SUPPORTING ARBITRARY ZOOM IN ZOOMABLE VIDEO

CONG PANG

NATIONAL UNIVERSITY OF SINGAPORE

2013

SUPPORTING ARBITRARY ZOOM IN ZOOMABLE VIDEO

CONG PANG

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF SCIENCE

SCHOOL OF COMPUTING
NATIONAL UNIVERSITY OF SINGAPORE

2013

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Cong Pang

February 12, 2014

Acknowledgements

First of all, I would like to thank my advisor Associate Professor Wei Tsang Ooi for providing me the opportunity to work on this project and guiding me through.

Prof. Ooi introduced me to the world of video coding and initiated the idea of this project. He arranged meetings and discussion sessions regularly at which I learned a lot about various video technologies and applications. He is always approachable and helpful throughout this project. His expertise and guidance helped me to drive the project forward, making me work efficiently especially on experimental simulation.

I also would like to thank Dr. Ravindra Guntur, Mr. Ngo Quang Minh Khiem, Mr. Arash Shafiei, Mr. Zhenwei Zhao and Dr. Vu Thanh Nguyen, who shared their experience, materials and expertise with me. The first few discussions with Dr. Ravindra Guntur and Mr. Arash Shafiei are particularly useful to get me started on this project.

Finally, I would like to thank the Jiku Project group for the valuable discussions with them on weekly meetings.

Contents

Summary

Zooming into a live video stream on small screen devices, such as mobile phones, provides a personalized experience in which users can watch interesting regions within the video at higher resolution. A common method to implement zoom operation on live video streams is bitstream switching, where the captured video stream is re-encoded into multiple streams with different resolutions. Zooming into a specific region therefore is equivalent to cropping the region from a higher resolution version of the video.

This thesis considers the following problem: which resolution levels should we re-encode the captured video in, given a set of zoom levels that the users are requesting for. The set of resolution levels depends on the amount of processing power available (to re-encode), the amount of bandwidth available, and the quality of the video region displayed to the user.

We proposed two strategies, one optimizes for quality and one trades off between quality and bandwidth. Both use the processing power as the constraints. We compare our strategies to a naive scheme that statically determine the resolution levels without considering the user requests, and showed that our strategies leads to lower bandwidth and better video quality.

List of Tables

List of Figures

Chapter 1

Introduction

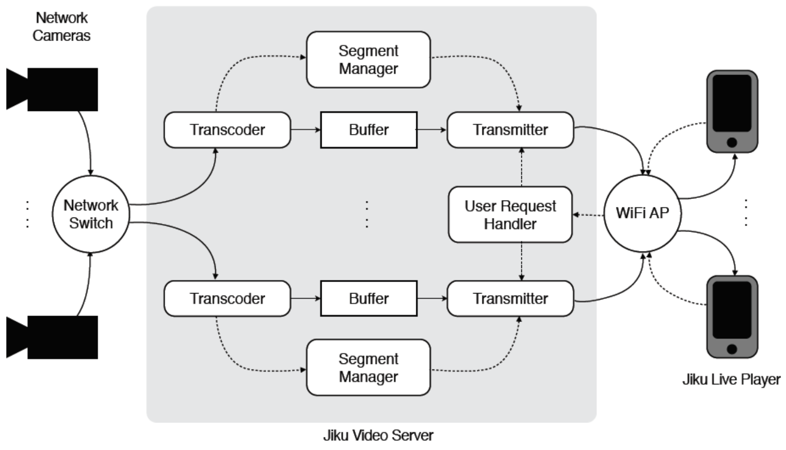


Figure 1.1: Jiku Architecture

To improve the experience of attending urban events, we have developed a system called *Jiku Live* [24] that allows attendees to use their mobile phones to access video streams of the events captured using networked cameras provided by the organizers. Users can browse, watch, interact, record, and share video streams of the events, providing a personalized experience in which attendees can watch the part of the events that are relevant and interesting to them, possibly from a different angle of their physical locations.

One of the key features in Jiku Live is *zoomable video streaming* [13, 10, 8]. Jiku Live streams live videos to mobile phones, allowing multiple users to zoom and pan to see different regions of interest within the video streams at the same time. The system captures the scene at a video resolution that is much higher than the display resolutions on the mobile phones. When the user zooms in, a higher resolution version of the video is cropped and transmitted for playback on the mobile phones.

To support zoomable video, the system does two things: (i) video frames are split into a grid of either non-overlapping [10] or overlapping [8] tiles, (ii) video frames are re-encoded into short segments (say, one second each) at different resolutions. When a user zooms in, tiles at the resolution closest to the requested zoom level that overlap with the required region-of-interest, or RoI, are sent to the client for decoding.

The Jiku Live system comprises of *Jiku Video Server* and *Jiku Live Player*. The overview of Jiku Video Server is presented in Figure 1.1. The Jiku Video Server is responsible for converting video streams from network cameras into tiles before streaming the RoI to the clients; while Jiku Live Player is a typical streaming video client with added support for zoom and pan operations. Jiku Live Player communicates with Jiku Video Server and receives video streams from it. The client also allows users to switch between the available network cameras.

Streaming at an *arbitrary RoI* and an *arbitrary zoom level* is essential to supporting smooth zooming and panning within the video streams. This thesis focuses on the second issue (we call it *arbitrary scaling*).

First, consider a Jiku Live system that does not support arbitrary zoom level. Users can only zoom in and out at levels that correspond to the resolutions of the tiles created. Zooming is "discrete." In such systems, it is typical that the zoom levels (and thus the resolutions of the tiles) are set at regular zoom intervals (e.g., 2×, 3×, 4×, etc).

Now consider how we could support arbitrary zoom levels. Since it is not practical to store the tiles at as many resolutions as the possible resolution levels in this case, one solution is to store the tiles at fixed resolution levels, and then scale the tiles to the resolution level corresponding to the zoom level when the zoom level is requested.

There are two choices regarding where the video is scaled. One is to scale the video to the requested size on the server and transmit it to the client. This is a naive solution. Every time a user requests the video at a given zoom level, the server just crops and scales the requested RoI to the resolution level corresponding to the user's request. This solution is costly because the server needs to transcode the original video for every unique zoom level requested. It is therefore not scalable.

The second option is to transmit the original video streams directly to the client and let the client scale down the video itself. This is another naive solution where the server always store the video at the original resolution level. Every time a user requests the video, the server only crops the requested RoI and sends it as a stream. Scaling is done only on the client. This solution is also not scalable due to the bandwidth cost, as even if the user needs a lower resolution video (e.g., watching the video with outer most zoom), a high resolution video still needs to be sent.

Each choice has its advantages and disadvantages. Running video scaling at the server reduces bandwidth, but requires a scaling operation on the server. Scaling the video at the client eases the computational burden on the server, but it requires a higher demand on the network bandwidth. Therefore, a tradeoff exists between computational efficiency of mobile devices and network bandwidth.

The bandwidth and computational demand for the process depends on the resolution levels one choose to encode the tiles in and the zoom levels requested by the users. In scenarios where each zoom level is equally likely to be requested by users, storing the resolution levels at regular intervals makes sense. However, previous studies have found that user's RoI tends to cluster around the certain region at certain zoom levels [2]. In this case, it is more beneficial to encode the tiles at a resolution level that is most likely requested by the users.

The problem considered in this thesis is the following: given the computational and bandwidth constraints, and the zoom levels requested by users, what resolution levels should we encode the tiles in, such that the client can playback the video at the best quality. We focus on the scenario with a single server and single input video.

We model the problem as follows. In our system, the input video is pre-encoded into m resolution levels on the server $R = \{r_1, r_2, ...r_m\}$. Without loss of generality, we assume $r_i < r_j$ if $i < j$. We define the resolution of the video captured as $r_m = 1$. A encoded video has resolution level r_i if the width and height of the video frames are scaled down by a ratio of r_i. For instance, if 1920×1080 is the resolution of the video captured, the re-encoded video with 960×540 has a resolution level of 0.5.

There are n clients, requesting videos at zoom levels $Z = \langle z_1, z_2, ...z_n \rangle$, $z_i \leq z_j$ if $i < j$. Note that two users can request the same zoom level. We define the innermost zoom level, where the user zooms in to the maximum resolution r_m, as zoom level 1. As the users zoom out, the zoom level decreases. If the user zooms to a resolution that is k times smaller than r_m, we say that the zoom level requested is $1/k$. For each zoom level requested z_i, the server sends back the pre-encoded zoom level $f(z_i) \in R$.

The objectives of this thesis are to determine m and R given Z, such that the total computational power required to create the resolution levels in R does not exceed the server computational limit, considering the video quality and the bandwidth cost.

We consider two variants of this problem, called *best-quality streaming* (BQ) and *balanced-scaling streaming* (BS). Best-quality streaming always sends the video back at a higher resolution level than a user's request to the user so as to keep the quality of the returned video best. In other words, $f(z_i) > z_i$. Balanced-scaling streaming tries to find the distribution of the resolution levels for the video to be zoomed and stored on the server by balancing bandwidth and video quality.

1.1 Organization

The rest of the thesis is organized into seven chapters. We begin with a review of literature in Chapter 2, followed by a report on a pilot study conducted to verify the need and usefulness of supporting zoom and pan, with arbitrary RoI cropping in a video stream at arbitrary zoom level in Chapter 3. Then chapter 4 describes arbitrary scaling and its formulation. Chapter 5 and 6 explain how we analyze the cost modeling and solve the problem in simplest situation. In Chapter 7, we present our results. Finally, we conclude in Chapter 8, where several issues encountered in the thesis are discussed and possible future work are explored.

Chapter 2

Related Work

Many studies have been conducted on region of interest (RoI) of an image/video, including its construction, characteristics, and interaction with users. A method for viewing large images on small displays was proposed by Liu et al. [18]. Later, Xie et al. [27] investigated user interest for image browsing on small-form factor devices. Santella et al. [23] presented an interactive method for cropping photographs by eye tracking. Aiming to produce an adaptive video stream for mobile devices with different display sizes, zoomable video, either manual [25] or automatic [5], is studied to enhance video viewing experience on small display; [21] take a further step to investigate ROI prediction strategies for a client-server system.

Apart from generating a multi-resolution representation, Mavlankar et al. studied the optimal slice size for zoomable video in a network streaming context [20]; [6] propose a mechanism to support region-of-interest adaptation of stored video by creating a compression compliant stream while still allowing it to be cropped.

Many recent works have been done on selection of an RoI to zoom into in the context of video. Early research efforts and projects in this area mainly focus on how to crop and pan or automatically determining an RoI, and finally simply zoom the video to proper size. Multiple ROIs support by adopting flexible macroblock ordering is investigated [1]. ROI prediction and recommendation for streaming zoomable video is studied [21, 3]; Meanwhile,[4] tracks the RoI by finding the globally optimal trajectory for a cropping window using a shortest path algorithm. [16] defined a framework that measures the preservation of the source material, and methods for estimating the important information in the video for video retargeting cropping. [26] present a system for automatically extracting the region of interest and controlling virtual cameras control based on panoramic video. [22] discussed the technique for frame accurate cropping of MPEG video. The technique is based on removing temporal dependencies of cropped frames from frames before of after the cropping point while decoding and encoding only the minimum number of frames. To support zoomable video for local playback through the decoding process, [17] implemented a system consisting of an online analyzer and a mobile video player that implements selective decoding in MPEG-4 Part 2 Simple Profile.

On the issue of content scaling of zoomable video streaming, bitstream switching has been proposed as a possible solution. The server encodes the video in multiple resolutions and streams the lowest resolution by default. When the user zooms into the video for RoI from the low resolution video, the corresponding RoI is cropped from a higher resolution video and transmitted. That is, the video server switches between different resolution videos when users zoom in and out. Different approaches have been proposed to encode videos with bit-stream switching in the context of viewing a selected RoI from a high-resolution panoramic video stream [9]. [12] described two new frame types (SP- and SI-frames) defined in H.264/AVC to provide functionalities such as bitstream switching, splicing, random access, error recovery, and error resiliency. [10] proposes a new data format and tile adaptive rate control to achieve high quality partial panoramic video transmission, even over restricted bandwidth networks.

More recently, Khiem et al. studied zoomable video at a network streaming context [15, 13, 14]. Based on user access patterns of ROI, their work focuses on encoding a video intelligently to save bandwidth, thus sharing a common objective as our works. However, the works differ in several aspects. Firstly, the contexts are different. Khiem focuses on arbitrary RoI cropping over the network, while this work deals with arbitrary scaling. Secondly, the approaches are different. Khiem's work simply stores videos in several fixed levels on the server side. Our work, however, dynamically adjust the resolution levels stored on the server according to the current user requests.

In addition, user studies [2, 15] have shown that users interaction is hard to predict, but the users' RoIs are highly similar. We need a streaming solution that can quickly adapt to the large amount of scaling requests changes, have a content independent architecture and be capable of handling any arbitrary scaling ratio. Recent works have been done to improve the throughput by optimizing RoI streaming methods in [14, 7]. However, current video standards do not support arbitrary, interactive scaling required for RoI-based streaming. [11] supports spatially scalable coding with arbitrary cropping, but it does not support interactions because of the pre-determined spatial resolutions and cropping. Few attempts have been made to address the optimization of interactive RoI based arbitrary scaling streaming of encoded video requested by many users.

In summary, extensive research has been done for arbitrary RoI cropping in the network streaming context in zoomable video systems, with focus on video encoding process. To the best of our knowledge, we have not found works on supporting zoomable video for arbitrary scaling.

Chapter 3

User Access Pattern

During the deployment of the Jiku Live sytem, we observe that users' zoom levels tend to cluster around some values. Users tend to zoom into interesting objects or events within the video, and there is a "natural" range of zoom levels to view these objects. This observation forms the basis of our work. Otherwise, if zoom levels requested by the users are uniformly distributed, the server should encode the videos into resolution levels that are uniformly distributed across as well.

To verify this observation, we conducted a preliminary user study.

3.1 Experimental Procedure

We used two 1920×1080 video clips, one of a publicly available (on YouTube) video recording of an open-air stage performance [1], which we named as *Rag&Flag*, and another of common indoor activities, named *Lounge*. In each video, the camera was mounted statically with a fixed view at the center of the site so as to have a full view. Movement of the actors in the

[1]http://www.youtube.com/watch?v=fX2dVlEC8AY

scenes was not explicitly tracked. The video clips were stored on the mobile phone. We implemented a local video player running on the Android platform, displaying the video at a size of 512×288.

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 finger touch gestures. The interface supports arbitrary zooming level and ROI cropping. Resolution 512×288 is the default view (lowest zoom, or zoom level 4/15) and the resolution 1920×1080 (zoom level 1) is the most detailed one.

We invited 50 users to watch the videos and operate freely using our mobile phone. Their interactions with these videos were logged.

Figure 3.1 and 3.2 show examples of zooming. Users can zoom to view the details within the video. The images show screen shots of the video player at different time while a user is watching the videos. The first image for each video shows the whole scene. In the Rag&Flag video, one could zoom into the stage around the vehicle for a clearer view and pan to view another place where people were dancing as the event proceeds. In the *Lounge* video, one might want to zoom into an area in a scene to examine the detail more clearly (e.g., faces of talking people, articles they were passing to each other).

3.2 Zoom Level Distribution

Figure 3.3 shows the overall distribution of aggregated zoom levels from 50 users when they view the two videos. We log the current zoom level every second while the users are watching the videos. The figure shows

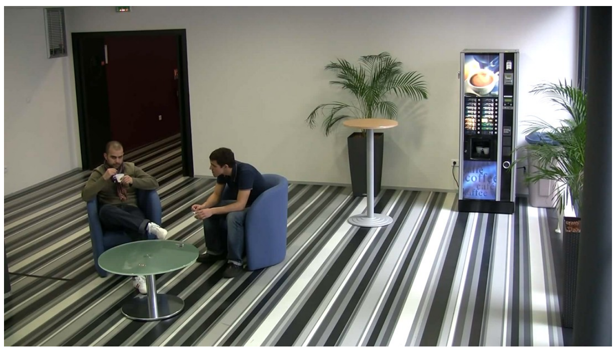
(a)

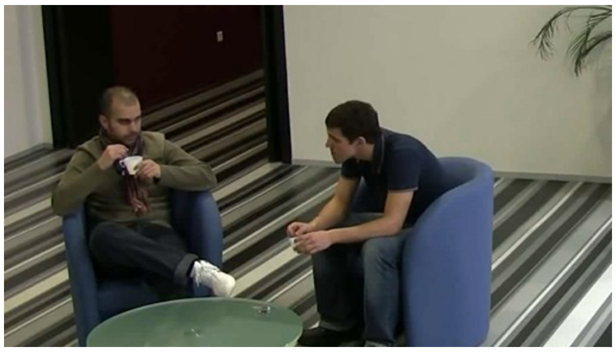
(b)

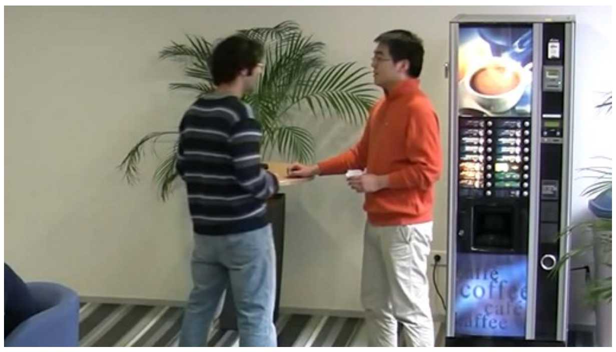
(c)

(d)

Figure 3.1: *Rag&Flag* Video.


(a)


(b)


(c)

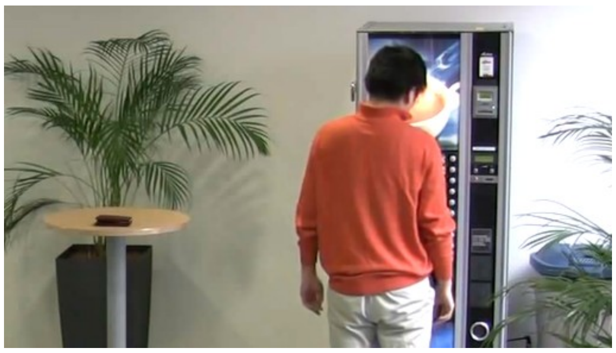
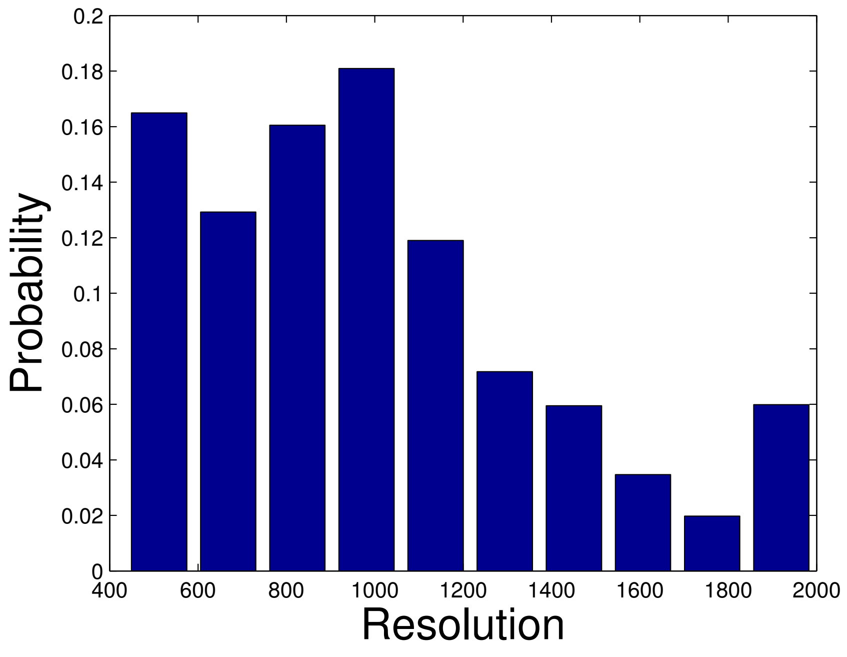
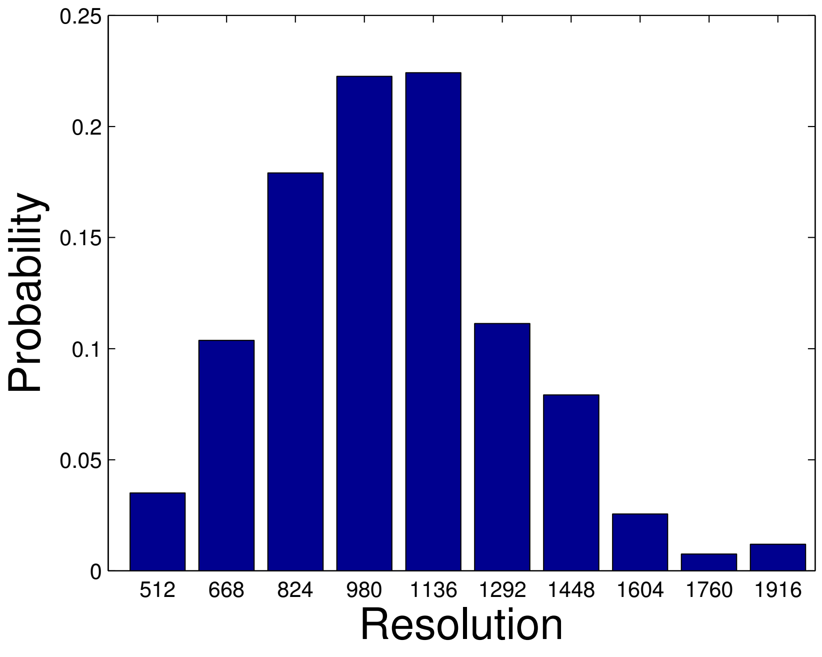

(d)

Figure 3.2: *Lounge* Video



]

(a) Rag&Flag



(b) Lounge

Figure 3.3: User Access Pattern

the percentage of time a zoom level is logged, with the x-axis showing the corresponding resolution of a zoom level. We observe that, on every video, users watch the video with zoomed in most of the time. Besides, the zoom levels preferred by users differ with the video content. This is understandable, *Rag&Flag*, for example, contains much motion. To see the details one needs to zoom in. To see the whole event, one needs to zoom out. People frequently zoom in and zoom out, thus causing that each zoom level is preferred almost equally. Afterwards, it is more comfortable for the user to stay in the middle zoom level since this level arrives at a compromise of both. While the video *Lounge* is dull, and users are only interested in certain zoom levels. So the distribution looks to be clustered.

3.3 Conclusion

Now that we have confirmed that user access patterns are not uniformly distributed, we will proceed to formulate the problem we solve in this thesis.

Chapter 4

Problem Statement and Formulation

In this chapter, we first intuitively explain the optimization problem that arises in supporting arbitrary zooming. We then explain how the cost model is built. Finally, we formulate the problem.

Table 4.1 lists the major symbols that will be used in the rest of the thesis, other symbols will be introduced when encountered.

4.1 The Problem

As discussed, we want to determine the resolution levels to encode the input video in, given the set of current zoom levels requested by the users and the computational constraints, to improve the quality of the video received by the users and to reduce the bandwidth. To solve this problem, we can partition it into two sub-problems.

The first is how to get the appropriate number of resolution levels of the video stored on the server. The maximum number of levels a server can support essentially depends on the computational resources of the server allocated to this task.

Given the number of resolution levels, the second sub-problem is to determine the resolution levels to encode the video in. We consider a simple version of the problem where there is only one server and one input video. There are several variations to this problem, depending on how a requested zoom level is mapped to the resolution level (the function $f()$ in Chapter 1).

Consider a user request to view the video at zoom level z, where $0 \leq z \leq 1$. The nearest two resolution levels stored on the server is r and r', such that $r < z < r'$. We can either send the video at resolution level r and let the client scale it up and play it back, or send the video at resolution level r' and let the client scale it down before playback. Sending resolution r leads to lower bandwidth and lower video quality, while sending resolution r' leads to higher bandwidth but better quality.

In an ideal case, there exist a video with resolution level z on the server. In which case the video with resolution level z is sent.

We model the cost of bandwidth, computation, and video quality as follows. The bandwidth cost depends not only on the resolution level sent r, but also on the zoom level requested z. We denote the bandwidth cost as $C_b(z, r)$. The loss of video quality is modeled as $C_q(z, r)$.

The computational cost is mainly due to scaling, encoding, and analysis of the video. We can generally model the scaling cost as $E_s(r, r')$, if the video of resolution level r is scaled down to level r'. We show in the next chapter, however, that the scaling cost only depends on the target resolution. We can therefore simplify the scaling cost as $E_s(r')$. The computation costs to encode and analyze the video depends only on the resolution r' of the video, and are denoted as $E_e(r')$ and $E_a(r')$ respectively. The computational cost are normalized such that the total computational capacity of the server is 1. We denote $C_p(r)$ as the total computational cost for creating a video with resolution level r.

Table 4.1: Notations

Notation	Description
B	The available maximum bandwidth coming out from one computer
$E_s(r_1, r_2)$	The computational time of scaling from resolution level r_1 to resolution level r_2 in one computer
$E_e(r)$	The computational time of encoding a one second video at resolution level r in one computer
$E_a(r)$	The computational time of analyzing a one second video at resolution level s in one computer
$C_p(r)$	The computational time of creating a video with resolution level r.
$C_b(z, r)$	The bandwidth cost when the level of the desired video by user is z but the level of the retrieved video is r
$C_q(z, r) \geq 0$	The quality cost when the zoom level of the desired video by user is z and the level of the retrieved video is r
Z	The set of zoom levels requested by users at a moment.
z_i, n	The zoom level of video requested by user i and number of users n. $z_i \in Z$
R	The set of resolution level on the server.
r_i, m	The resolution level i of stored videos and number of levels m. $r_i \in R$
$f(z_i)$	The resolution level send back to the ith user. $f(z_i) \in R$

4.2 Problem Formulations

We now present two different formulations of the problem.

First, consider we want to maximize the video quality at the clients. In other words, we want to minimize the loss in quality, subjected to bandwidth and computational constraints. We can formally formulate the problem as follows. Given n users, requested for zoom levels $z_1, z_2, ..., z_n$ respectively, find R, the resolution levels to encode the video in $R = \{r_1, r_2, ..., r_m\}$, to

$$\text{minimize } \sum_{i=1}^{n} C_q(z_i, f(z_i))$$

subjected to:

$$\begin{aligned} \sum_{i=1}^{n} C_b(z_i, f(z_i)) \leq B \\ \sum_{i=1}^{m} C_p(r_i) \leq 1 \end{aligned} \tag{4.1}$$

where $r_i \in \{f(z_i) | i = 1..n\}$.

It is also possible to formulate the problem, to jointly minimize bandwidth and quality loss. Suppose we define α, $0 \leq \alpha \leq 1$, as the factor that represent the relative importance of bandwidth and quality, we can reformulate the problem as:

$$\text{minimize } \sum_{i=1}^{n} (\alpha C_b(z_i, f(z_i)) + (1 - \alpha) C_q(z_i, f(z_i)))$$

subjected to:

$$\sum_{i=1}^{m} C_p(r_i) \leq 1 \tag{4.2}$$

where $r_i \in \{f(z_i) | i = 1..n\}$.

When $\alpha = 0$, Equation (4.2) turns out to be Equation (4.2). As α decreases, the importance of quality increases. As α increases, bandwidth becomes more important.

Chapter 5

Modeling

Before we proceed to the solution, we first explain how we model the cost functions for bandwidth, computation, and video quality.

To determine these functions, we build an off-line training system to analyze the running time, bandwidth usage, and resulting PSNR of video while transmitting video streams in different resolution levels and different RoIs. We then build a regression model for approximating the bandwidth, video quality, and computational cost.

5.1 Bandwidth Cost Modeling

The function for cost of bandwidth is denoted as $C_b(z, r)$, and refers to the bandwidth when the zoom level requested by the user is z but the resolution level of the transmitted video r. We proposed two methods for bandwidth cost modeling.

The first, simpler, method estimates the bandwidth cost based on the resolution of the video alone, without considering the content. As the content

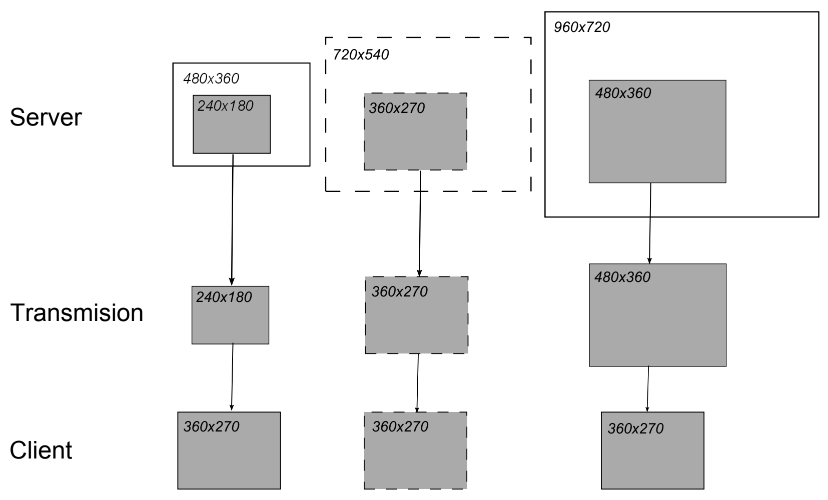


Figure 5.1: Different Choices of Resolution Level to Send When Users Requested Zoom Level Corresponding to Resolution 720×540.

of the video has variable background and motions, taking the content into consideration would lead to a more sophisticated method.

For example, if the original resolution of input video streams is 1280×960, we transcode and store them in two resolution levels 0.375 and 0.75, corresponding to resolution 480×360 and 960×720. Meanwhile, the resolution of ROI displayed on the mobile is constrained as 360×270. User requests video streams at zoom level 0.5625 (720×540). Thus, the server should crop a video of which the size is 360×270 from the video stored on the server of which the size is 720×540. However, this level does not exist in the server. So the server has to choose from resolution 480×360 or 960×720, then crop and transmit RoI of the stored video to the client. The client receives the video streams and scale it to 360×270. Figure 5.1 shows how the transmitted part is scaled while cropping from different resolution levels. As can be seen, the bandwidth cost transmitted from the server is related to both z and r.

Let the bandwidth cost transmitted back to user as I if we already have stored the video at the resolution 720×540 on the server. Then the bandwidth required of the transmitted video is modelled as:

$$C_b(z, r) = (r/z)^2 \cdot I \tag{5.1}$$

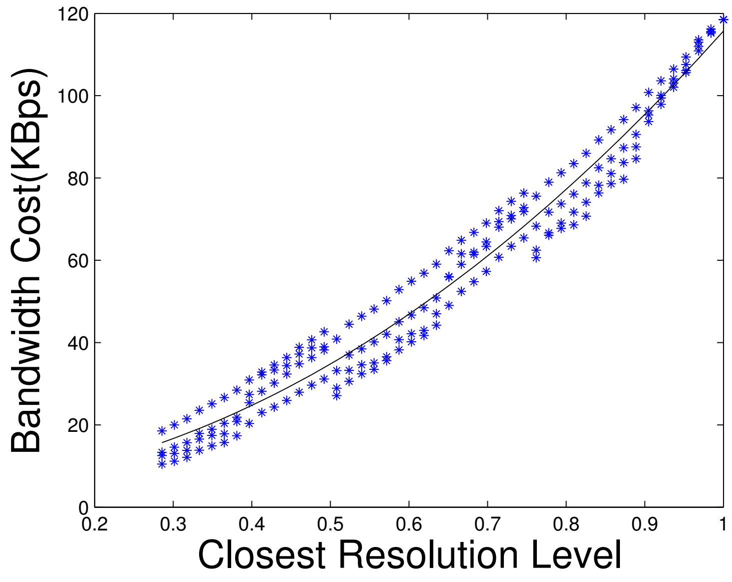


Figure 5.2: Curve Fitting for Bandwidth Cost ($z = 0.28$)

The second method is to estimate the bandwidth considering content. In this method, the bandwidth cost is related not only to z and r, but also the content in the video streams. We assume that the content of the previous video segment is similar to the current segment, and therefore use the RoI size C_b^{prev} of the previous segment to estimate RoI size C_b of the current segment.

Figure 5.2 depicts how bandwidth cost increases while the closes resolution level increases. Regression analysis is introduced here to form the cost function. Equation (5.1) indicates that it is reasonable to use polynomial curve fitting by r/z to estimate the bandwidth. The degree of the approximating polynomial should be 2. We finally construct the function as shown in Equation (5.2). Coefficients a, b, c are trained from experiments. In our system, $a = 0.6232, b = 0.3708, c = 0.0060$. As shown in Figure 5.2, the curve of our function fits well to the real bandwidth cost.

$$C_b(z, r) = C_b^{prev} \cdot (a \cdot (r/z)^2 + b \cdot (r/z) + c) \tag{5.2}$$

Note that I in the first method is different from C_b^{prev} in the second method. I actually do not consider too much about the content of the video. In the example, we can either assign it a constant value as 1 for simplicity or the video size of the original video segment by the constant scale ratio of video played on the mobile to make it more reliable. On the contrary, C_b^{prev} is the real size of an RoI segment of a continuous video stream in the previous second. So we usually have to analyze the video stored on the server to compute the bandwidth of the RoI when the RoI changes. The second method is therefore more complicated and more computationally intensive.

We use both cost models in our solution. We use the first one when for determining the resolution level to store, since this decision is needed frequently (every second in our implementation), and we periodically use the second, more expensive method, to determine how many resolution levels we need to store (in the order of minutes).

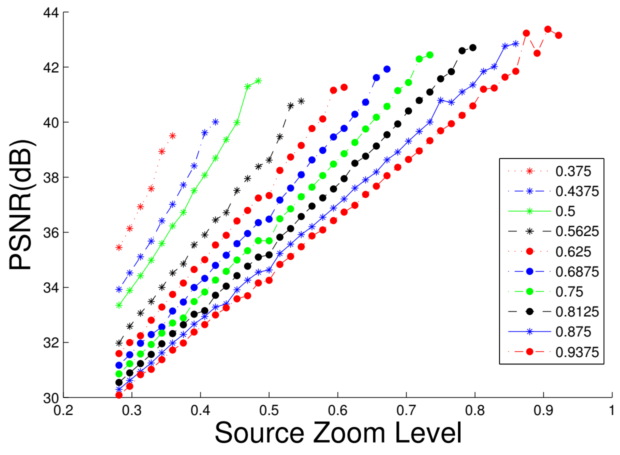
5.2 Quality Cost Modeling

When the client receives the video from the server, it may need to scale it to the right size. However, if the client scales up the video, there will be quality loss. We use the differences in PSNR to measure the quality of lossy scaling on the client. The loss is denoted as $C_q(z, r) \geq 0$, which means the quality loss when the zoom level of demanded video by user is z and the level of transmitted video is r.

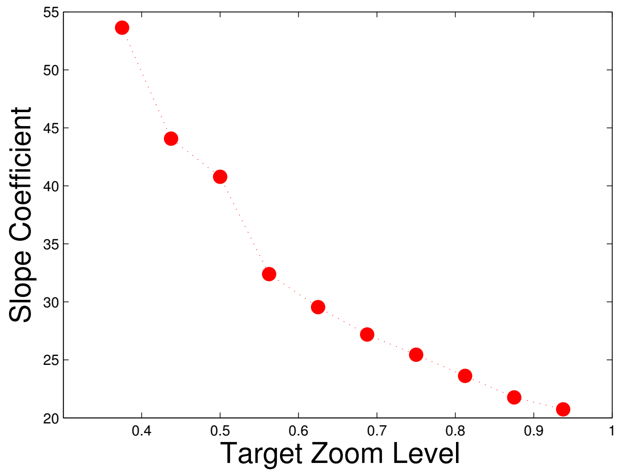
If $z < r$, a user receives a video with a resolution level higher than needed. In this case, we set $C_q(z, r)$ to zero since scaling down from higher resolution video will not result in quality loss. If $r < z$, $C_q(z, r)$ is non-zero. Further, $C_q(z, r)$ should be monotonically decreasing with increasing r, reaching zero when $z = r$.

Our experiments show that we can estimate PSNR based on r and z. Figure 5.3(a) presents the PSNR results while scaling up to different resolution levels. Each data point shows the PSNR between an image scaled up from a source resolution level to the target resolution level and the image at the target zoom level. The X-axis shows the source resolution levels. There are 10 lines in the figure and points on the same line have the same target resolution level. It can be easily found that the each of the lines can be formulated by linear regression. We calculate the slope and intercept of those lines and depicts the results separately in Figures 5.3(b) and 5.3(c). The 10 points in the each of the two figures present the 10 lines. These points fit a polynomial curve of the second degree. An equation to estimate the quality cost is finally obtained.

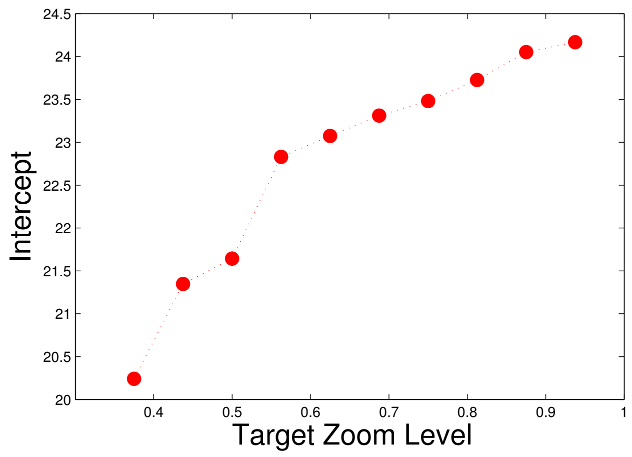
Our experiments also show that PSNR value is related to the image content; however, curves for different images behave similarly. In our model, we add a multiplier to model the effect of image content.



(a) PSNR values if image Scale Up



(b) Slope of Regression Lines



(c) Intercept of Regression Lines

Figure 5.3: PSNR of Images After Scaled Up

5.3 Computational Cost Modeling

The original video streams received by the server from the cameras need to be transcoded and stored on the server in one-second segments in several resolution levels. The whole procedure mainly consists of video scaling, encoding, and analysis. The computational cost is measured by the time spent on processing a one-second video segment. Since the computational cost depends on individual host, we need to train our model on every computer used in the system. A regression model similar to that for bandwidth is then built. The regression analysis is the same as that for the quality regression model. We find that we can estimate the computational cost based on the scale ratio of resolution level r_i and the original resolution r_1.

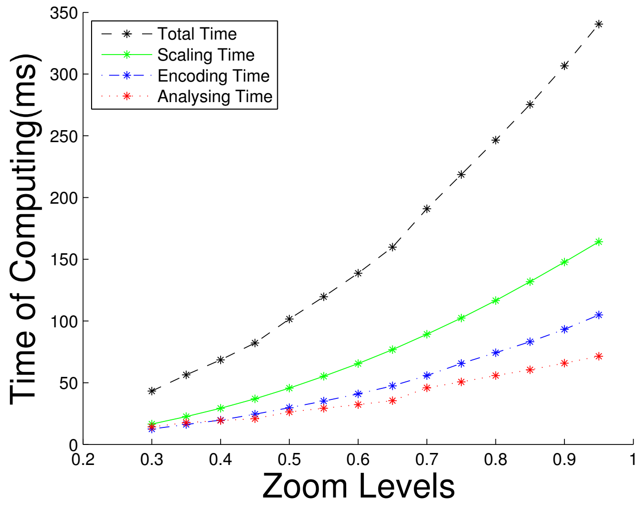
To build this function, we make 184 measurements that compare the observed actual computation time on our server for different input videos. The data in each measurement contain the costs separately for video scaling, encoding, and analysis of a 60-seconds video.

Figure 5.4(a) demonstrate the time cost of the main parts in the system. The total computational cost at a resolution level r can be computed as:

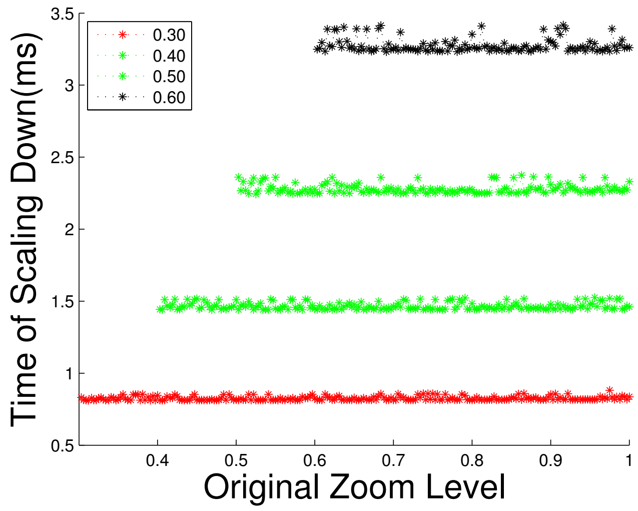
$$C_p(r) = E_s(r_m, r) + E_e(r) + E_a(r) \tag{5.3}$$

where $E_s(r_m, r)$ is the cost of scaling the video resolution from r_m to r, $E_e(r)$ is the cost of encoding the video at resolution level r, and $E_a(r)$ is the cost of analyzing the video at resolution level r.

Note that E_s depends on the scaling algorithm we chose. For a particular algorithm, the computational cost of the scaling is related to both the size of source image and the target image. Scaling down from an image of lower



(a) Computing Time for Different Resolution Levels



(b) Scaling Down Using Naive Bilinear Algorithm

Figure 5.4: Computational Cost

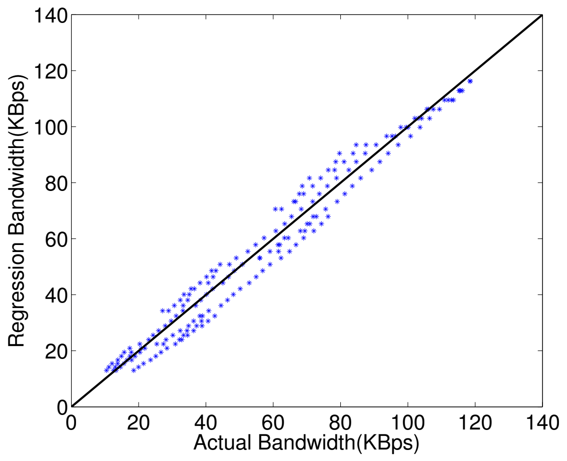
resolution cost less than scaling down from an image with a larger resolution, when the target resolution is constant. Thus, $E_s()$ should depend on both the source and target resolution. In our system, however, we use a naive bilinear scaling algorithm without any optimization of our system. Figure 5.4(b) shows the result of scaling down using this algorithm. Each data point represents a single experiment of scaling down one image from resolution level r_i to a target resolution level r_j. The X-axis represents the source resolution levels. These points approximately form 4 lines, points in the same line have the same target resolution level. These data reveal that the computational cost of scaling is only related with the target size of image for our algorithm, and we can always scale down from the video of original size. We therefore model E_s as a function on r only.

5.4 Model Validation

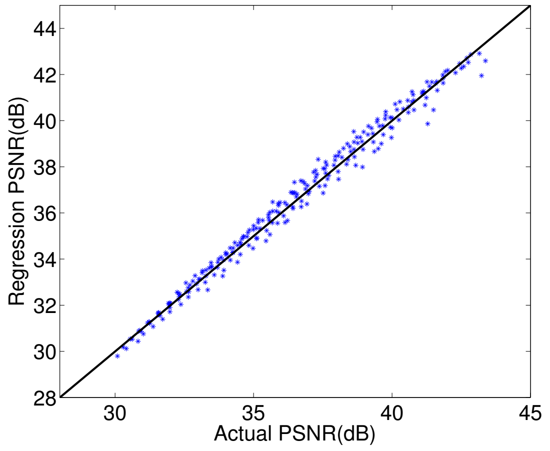
We make a series of bandwidth, quality, and computational time measurements for different input videos. On computational cost, a multi-factor regression model is fitted for the computer hardware. The regression models predict bandwidth, quality loss, and computational time. We use the model to predict the cost of bandwidth, quality, and computational time in the next second.

Around 100 to 200 measurements are made for each regression model. Figure 5.5(a) summarize the results of these measurements for bandwidth under the second approach mentioned. Each data point represents a single validation experiment result in which a single bandwidth cost estimation was compared to an actual value. The goodness of fit values for the figure is 93.51. As shown in Figures 5.5(b) and 5.5(c), similar results were found for quality loss as well.

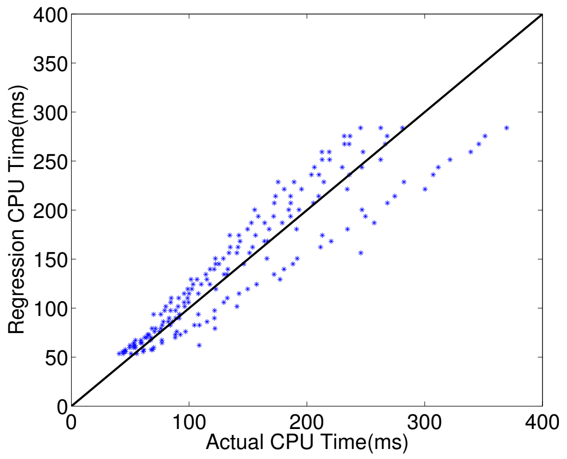
Figure 5.5(c) indicates that the prediction of computational cost is not quite accurate. In practice, however, we overestimate the computational cost so that we can still generate the videos of different resolution levels within the stipulated time, considering the prediction error.



(a) Bandwidth Cost



(b) Quality



(c) Computational Cost

Figure 5.5: Accuracy of Model

Chapter 6

Methodology

With the cost functions determined, we can now focus on the main issues of the paper: given the requested zoom levels Z and the system constraints, find R, the resolutions encode the videos in. As we described before, we have two sub-problems, finding m and finding R given m. For each problem, we have two variations, to optimize the video quality (Equation (4.2)) or to optimize a weighted sum of video quality and bandwidth (Equation (4.2)).

We first focus on the case where m is known, and consider the two optimization problems.

6.1 Optimizing Bandwidth

When quality is required to be much more important than bandwidth, we only consider the cost of quality for optimization. The server always sends the video of the best quality to the user. We name this strategy as *best quality streaming* (**BQ**). Equation (4.2) is our objective function.

We solve this problem using dynamic programming. Supposing we have set of resolution levels $R = \{r_1, r_2, ..r_m\}$, If we set $f(z_i)$ to the nearest r_k where $r_k \geq z_i$, then the server always sends a higher quality of video to the user. There will be no quality loss. We define an indicator variable $x_{i,j}$ as follows:

$$x_{i,j} = \begin{cases} 1 & if\ r_{j-1} < z_i \leq r_j \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

Our goal then is to find R to minimize the bandwidth:

$$OPT(n, m) = min \sum_{r_j \in R} \sum_{z_i \in Z} x_{i,j} C_b(z_i, r_j), \tag{6.2}$$

We observe that every element in R must be equal to some elements in Z. A naive solution is thus to exhaustively try all possible solutions of R, but this solution would lead to a $\binom{n}{m}$ search space.

We can solve this problem using dynamic programming in $O(nm)$ time. Let $\mathcal{B}(i, j)$ be the minimum bandwidth cost considering the last $n - i + 1$ users $(z_i, ..., z_n)$ and j resolution levels are stored on the server, then:

$$\mathcal{B}(i, j) = \begin{cases} \min\limits_{k=i}^{n-j+1} \mathcal{B}(k+1, j-1) + \sum\limits_{l=i}^{k} C_b(z_l, z_k) & j > 1 \\ \sum\limits_{k=i}^{n} C_b(z_k, z_n) & j = 1 \end{cases}$$

The intuition behind the recurrence above is as follows. If $j = 1$, we only store one resolution level on the server – the one that corresponds to the highest zoom level requested, z_n (i.e., $r_1 = z_n$). The total bandwidth cost can be calculated in a straightforward manner, by summing the cost of each zoom level with respect to z_n. If $j > 1$, then we first try to decide the smallest resolution level to store on the server (which could be any level between z_i to z_{n-j+1}). Each possible level k splits the list of zoom levels in two, whose costs are computed separately. The cost for the zoom levels z_i to z_k is the sum of the individual bandwidth cost of each zoom level; the bandwidth cost for zoom level z_{k+1} to z_n is then computed recursively.

$\mathcal{B}(1, m)$ gives the minimum bandwidth cost. By tracing the value of variable k which leads to this minimum cost, we can find the values of z_k that should be included in R.

We can make the algorithm runs faster by approximating the solution. In cases where n is large, we can group zoom levels that are near each other at a single level. We define a parameter θ, the step size for R. This is a value of the minimum distance between any r_i and r_j. That is, $|r_i - r_j| > \theta$. We use θ to reduce the number of elements in Z, grouping zoom levels into buckets of size θ.

6.2 Optimizing Bandwidth and Quality

We now present the *balanced-scaling streaming* (**BS**) that balances quality and bandwidth. The cost function for this method is shown in Equation (4.2). The cost function is asymmetric. Motivated by the nature of the k-means algorithm, we proposed our clustering method for this asymmetric cost function. It starts from random choice of m zoom levels from $z_1, ..., z_n$ as cluster centers. These centers are initial centroids. The centroids represent R, the set of resolution levels stored on the server. There are two alternating steps in this method.

Step 1 Allocating elements to clusters: Traverse all the zoom levels requested and allocate each request z to the nearest centroids:

$$min_{r \in R}\left(\alpha C_b(z,r) + (1-\alpha)C_q(z,r)\right) \tag{6.3}$$

z will be assigned the cluster centered at r. After this step, we created m clusters, denoted as Z_1, Z_2,..., Z_m.

Step 2 Find new centroids for clusters: New centroids for each cluster Z_i is calculated by minimizing the mean squared error (MSE) for each cluster:

$$\arg\min_{r \in Z_i} \sum_{z \in Z_i} \left(\alpha C_b(z,r) + (1-\alpha)C_q(z,r)\right)^2 \tag{6.4}$$

The new centroids, r, from each cluster forms the new R.

After each cycle, a value of the following mean-squared-error objective function needs to be computed in order to track the convergence of the whole clustering process:

$$min \sum_{r \in R} \sum_{z \in Z} \left(\alpha C_b(z,r) + (1-\alpha)C_q(z,r)\right)^2 \tag{6.5}$$

In order to guarantee the monotonic property of the k-means algorithm, both steps should be carried out with the same loss function (Equation 6.3). The two steps will be repeated until the termination condition is met. We define the termination condition reaching convergence of objective function (Equation 6.5).

To understand the behavior when the cost function of the k-mean algorithm is asymmetric, there is an exhaustive study conducted in [19]. Our method cannot guarantee that the clustering process will converge to an optimal solution. It can only assure local optimality which depends on the initial centroids of user requests.

6.3 Joint Optimization of m and R

We have previously assumed that number of resolution levels m to store on the server is pre-determined. We show how we can determine m.

Since larger m would lead to better video quality, we want to maximize m under the constraint of computational cost. Figure 5.4(b) shows that the time cost monotonically increases with respect to the resolution level. It implies we may be able to find the boundary for the number of resolution levels. In our system, we define the minimum resolution level as r_{min} and the maximum resolution level as r_{max}. The two parameters help us bound the boundaries for R. That is,

$$\frac{1}{C_p}(r_{max}) < |R| < \frac{1}{C_p}(r_{min}) \tag{6.6}$$

While the number of resolution levels m to store depends mainly on the computational cost, the computational cost, however, depends not only the number of levels, but also what these levels are. It is therefore not possible to determine only m. Both R and m have to be jointly determined.

Our solution to find the maximize m under the computational cost constraint is to use binary search within the range of m in Equation (6.6). For each m, we use the algorithms described in previous sections to find the optimal R, and check if, for this optimal R, whether the computational cost is satisfied.

Algorithm 1: FindResolutionNumber(lower bound L, upper bound U)

```
1: if $L = U$ or $U - L > 1$ then
2:     check if $L$ or $U$ is a valid solution and return a valid solution.
3: end if
4: $mid \leftarrow (L + U)/2$
5: {check if $mid$ is a valid solution}
6: $R \leftarrow$ FindResolutionLevels$(Z, mid)$
7: $C \leftarrow \sum_{r \in R} C_p(r)$
8: if $C > 1$ then
9:     return FindResolutionNumber$(L, mid)$
10: end if
11: return FindResolutionNumber$(mid, U)$
```

Here, FindResolutionLevels() invokes one of the algorithms to determine R given m, as described in the previous sections.

The algorithm to jointly find the optimal m and R is expensive. We therefore do not run this algorithm for every video segment. Instead, in practice, we run the algorithm periodically every 30 seconds. Between the runs, we assume that the optimal m remains the same and only optimizes R.

There is a problem with this algorithm if we directly use it to search for m because we do not consider the bandwidth constraint. There are ways to extend this algorithm to consider the bandwidth constraint. In our implementation, however, we did not consider the bandwidth constraint.

Chapter 7

Performance Evaluation

With the algorithms described in Chapter 6 implemented, we carried out experiments to evaluate the performance of our system that supports arbitrary zoom levels in zoomable video. This chapter describes these experiments and presents the results.

As stated in Chapter 1, the objectives of our work are to improve the quality of video streams that are delivered from the server to the users in one request period under computational resource constraints, with bandwidth either as a constraint or another optimization objective. We experimentally evaluate these two aspects in this chapter.

7.1 Methods for Comparisons

Static Scaling (SS) Our experiments compare our methods with the baseline methods used in the existing Jiku Live system, which we called static scaling. In this method, the resolution levels stored on the server are static and do not depend on Z. This method corresponds to that described by Shafiei et al. [24]. We pick three resolution levels $R = \{r_1, r_2, r_3\}$, with $r_2 = (r_1 + r_3)/2$. This method works well when there are many users requesting at the same time. First, the bandwidth is reduced if the user zooms out, as only the lower resolution levels need to be sent. Second, by only re-encoding the input videos to three resolution levels only, the burden of the server is eased.

Dynamic Scaling (DS) This is the method proposed in this thesis, where the number of resolution levels m and the resolution levels are dynamic, depending on Z.

In addition to how we fix R, the performance of the methods also depends on $f()$, how we map from the requested zoom levels to the stored resolution. As explained, we have two variations of objective functions:

Best Quality (BQ): The server always sends the best-quality video to the client, where $f(z_i)$ maps to the resolution level that is next higher than z_i.

Balanced Strategy (BS): The server trades off the bandwidth and video quality, and pick a resolution level that is "close" to the zoom level, depending on the cost function (Equation 4.2).

Considering objectives of the system, the combination of these gives 4 algorithms: **SS-BQ**, **SS-BS**, **DS-BQ**, **DS-BS.**

In our experimental system, **SS-BS** stores the videos at resolution 512×288, 1216×684, and 1920×1080. The three levels for **SS-BS** corresponds to the minimum, middle, and maximum resolution levels. For **SS-BQ**, since the server always sends the next higher resolution level to the client, storing the minimum level is not useful. We have thus divided the possible resolution levels into three equal size ranges, with four resolution levels at the boundary (512×288, 960×540, 1440×810, and 1920×1080). We store the three larger resolution levels out of these four on the server.

We use a pixel gap of 4 pixels, i.e., a θ of 4/1920, as the step size for grouping R. We use 0.0001 as the threshold to terminate the clustering algorithm.

Our experiment runs on a Mac Pro 4.1 with a 2.66 GHz Quad-Core Intel Xeon processor, 8 GB RAM, running Mac OS X 10.6.8. Every core is treated as one computer in our experiments.

The system performance depends on the zoom levels requested. Chapter 3 presents the data gathered from user study where 50 users watched two videos, and their current zoom levels and RoIs are logged every second. Our evaluation is conducted using these traces.

7.2 Findings and Discussion

The generated zoom level requests are fed into the systems every second as Z. We run two sets of experiments, one for each input video.

Figures 7.1 and 7.2 compares the results for both videos. In these experiments, we set $m = 3$, and vary α from 0 to 1.

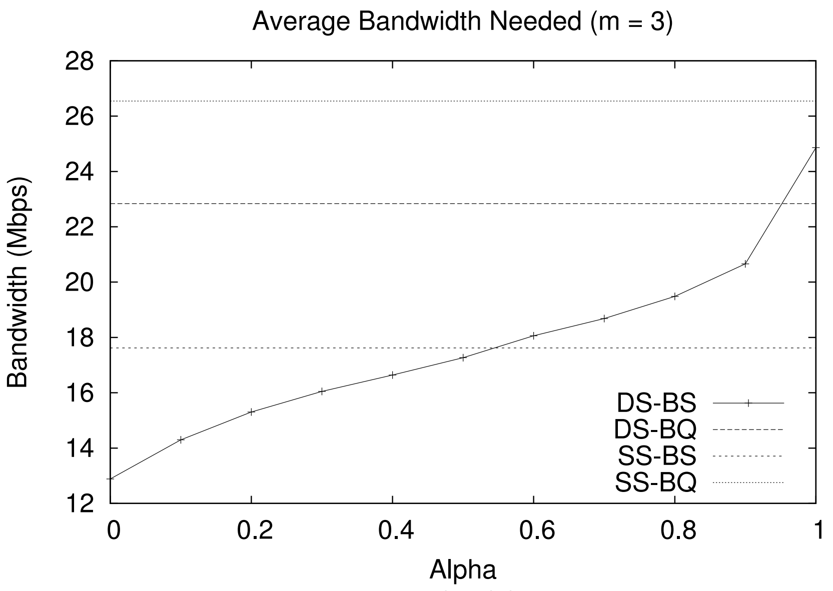
For the *Rag&Flag* video, comparing **SS-BQ** and **DS-BQ**, we can see that **SS-BQ** leads to larger bandwidth (at least 3.2 Mbps more than **DS-BQ**, but with no significant improvement in quality (less than 1 dB difference).

The results for **DS-BS** shows that, we can tune the tradeoff between bandwidth and quality, and is roughly equivalent to **SS-BS** when alpha is between 0.4 and 0.5.

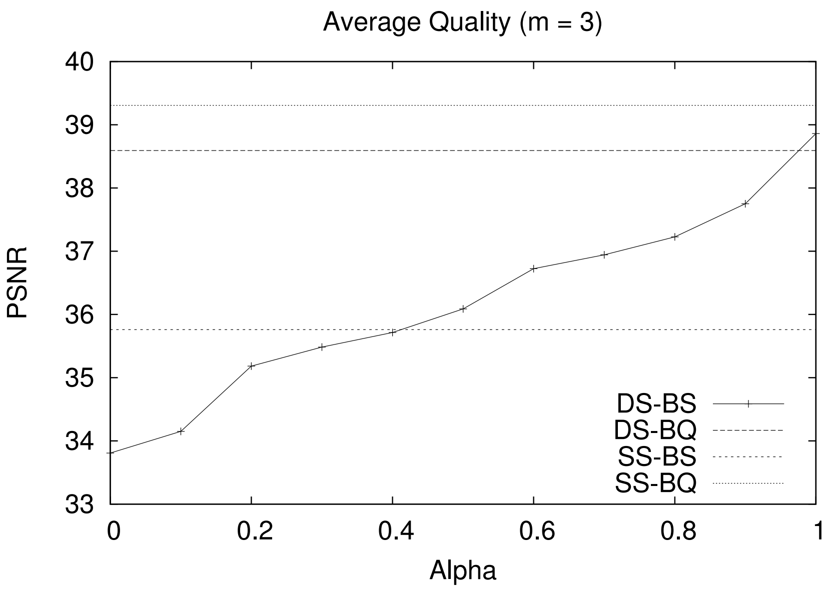
The results for the *Lounge* video shows the same relative relationship among the different schemes and leads to the same conclusion as the Rag&Flag video.

The previous results are for $m = 3$. Our model for computational cost indicates that our server is capable of re-encoding the input videos into 15 different resolution levels, or $m = 15$. Fixing $m = 15$ and $\alpha = 0.5$, we rerun the experiments with dynamic scaling. The results are shown in Figure 7.3.

Figure 7.3 plots the quality of the video against bandwidth needed. Points that gravitate towards to top left corner are more desirable, since they indicates better quality with lower bandwidth. For both video clips we tried, we found that **DS-BQ** with $m = 15$ as determined by the algorithm has the best tradeoff between quality and bandwidth.

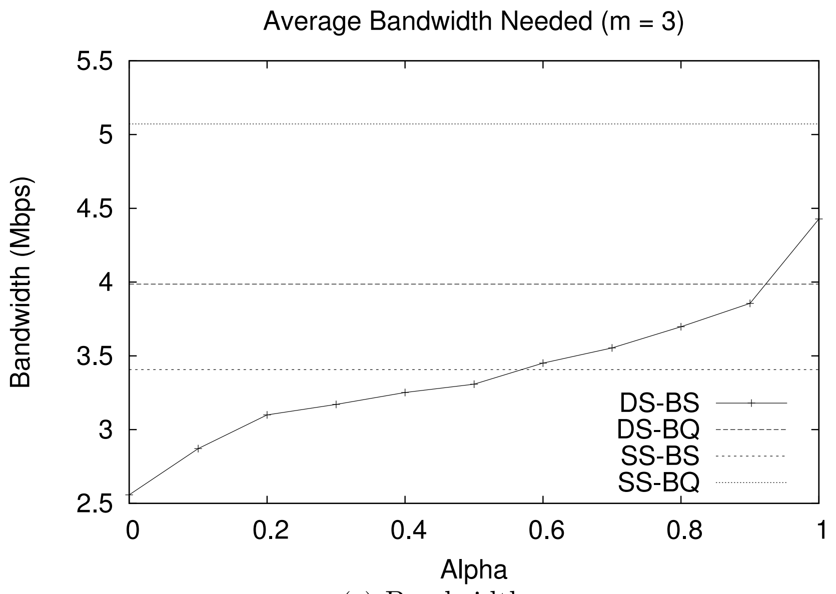


(a) Bandwidth

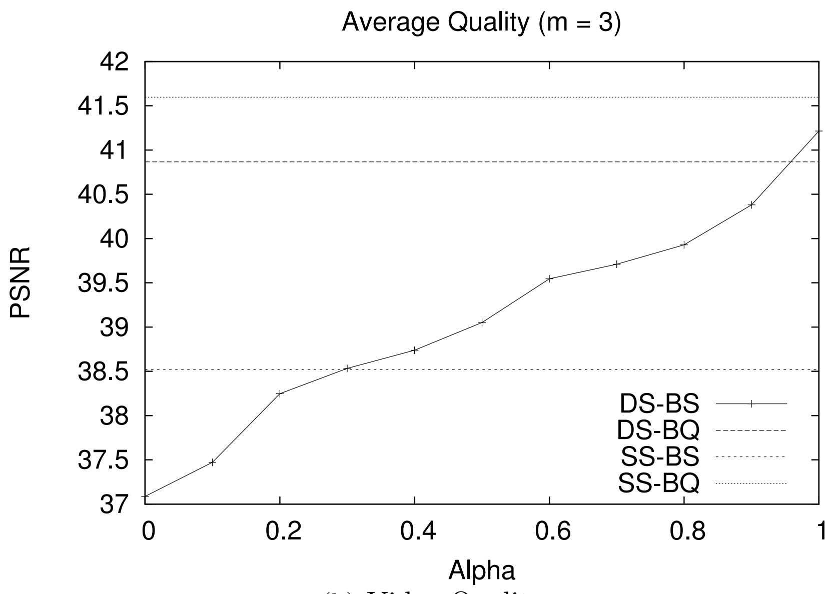


(b) Video Quality

Figure 7.1: *Rag&Flag* Video

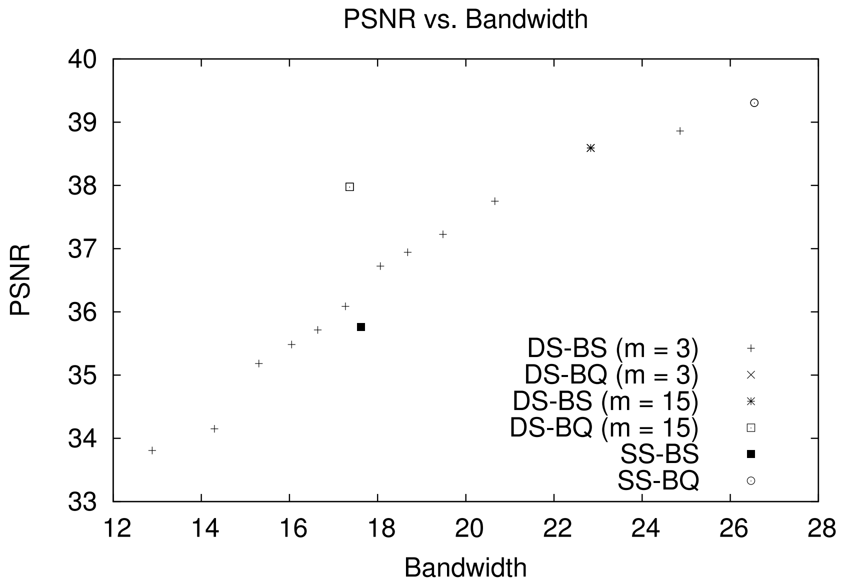


(a) Bandwidth

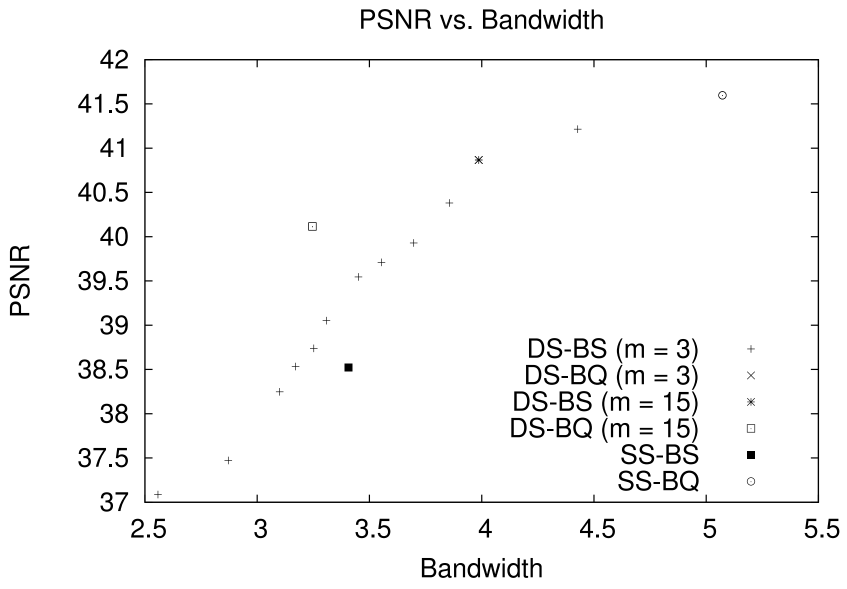


(b) Video Quality

Figure 7.2: *Lounge* Video



(a) *Rag&Flag*



(b) *Lounge*

Figure 7.3: Quality vs. Bandwidth

Chapter 8

Conclusion and Future Work

In this thesis, we study the problem of resolution level selection for arbitrary scaling in the live zoomable video. The contributions of this thesis are twofold. First, we conducted a user study with 50 users and reveal the user behavior with watching two video clips with zoomable interfaces on a mobile phone. The study reveals that users tend to zoom but the distribution of zoom level is not uniform. Second, using the requested zoom levels as input, we designed, implemented, and evaluated online algorithms for deciding the resolution levels under computational constraints, considering both bandwidth and quality.

The following issues are still open. First, we did not consider the bandwidth constraints when we jointly optimizes for m and R. Second, we only consider a version of the problem with one server and one camera. We would like to extend the problem to multiple servers and cameras. Third, more user studies should be carried out, on more video clips.

Bibliography

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

[2] A. Carlier, R. Guntur, and W. T. Ooi. Towards characterizing users' interaction with zoomable video. In *Proceedings of the 2010 ACM Workshop on Social, Adaptive and Personalized Multimedia Interaction and Access*, SAPMIA '10, pages 21–24, Firenze, Italy, 2010.

[3] A. Carlier, G. Ravindra, V. Charvillat, and W. T. Ooi. Combining content-based analysis and crowdsourcing to improve user interaction with zoomable video. In *Proceedings of the 19th ACM international conference on Multimedia*, MM '11, pages 43–52, Scottsdale, Arizona, USA, 2011. ACM.

[4] H. El-Alfy, D. Jacobs, and L. Davis. Multi-scale video cropping. In *Proceedings of the 15th ACM International Conference on Multimedia*, MM '07, pages 97–106, Augsburg, Germany, 2007.

[5] X. Fan, X. Xie, H. Zhou, and W. Ma. Looking into video frames on small displays. In *Proceedings of the 11th ACM International Conference on Multimedia*, MM '03, pages 247–250, Berkeley, CA, USA, 2003. ACM.

[6] W.-C. Feng, T. Dang, J. Kassebaum, and T. Bauman. Supporting region-of-interest cropping through constrained compression. *ACM Transactions on Multimedia Computing, Communications and Applications*, 7(3):17:1–17:16, Aug. 2011.

[7] R. Guntur and W. T. Ooi. On tile assignment for region-of-interest video streaming in a wireless lan. In *Proceeding of the ACM Workshop on Network and Operating Systems Support on Audio and Video*, NOSSDAV'12, Toronto, Canada, 2012.

[8] S. Halawa, D. Pang, N.-M. Cheung, and B. Girod. ClassX: an open source interactive lecture streaming system. In *Proceedings of the 19th ACM International Conference on Multimedia*, MM '11, pages 719–722, Scottsdale, Arizona, USA, 2011.

[9] S. Heymann, A. Smolic, K. Mueller, Y. Guo, J. Rurainsky, P. Eisert, and T.Wiegand. Representation, coding and interactive rendering of high-resolution panoramic images and video using MPEG-4. In *Proceedings of the Panoramic Photogrammetry Workshop PPW'05*, Feb 2005.

[10] M. Inoue, H. Kimata, K. Fukazawa, and N. Matsuura. Interactive panoramic video streaming system over restricted bandwidth network. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, pages 1191–1194, Firenze, Italy, 2010.

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

[12] M. Karczewicz and R. Kurceren. The SP- and SI-frames design for H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):637–644, 2003.

[13] N. Q. M. Khiem, R. Guntur, A. Carlier, and W. T. Ooi. Supporting zoomable video streams with dynamic region-of-interest cropping. In *Proceedings of ACM Multimedia Systems*, MMSys'10, pages 259–270, Scottsdale, Arizona, USA, 2010.

[14] N. Q. M. Khiem, R. Guntur, and W. T. Ooi. Adaptive encoding of zoomable video streams based on user access pattern. In *Proceedings of ACM Multimedia Systems*, MMSys'11, pages 211–222, San Jose, CA, USA, 2011.

[15] N. Q. M. Khiem, R. Guntur, and W. T. Ooi. Towards understanding user tolerance to network latency in zoomable video streaming. In *Proceedings of the 19th ACM International Conference on Multimedia*, MM '11, pages 977–980, Scottsdale, Arizona, USA, 2011.

[16] F. Liu and M. Gleicher. Video retargeting: automating pan and scan. In *Proceedings of the 14th ACM International Conference on Multimedia*, MM '06, pages 241–250, Santa Barbara, CA, USA, 2006. ACM.

[17] F. Liu and W. T. Ooi. Zoomable video playback on mobile devices by selective decoding. In *Proceedings of the Pacific Conference on Multimedia*, PCM'12, pages 251–262, 2012.

[18] H. Liu, X. Xie, W.-Y. Ma, and H.-J. Zhang. Automatic browsing of large pictures on mobile devices. In *Proceedings of the 11th ACM International Conference on Multimedia*, MM '03, pages 148–155, Berkeley, CA, USA, 2003. ACM.

[19] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, page 14. California, USA, 1967.

[20] 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 *Proceedings of the 15th European Signal Processing Conference*, EUSIPCO'07, pages 1275–1279, 2007.

[21] A. Mavlankar, D. Varodayan, and B. Girod. Region-of-Interest prediction for interactively streaming regions of high resolution video. In *Proceedings of the International Packet Video Workshop*, PV'07, pages 68–77, Lausanne, Switzerland, Nov. 2007.

[22] M. Rehan and P. Agathoklis. Frame-Accurate video cropping in compressed MPEG domain. In *Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, PacRim'07, pages 573–576, 2007.

[23] A. Santella, M. Agrawala, D. DeCarlo, D. Salesin, and M. Cohen. Gaze-based interaction for semi-automatic photo cropping. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 771–780. ACM, 2006.

[24] A. Shafiei, Q. M. K. Ngo, R. Guntur, M. K. Saini, C. Pang, and W. T. Ooi. Jiku Live: a live zoomable video streaming system. In *Proceedings of the 20th ACM international conference on Multimedia*, MM '12, pages 1265–1266, New York, NY, USA, 2012. ACM.

[25] K. B. Shimoga. Region of interest based video image transcoding for heterogeneous client displays. In *Proceedings of the International Packet Video Workshop*, PV'02, 2002.

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

[27] X. Xie, H. Liu, S. Goumaz, and W.-Y. Ma. Learning user interest for image browsing on small-form-factor devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 671–680. ACM, 2005.