.A. Segall and G.J. Sullivan. 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.

[4] H. El-Alfy, D. Jacobs, and L. Davis. Multi-scale video cropping. In *MULTIMEDIA '07: Proceedings of the 15th international conference on Multimedia*, pages 97–106, New York, NY, USA, 2007. ACM.

[5] X. Fan, X. Xie, H. Zhou, and W. Ma. 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.

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

[7] J. Huang, W. Feng, and J. Walpole. An experimental analysis of DCT-based approaches for fine-grained multiresolution video. *Multimedia Systems*, 11(6):513–531, June 2006.

[8] ISO/IEC. ISO/IEC JTC 1/SC 29/WG 11, scalable video coding applications and requirements, 2005.

[9] F. Liu and M. Gleicher. 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.

[10] A. Mavlankar, P. Baccichet, D. Varodayan, and B. Girod. 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.

[11] A. Mavlankar, D. Varodayan, and B. Girod. 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.

[12] R. Pea, M. Mills, J. Rosen, K. Dauber, W. Effelsberg, and E. Hoffert. The diver project: interactive digital video repurposing. *IEEE Multimedia*, 11(1):54–61, Jan-March, 2004.

[13] M. Rehan and P. Agathoklis. Frame-Accurate video cropping in compressed MPEG domain. In *Proc: PacRim 2007*, pages 573–576, 2007.

[14] R.S. Aygun and A. Zhang. Integrating virtual camera controls into digital video. In *Proc. IEEE International Conference on Multimedia and Expo, ICME'04*, volume 3, pages 1503–1506, June, 2004.

[15] X. Sun, J. Foote, D. Kimber, and B.S. Manjunath. Region of interest extraction and virtual camera control based on panoramic video capturing. *IEEE Transactions on Multimedia*, 7(5):981–990, Oct, 2005.