Viewport-adaptive Streaming of 360-degree Video over Networks

Nguyen Van Duc

A DISSERTATION

SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY IN

COMPUTER SCIENCE AND ENGINEERING

Graduate Department of Computer and Information Systems

University of Aizu

2019

The thesis titled

Viewport-adaptive streaming of 360-degree video over networks

by

Nguuyen Van Duc

is reviewed and approved by:

Chief Referee
Professor
Cong-Thang Truong [illegible] Truong

Professor
Anh T. Pham Anh T. PHAM [illegible]

Professor
Incheon Paik [illegible] [illegible]

Professor
Michael Cohen Michael Cohen マイケル コーエン

The University of Aizu
2019

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Virtual Reality (VR) is changing the way we play, learn, and communicate by offering "immersive experiences". Immersive experiences stimulate our human senses, draws us in and take us to another place [1]. With VR, we are able to be anywhere and do anything, with anyone. Especially, watching videos will be brought to a whole new level with spherical video capture and playback. You can watch your favorite football game from the best seat in the stadium while still sitting on your sofa, or can view the game from different viewpoints [3]. VR will take gaming to the next level by making you feel like you are actually a character inside the game. Education will be revolutionized when students are taught with VR-assisted materials and lectures [4]. For example, students can take a field trip to Venice, the Great Pyramid of Giza without ever leaving the classroom [5]. Healthcare, military, manufacturing training will become more cost-effective and safer with VR. For instance, doctors can learn everything from surgical procedures to diagnosing a patient though realistic training [6]. VR can help individuals virtually meet and talk with one another while feeling as if they are physically located in the same place [7]. VR-assisted storytelling can be much more realistic and impactful [8].

Truly immersive VR has extreme requirements on three immersion aspects of 1) visual quality, 2) sound quality, and 3) interactions [1]. The key requirements of VR is summarized in Fig. 1.1. As the user is free to look around in the virtual world, VR needs to provide a full 360-degree spherical view. In other words, 360-degree video content is required. Furthermore, the human eyes have a wide FOV and the display in a VR headset is brought close to the eyes. As a result, an extreme number of pixels are required to eliminate the so-called "screen-door effect". According to [9], truly immersion requires 360-degree videos with a resolution of 24K and near-eye display with a monocular resolution of 8Kx8K. Sound is another crucial

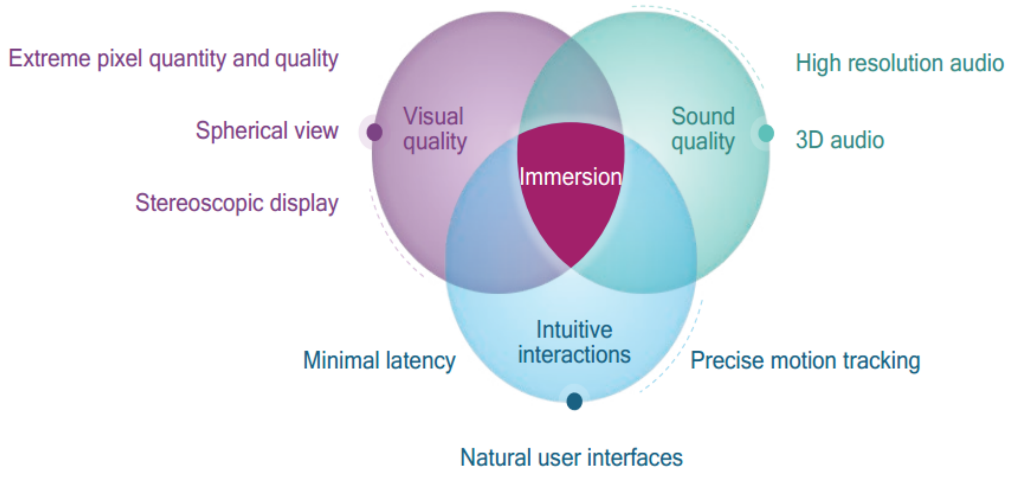


FIGURE 1.1: Requirements for Virtual Reality [1].

component in VR. VR requires 3D audio to provide the sense of presence in virtual world. Currently, 3D sound production in VR is mainly based on Head-related Transfer Function (HRTF) that models the Human Audiotory System [10]. One of the biggest challenges for VR is the amount of time between an input movement and the screen being updated, which is known as “motion to photon” latency. The motion-to-photon latency must be less than 20 ms for a excellent experience in VR [1]. In addition, high accuracy position/gesture/eye tracking are also required to increase the immersive experience in VR.

To satisfy all the extreme requirements of VR, new technologies for the entire VR delivery chain are needed. A summary of existing VR technologies is shown in Fig. 1.2. Existing VR technologies can be classified into four main groups, which are 1) near-eye display, 2) perception and interaction, and 3) rendering processing, and 4) network transmission [9].

The near-eye display technologies focus on improving the visual quality with higher resolution, higher refresh rate, and wider FOV display [11, 12]. The perception and interaction technologies involve position/eye tracking [13], hand interaction [14, 15], and 3D sound [16, 17]. Rendering processing aims to achieve high quality images, low overhead, and low latency. Currently, foveated rendering [18, 19] and multi-view rendering [20] are the main technologies for GPU-based rendering. Also, cloud rendering is emerging as an potential technology to reduce the computation workload on the user devices [21, 22]. The next generation rendering technology will be based on light field technology [23].

To delivery the bulky VR content over networks with low delay, advanced network communication technologies are necessary. Next-generation access networks such as 5G [24],

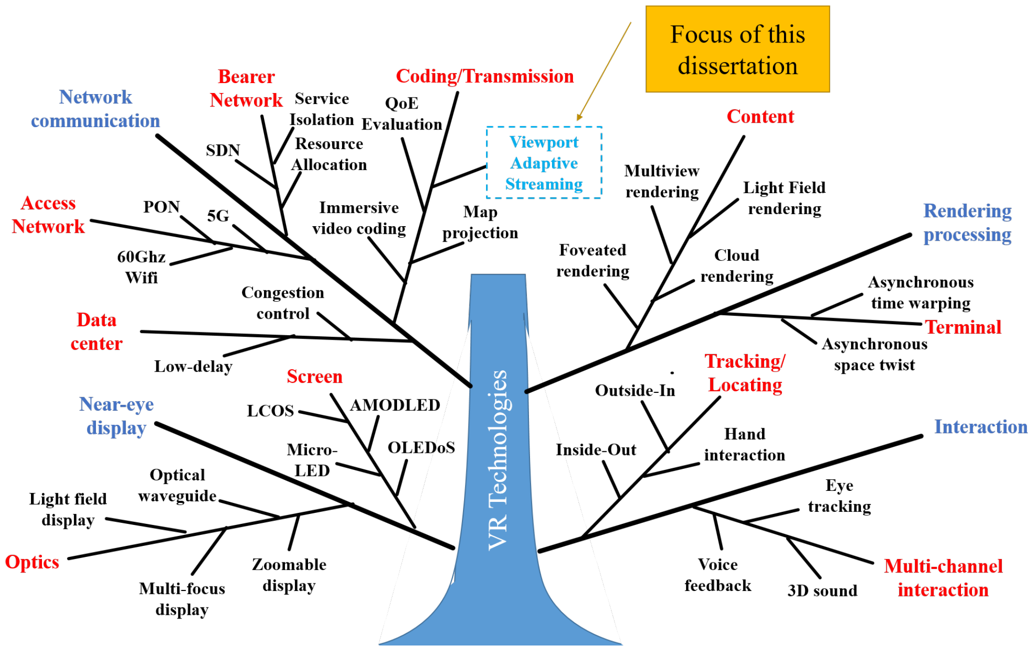


FIGURE 1.2: Key VR technologies.

Passive Optical Network (PON) [25], and new Wifi [26] that can provide high bandwidth/low latency access networks to the users are being developed. Another important delay source in the Virtual Reality delivery chain is caused by congestions at data centers. Thus, effective congestion control methods are required to enable low-latency data centers [27, 28]. Bearer networks also should be re-designed to meet the stringent bandwidth and latency requirements of VR services. Especially, superior VR experience requires customized/agile network that can support resource reservation [29], resource allocation [30], as well as service isolation [31].

Even with high bandwidth/low-latency networks, the bitrate of VR content is still extremely high. Also, the representation of VR content is also very different from the conventional content. Thus, new video coding technologies for VR content (especially 360-degree video) would be essential for storage and transmission of VR content. In terms of standardization, the Joint Video Experts Team (JVET) has just started the standardization activities for Versatile Video Coding (VVC) for 2D/3D Virtual Reality video [32].

In addition, advanced streaming technologies are highly desired to deliver 360 video over networks in an effective manner. State-of-the-art streaming technologies such as HTTP Adaptive Streaming are originally designed to provide interruption-free video playback under varying network conditions [33–36]. For 360 video, the main focus is reducing the bandwidth requirement using viewport-adaptive streaming [37]. Since 360 video is usually viewed on Head-Mounted Displays, existing Quality of Experience(QoE) models such as [38–40] cannot

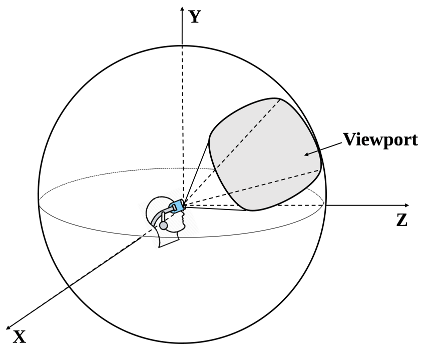


FIGURE 1.3: Viewing 360-degree video using Head-Mounted Display.

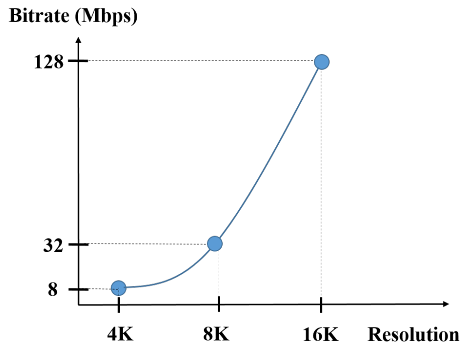


FIGURE 1.4: Expected bitrates at different resolutions with Versatile Video Coding (VVC).

be directly applied to 360 video. Thus, QoE evaluation for 360 video is highly required.

This dissertation focuses on viewport-adaptive streaming, which is the key technology for 360-degree video streaming over networks.

1.1 Motivations

360-degree videos (360 video for short) are video recordings where views in every directions at every time instant are recorded [41]. 360 video is usually viewed on a Head-Mounted Display (HMD) that puts the user at the center of the sphere as illustrated in Fig. 1.3. The user can freely change his/her viewing direction while watching. 360 video streaming can be applied in many applications in entertainment, training, and media. For example, the user can virtually visit some famous places by watching pre-recorded 360 videos. Using 360 video as a training environment, companies can get rid of costly training program. Realistic views provided by 360 video can provide a much better story-telling experience. Due to the limited field of view of human and the headset, the user can only see a portion of the video at a given time. The visible area to the user of the 360 video is called *viewport*.

To provide an excellent immersive experience, 360 video should have extremely high resolution (i.e., 4K or higher) and high frame rate (i.e., 60 fps or higher) [9]. To improve the compression efficiency for very high resolution VR content such as 360 video, the Joint Video Experts Team (JVET) has just started the standardization activity for the next generation video coding called Versatile Video Coding (VVC) [32]. The VVC is expected to provide bitrate reduction of about 50% compared to High Efficiency Video Coding (HEVC) [42]. Nevertheless, VCC will still result in very high bitrates, especially at high resolutions as can

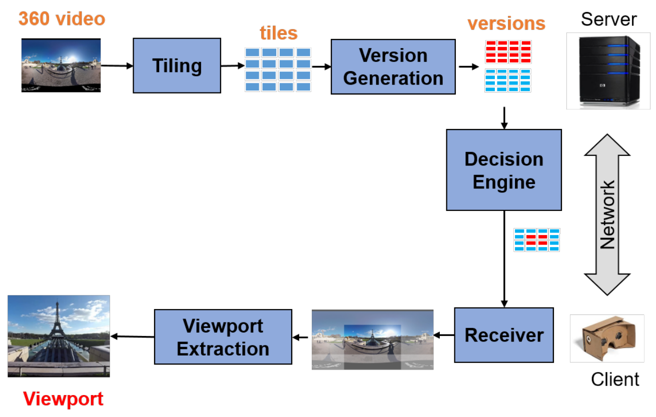


FIGURE 1.5: General architecture of tiling-based viewport-adaptive streaming.

be seen in Fig. 1.4. Such high bitrates are generally much higher than the current network's capacity, especially mobile networks. Thus, effective delivery method for 360 video over networks is necessary for the wide adoption of Virtual Reality applications. It should be noted that, though the emerging 5G networks can provide much higher bandwidth and lower latency, effective use of the available network resources is still very important to 1) user viewing experience and 2) network service providers. Also, the high bandwidth provided by 5G networks can allow the proposed system to support 360 videos with higher resolutions (quality).

To reduce the high bitrate of 360 video, viewport-adaptive streaming has been proposed. The general idea is that the viewport is delivered at a high quality level while the remaining part of the video is delivered at a lower quality level [43]. As only the viewport is visible to the user, viewport-adaptive streaming can significantly reduce the video bitrate while not affecting the user experience.

One of the most popular approaches is tiling-based viewport-adaptive streaming [43–46]. The general architecture of tiling-based viewport-adaptive streaming is shown in Fig. 1.5. In tiling-based viewport-adaptive streaming, a 360 video is spatially divided into small parts called *tiles*, each is encoded into multiple versions of different quality levels. Given the user's viewing direction and the network status, the most appropriate version of each tile is selected and delivered to a client running on the user's device. The client first decodes the tiles' versions, and then reconstructs the 360 video. Finally, the viewport corresponding to the user's current viewing direction is extracted and displayed [44]. Generally, for tile version selection, the *visible tiles* (i.e., tiles overlapping the viewport) are delivered at high quality, while the other tiles at low quality.

However, tile version selection is not a simple task because a lot of challenges are needed to be overcome as follows.

1. In practice, viewport adaptation is sequentially applied to temporal segments called *adaptation intervals* or simply *segments* [47]. Also, the client should buffer some amount of video data before the video can start to play. The segmentation and client buffer cause a delay from when the tiles' versions of a segment are decided until when the playback of the frames in that segment starts. Thus, in order to decide the tiles' versions, future viewport positions need to be estimated. Since the user can freely change his/her viewing direction, errors in viewport estimation are likely to occur.

   To deal with viewport estimation errors, the tiles surrounding the estimated viewport should also be delivered at high quality. Those tiles form the so-called *extension area* [37]. The remaining tiles form the so-called *background area* [37]. As the likelihood that the user will see the tiles in the background area is low, these tiles are usually delivered at the lowest quality, just in case there is sudden changes in head movements. For tile version selection, the two key questions are 1) *How big should the extension area be?* and 2) *What are the versions of the viewport and extension areas?*

   In addition, most previous studies (e.g., [48–50]) employ HTTP Adaptive Streaming (HAS) systems [51] for viewport-adaptive streaming. However, HAS results in a high delay due to segmentation and client buffering [45] that causes viewport prediction errors and so low viewport quality. This factor actually has never been investigated in the existing literature.

2. For client-based adaptation, the existing HAS standards provide little information to optimize the adaptation of 360 videos. For example in MPEG DASH standard, the quality of a version is provided as a rank/order, not an actual quality value. Also, as discussed in our previous work [37], the current HAS standards define the bitrate of a version as the maximum bitrate of all segments (intervals) in that version. That is the reason why instant bitrate estimation is shown to be useful to improve adaptation performance in [50].

   As for evaluation, most of the previous studies use bandwidth saving as the key performance metric [49]. However, the bandwidth saving is not able to reflect the quality seen by users. In [44], average viewport PSNR is used to demonstrate the effectiveness of some viewport-adaptive strategies compared to a non-viewport-adaptive strategy. However, it is not possible to see how the video quality changes throughout a streaming session with the average PSNR. Note that the system delay is not described in these studies.

3. In the current literature, typical tiling schemes include 4x3 [45], 6x4 [44], 8x4 [52], and 8x8 [37]. Some studies investigate good tiling schemes from the server's point of view, by considering the tradeoff between coding efficiency and the number of tiles [44, 53]. However, no metric to decide the optimal tiling scheme has been considered.

   In our opinion, the existing studies on optimal tiling scheme miss two important issues. First, the optimal tiling scheme should be mainly considered from the client's point of view. That is, it should be based on the quality performance measured at the client, not at the server. Second, a tiling scheme so far is fixed during a whole streaming session. Intuitively, when the head-moving speed is small, one should use high tiling granularity (i.e., large number of tiles) as it can reduce the amount of redundant pixels, which are the pixels belonging to high quality tiles but not in the viewport. Meanwhile, redundant pixels in case of low tiling granularity (i.e., small number of tiles) can help cope with a high head-moving speed. Since the user head movement is generally varying throughout a streaming session, using a fixed tiling scheme as in existing studies might lead to non-optimal viewport quality.

   In this context, it is important to answer some related questions such as "what is the benefit of adaptive tiling compared to fixed tiling?", "which tiling should be selected given a speed of head movement?", or "if fixed tiling is preferred in a given context, what is the best tiling scheme for the client?".

1.2 Limitations

In this part, the scope of this thesis will be presented. First, the methods/frameworks proposed in this thesis are limited to monoscopic (2D) 360-degree videos. Although stereoscopic (3D) 360-degree videos are generally required to provide truly immersive experience to the users, it will make the system become very complex. The task of extending the proposed

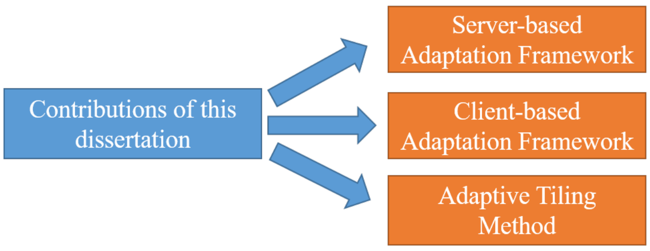

Server-based Adaptation Framework
Contributions of this dissertation
Client-based Adaptation Framework
Adaptive Tiling Method

FIGURE 1.6: Three main contributions of this dissertation.

methods/frameworks to support 3D 360 videos is reserved for our future work. As mentioned above, immsersive audio is another key component in Virtual Reality. However, since our focus is on delivery of 360-degree video, the generation, delivery, and playback of 3D audio are not considered in this thesis.

Second, the proposed methods/frameworks presented in this thesis are specific to tiling-based viewport-adaptive streaming. Although there exist other viewport-adaptive streaming approaches such as viewport-dependent approach [54], tiling-based approach is the most popular approach used in the literature. Moreover, according to the results from a recent study [55], viewport-dependent approach results in in much lower viewport quality than that of the tiling-based approach.

Third, this thesis focuses on improving the viewing experience for single-user scenario in which an user watches a 360 video streamed from a server using a Head-Mounted Display. Multiple-user scenario will not be considered in this thesis. We also assume that the user head movements contain only rotations in pitch and yaw dimensions. The rotations in the roll dimension are not considered.

1.3 Main contributions

In this dissertation, the above challenges in tiling-based viewport-adaptive streaming of 360 video over networks will be addressed. The main contributions of this dissertation are summarized in Fig. 1.6. The details of each contribution is described as follows.

- First, a new server-based adaptation approach for tiling-based viewport-adaptive streaming that can systematically decide the version of each tile according to user head movements and network conditions is proposed. Compared to existing approaches, the proposed approach can improve the average viewport quality by up to 3.8 dB and reduce the standard deviation of the viewport quality by up to 1.1 dB. This contribution has been published in [37,56] and will be presented in Chapter 3.

- Second, a client-based framework for 360 video streaming is proposed. The proposed framework addresses all the key issues in client-based viewport-adaptive streaming. Experiment results show that the use of estimated bitrate/quality can help improve the viewport quality significantly. The improvements were seen in all three tile selection methods of the framework. Especially, the performance with estimated bitrate/quality is nearly the same as the performance with full information of bitrate/quality. This means the complexity of sending full bitrate/quality information of tiles' versions could be avoided. This contribution has been published in [57] and will be presented in Chapter 4.

- Third, a novel solution for adaptive tiling selection is presented. The general problem for adaptive tiling in 360 video streaming is first formulated. Then, correspondingly a simple solution to that problem is devised. Experiment results show that adaptive tiling can improve the average viewport quality by up to 2.3 dB compared to a fixed tiling solution. It is also found that among fixed tiling schemes, 4x3 tiling achieves the lowest viewport quality and thus should not be used. This contribution has been published in [58] and will be presented in Chapter 5.

Chapter 2

Background

In this chapter, the basic concepts of 360-degree (360 video for short) video delivery over networks will be presented. Note that, the term “360 video” here refers to “monoscopic” 360 videos. The more complex stereoscopic 360 video is not considered in this dissertation.

360 videos, also known as spherical videos, are video recordings where views in every directions at every time instant are recorded. Conventional videos, on the other hand, contain only a single view at a time instant. 360 video allows viewers to freely change their viewing direction while watching, creating the sense of presence, which is essential to provide the immersive experiences to users in VR services.

Field of view (FOV) is the extent of the observable environment at any given time. In Virtual Reality, FOV is dependent on both human eyes and the Head-Mounted Display. Human eyes have a horizontal monocular field of view between 170-175 degrees. The combination of the two monocular FOVs of two eyes is called binocular FOV, which is between 200-220 degrees. Each eye has a vertical FOV of approximately 135 degrees. Existing Head-Mounted Displays on the market usually have FOVs between 90-110 degrees, which are generally smaller than human FOV.

A system overview for 360 video delivery chain is shown in Fig. 2.1. The delivery chain starts with the acquisition of 360 video. Then, projection mapping will be performed to generate a 2D representation of the original 360 video. The encoding process of the video should produce an encoded bitstream that allows access to the portions of the encoded video corresponding to the current viewport. Next, the video data is segmented and encapsulated for transmission over the network. The process of segmentation/encapsulation is dependent on which streaming protocol is used. At the receiver side, the video segments are decapsulated and fed to the video decoding block. The decoder passes the reconstructed video data to the

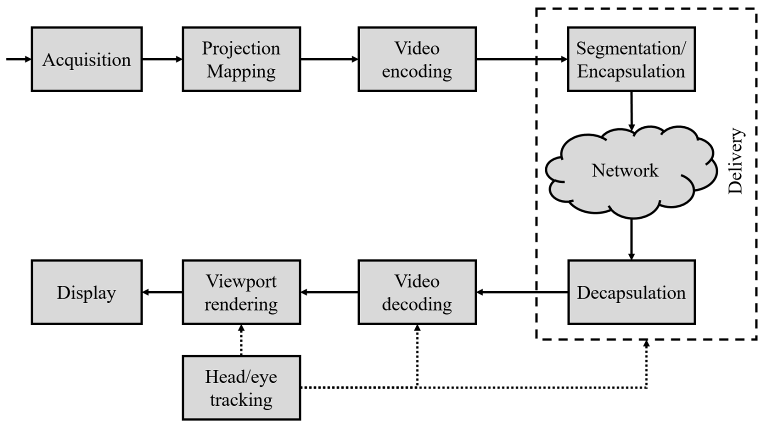
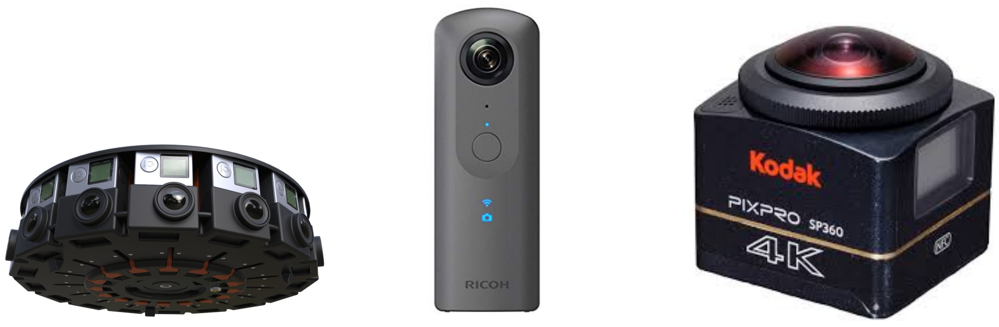


FIGURE 2.1: 360 video delivery chain.



(A) Camera rigs (B) Dual-lens cameras. (C) Single len cameras.

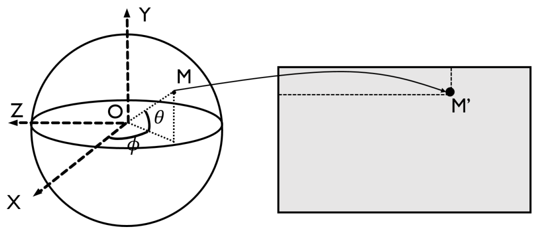
FIGURE 2.2: Three typical 360 video capturing devices.

viewport rendering block which generates viewports based on the current viewing direction and the user device's capacities.

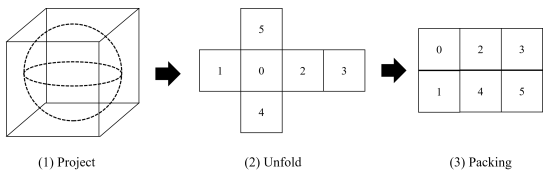
In the following sections, the key building blocks of content acquisition, projection mapping, video coding, video delivery, and rendering are described in detail.

2.1 Content Acquisition

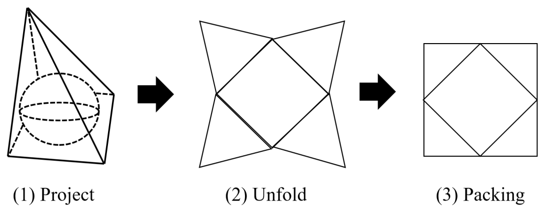
360 video is typically recorded using either special camera rigs or dedicated 360 cameras as shown in Fig. 2.2. Camera rigs such as GoPro's Odyssey [59] consist of multiple cameras installed within a frame. Popular handheld 360 cameras such as Ricoh Theta [60] and Samsung Gear 360 [61] use dual-lens in opposite directions (i.e., front and back) to capture 360 video. There have also been single spherical len cameras such as Kodak Pixpro SP360 4K [62]. Existing consumer-level 360 cameras typically produce 360 videos at resolutions of 2K-4K and frame rates of 25-30 fps.



(A) Equirectangular projection (ERP)



(B) Cubemap projection (CMP)



(C) Pyramid projection.

FIGURE 2.3: Three typical projection mapping.

To generate a 360 video, videos from multiple cameras (or lens) are needed to be stitched. Video stitching process generally involves generation of reference projection, determination of projection matrix, and generation of blending map [63]. To generate the reference projection, one camera (len) is taken as reference. The relative rotations and translations of the remaining cameras (lens) are then estimated. The next step is to estimate the projection matrix that maps every pixel of the input videos to a projection surface, which is a sphere in case of 360 video. Finally, a blending function is estimated and applied to each video to handle the exposure difference in the overlapped areas between cameras (lens). Existing video stitching methods can be divided into two main categories which are 1) deformation-based [64] and seam-based [65]. As stitching process is performed at pixel level, several fast stitching techniques have been proposed for real-time scenarios such as using saliency map [63] or GPU-assisted [66].

2.2 Projection Mapping

To preserve a lot of performance improvements in modern rectangular block-based coding architecture (e.g., Advanced Video Coding (AVC) [67], High Efficiency Video Coding (HEVC) [68]), a 360 video should be mapped to a rectangular video by means of projection mapping before encoding. Various projection surfaces can be used for mapping a sphere such as cylindrical [69], cubic [70], or pyramid [71]. In addition, given a projection surface, there exists multiple ways to map the sphere on to the chosen surface. Fig. 2.3 shows three common projection formats of equirectangular projection (ERP), cubemap projection (CMP), and pyramid projection. There are two main types of projection formats which are 1) viewport-independent format and 2) viewport-dependent format.

The ERP is the most simple viewport-independent projection format which uses a constant spacing of longitudes. In the ERP, each latitude is stretched to the width of the rectangular. As a result, this projection causes a lot of redundant pixels at the two poles that in turns reduces the compression efficiency. Another drawback of ERP is that straight line motions are curved, making the projected video become harder to encode.

The CMP is another popular viewport-independent format which places the sphere at the center of a cube [55]. Each face of the cube is generated by a rectilinear projection with a 90-degree field of view. The six faces of the cube are then rearranged into a rectangular as shown in Fig. 2.3b. The CMP projection can eliminate the severe stretching at the two poles in the ERP projection. As a result, the CMP projection have lower number of redundant pixels compared to ERP. Yet, this projection still results in a sampling density that varies over each face of the cube. Several CMP-variant projections have also been proposed such as Equi-Angular Cubemap (EAC) [72], Hybrid Equi-Angular Cubemap (HEC) [2]. Those projections adjust the mapping functions to make the distribution of the sampling points on the sphere more uniform for compression enhancement.

Different from viewport-independent formats such as ERP and CMP, viewport-dependent projection formats like the pyramid projection are composed of two areas: primary area and secondary area. The primary area is the visible area corresponding to a pre-defined viewport and has high quality. The secondary area is the region outside the primary area and has lower quality. In case of the pyramid projection (i.e., Fig. 2.3c), the base of the pyramid is the primary area while the sides of the pyramid composed of the secondary area. Other viewport-depedent formats are truncated square pyramid projection [73] and compact-based cube projection [54]. A key problem of viewport-indepedent formats is that a lot of versions corresponding to a set of pre-defined viewports must be generated in advanced to support viewport adaptation, which requires significant storage requirement.

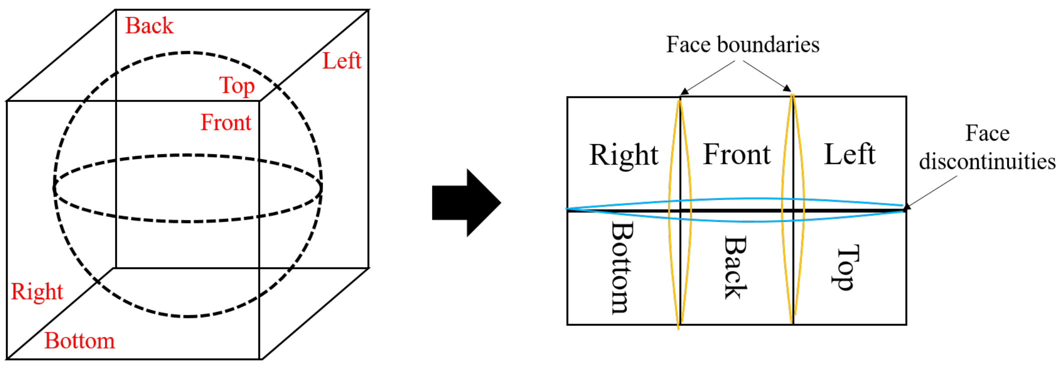
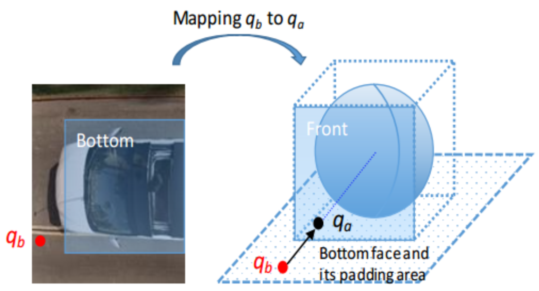
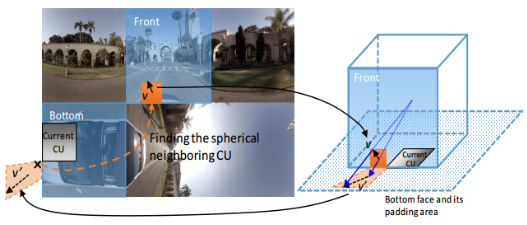


FIGURE 2.4: Face boundaries and discontinuity in cube map projection with 3x2 frame packing.



(A) Geometric Padding


Front
Bottom
Current CU
Finding the spherical neighboring CU
Front
Current CU
Bottom face and its padding area

(B) Motion Estimation

FIGURE 2.5: Geometry padding and motion estimation in 360 video coding [2].

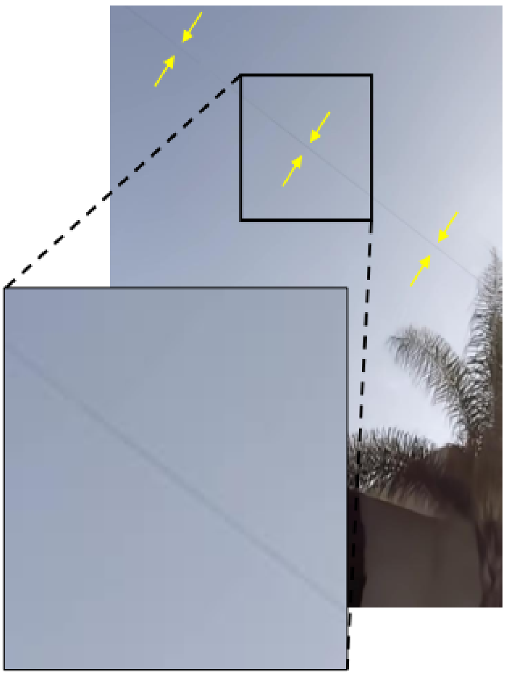


FIGURE 2.6: Seam artifacts in a rendered viewport [2].

2.3 Video Encoding

Spherical continuity is one of the special properties of 360 video. Yet, projection mapping from spherical domain to plane domain causes face discontinuities and face boundaries in the projected video as can be seen in Fig. 2.4. Thus, conventional coding tools were modified to consider the correct spherical neighborhood.

For inter prediction, the repetitive padding is replaced with geometry padding when the reference pixels are located outside the frame boundary or across the discontinuous edges [2, 74]. For example, as shown in Fig. 2.5a, a pixel q_b outside the bottom face is padded by its corresponding pixel on the sphere q_a locating on the front face. Conventionally, the search area to find the motion vector of a coding unit (CU) is centered around the location of the CU [67]. However, when a coding unit is located beside the frame boundary or the discontinuous edge, the search area should consider the spherical neighboring CUs [2]. An example is shown in Fig. 2.5b where the current CU in the bottom face has a spherical neighboring CU located in the front faces.

In conventional coding systems, in-loop deblocking filters are applied to decoded video frames to mitigate the block artifacts [75]. Yet, applying conventional filters to 360 video may result in the seam artifact that can significantly reduce the perceptual quality. Fig. 2.6 shows an example of the seam artifact appeared in a rendered viewport. In [2], four advanced filters, namely de-blocking filter, sample adaptive offset (SAO) filter, adaptive loop filter (ALF), non local mean filter (ALF), and convolution network (CNN) filter, are proposed and found to be effective in eliminating the seam artifact. In [74], a face discontinuity handling (FDH) is proposed in which information from a frame packed neighbor block is used to code the current block only when the frame packed neighbor block is also a spherical neighbor block of the current block.

2.4 360-degree Video Delivery

Though advanced coding technologies can significantly reduce the bitrate of 360 video, delivery 360 video over networks is still a challenging task due to limitations in network resources, as well as constraints imposed by end-user devices. Therefore, cost-effective delivery technology is necessary for the wide adoption of VR/AR applications.

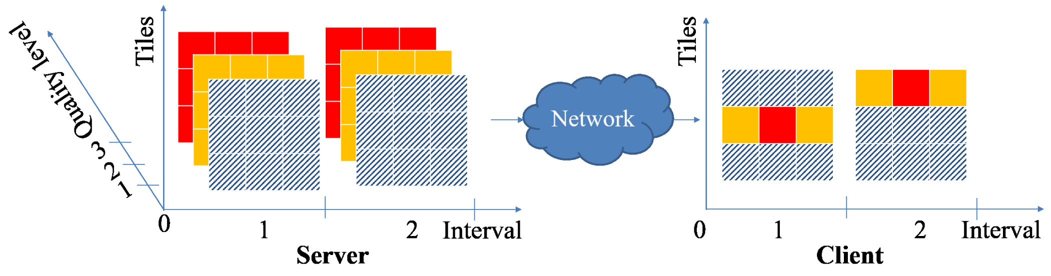


FIGURE 2.7: Tiling-based viewport-adaptive streaming approach.

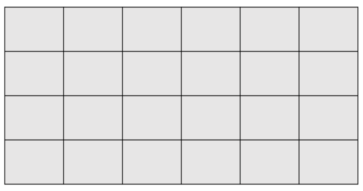
For effective streaming of 360 video over networks, viewport-adaptive streaming has been introduced. The basic idea is to deliver a *viewport*, which is the visible part according to current the user's viewing direction, at high quality, whereas the other parts are delivered at lower quality [37]. There are two main approaches for Viewport-Adaptive Streaming, which are 1) tiling-based approach and 2) viewport-dependent approach.

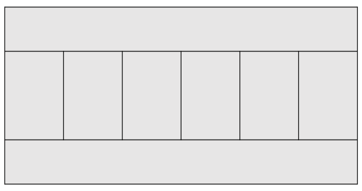
2.4.1 Tiling-based Viewport-adaptive Streaming

Tiling-based is the most popular approach to enable viewport-adaptive streaming where a 360 video is spatially divided into small parts called *tiles* [37,44,52]. The resulting tiles are independently encoded at multiple quality levels (or versions). Given a user viewport position and network conditions, the system decides the most appropriate version of each tile and delivers the selected tiles to the client. Generally, high quality versions will be selected for the visible tiles (i.e., tiles overlapping the viewport), low quality versions for the other tiles. At the client, the tiles are firstly decoded, then combined to reconstruct the 360 video. Finally, the viewport is extracted and shown to the user.

Tiling Scheme

In most previous studies, 360 video is divided into equal-sized, non-overlapping tiles by a grid. Various tiling schemes have been proposed such as 6x4 (ERP) [44], 8x8 (ERP) [76], 12x6 (ERP) [77], 2x2 (CMP) [53, 78], and 4x4 (CMP) [78]. Some variable-sized tiling schemes have also been proposed, utilizing the user viewing behaviors [43,79–81], the content characteristics [79,82], or the visual attention model [83]. The tiles can also be overlapping as proposed in [84]. The tile scheme can be the same for all video segments [44] or variable per video segment [79].


(A) Non-overlapping, equal-sized tiling scheme


(B) Non-overlapping, variable-sized tiling scheme

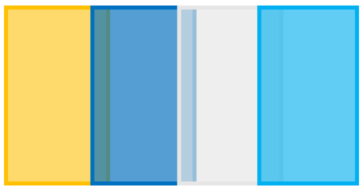

(C) Overlapping, equal-size tiling scheme.

FIGURE 2.8: Three typical tiling schemes for the ERP projection format.

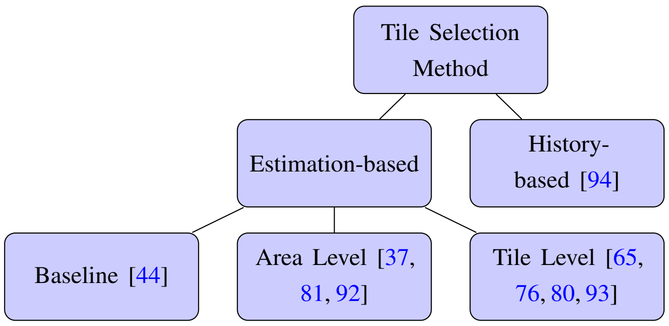
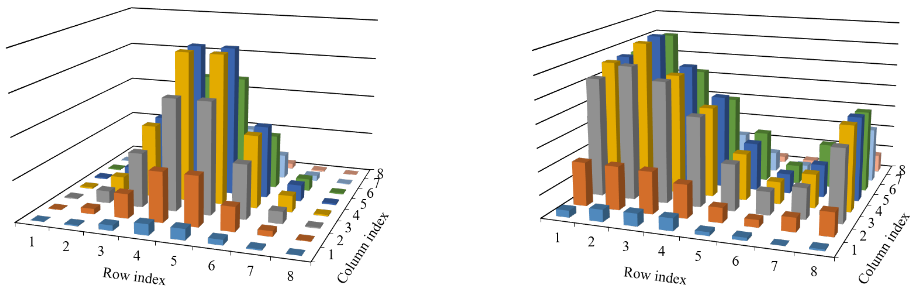


FIGURE 2.9: A classification of existing tile selection methods.

Tile Version Selection Methods

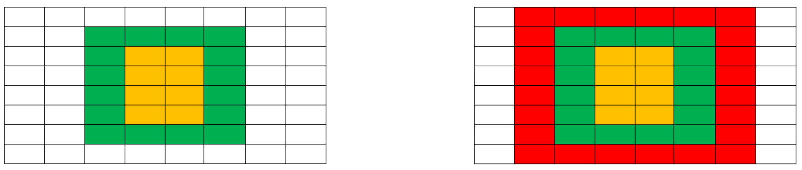
In the literature, adaptation methods for conventional video have been extensively studied [85–90]. However, the main goal of these adaptation methods is to provide smooth video playback under varying network conditions. In contrast, viewport-adaptive streaming aims to reduce the bandwidth requirement by delivering visible tiles at high bitrate.

If future viewport positions can be estimated accurately, a simple method that selects the highest possible versions for the visible tiles should be sufficient. However, it is found that errors in viewport position estimation are not trivial, especially for long-term estimation [49, 91]. Thus, in addition to the visible tiles, some invisible tiles should also be delivered to deal with viewport estimation errors. In practice, it is required that all invisible tiles must be delivered at least at the lowest version. Invisible tiles surrounding the viewport are called *extension tiles*. The existing methods differ in 1) the selection of the extension tiles and 2) the versions assigned to the visible and extension tiles.



(A) *Estimation-based*: Tiles' versions gradually decrease from the estimated viewport position. (B) *History-based*: Tiles' versions are dependent on past users' viewing preferences.

FIGURE 2.10: Examples of tiles' versions of the *Estimation-based* and *History-based* methods.



(A) Invisible tiles are divided into 2 groups [81,92]. (B) Invisible tiles are divided into N groups [37].

FIGURE 2.11: Different classification schemes for the invisible tiles. The visible tiles are marked by yellow color.

The simplest tile selection method is to select the highest possible version for the visible tiles and a low version for all invisible tiles. This method is denoted as *Baseline*. The *Baseline* method is simple but may result in fluctuating viewport quality, especially when viewport estimation errors are high [37]. A number of tile selection methods have been proposed to address the problem of the *Baseline* method such as [37,65,76,80,81,92–94]. A classification of existing tile selection methods is shown in Fig. 2.9. Specifically, existing methods can be classified into two groups, namely *Estimation-based* and *History-based*. In *Estimation-based* methods, tiles closer to the estimated viewport position are assigned higher versions. As a result, the tiles' versions will be gradually decreasing from the estimated viewport position as illustrated in Fig. 2.10a. Note that, the *Baseline* method also belongs to the *Estimation-based* group. In *History-based* methods, the version of a tile is decided based on how that tile has been viewed by past users. If a tile has been watched by the majority of the users, that tile will be assigned a high version. In contrast, those tiles that have been rarely watched will have lower versions. Fig. 2.10b shows an example of the selected versions of tiles of a *history-based* method.

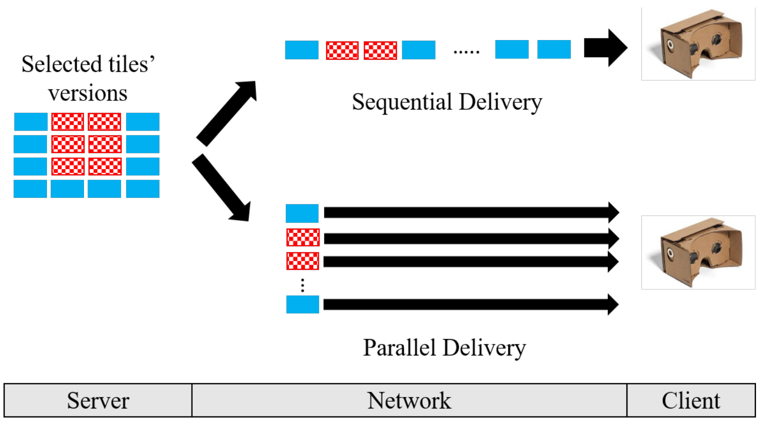
Estimation-based methods first estimate the viewport position when the playback of the segment starts. Then, it assigns higher quality versions for tiles that are closer to the estimated viewport position. The selection of tiles' versions can be done at either area level [37, 81, 92] or tile level [65, 76, 80, 93].

In [37, 81, 92], the invisible tiles are divided into several areas or groups. Tiles in the same group have the same version. In [81, 92], two groups, namely extension group and background group are considered. Note that, the extension and background groups are respectively called adjacent group and outside group in [81]. In the method of [81] and the *EXT-1* method of [92], the extension group consists of the invisible tiles that are adjacent to the visible tiles (called the adjacent tiles). Similar to the *EXT-1* method, the extension group in the *EXT-2* method of [92] includes the adjacent tiles and the invisible tiles that are adjacent to the adjacent tiles. In [37], the invisible tiles are divided into N groups, depending on the total number of tiles. The classification schemes for the invisible tiles are illustrated in Fig. 2.11.

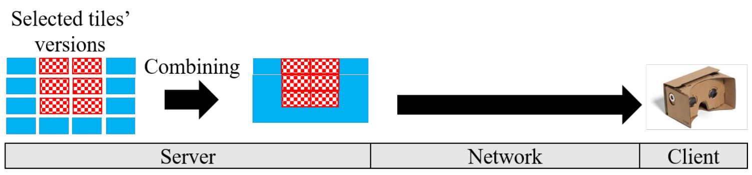
For tile version selection, the *EXT-1* and *EXT-2* methods select the same version for the visible and extension tiles. In contrast, the method of [81] selects the highest possible version for the extension tiles *after* choosing the highest possible version for the visible tiles. As a result, the extension tiles usually have much lower version than that of the visible tiles. In [37], the tiles' versions are selected so as to minimize a weighted sum of versions' distortions. The weight of a tile is calculated as the ratio of the overlapped area of that tile and the estimated viewport. In addition, the last viewport estimation error is also considered in the calculation of the expected viewport distortion.

The methods of [65, 76, 80, 93] decide the versions of individual tiles. For that purpose, each tile is assigned a weight. Basically, the higher the weight of a tile is, the higher the version of that tile would become. These methods are different in the way the weights of tiles are calculated. In [80], the weight of a tile is calculated as a reciprocal function of the distance between the tile's center point and that of the estimated viewport position. Meanwhile, Gaussian functions are employed in [65, 76, 93]. The tiles' weights thus conform a bell shape centered at the estimated viewport position. The difference between the methods of [65, 76, 93] is the value of the standard deviation, which controls the width of the "bell". In [65, 76], the standard deviation is calculated as $s \times d$ where s is the user's head movement speed and d is the network delay [76] or the segment duration [65]. In [93], the standard deviation of past estimation errors is used.

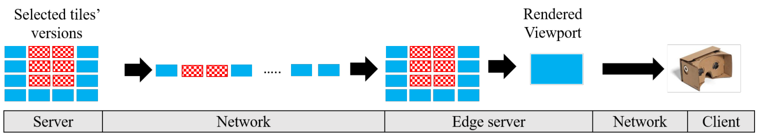
In history-based methods, the weights of tiles are determined based on the viewing



(A) Tile versions are delivered seprately to the client.



(B) Tile versions are combined at the server first, then delivered to the client.



(C) Tile versions are delivered separately to an edge server, which renders and delivers the viewport to the client.

FIGURE 2.12: Three typical tile delivery options.

preferences of past users watching the same video. Specifically, the tiles that have been watched by the majority of the users will have higher weights than those of rarely watched tiles. In [94], the weight of a tile of a user is calculated as the fraction of the surface area of that tile occupied by the viewport as seen by that user. Then, the weights of individual users are aggregated to obtain the final weight of that tile. The tiles' versions are then chosen so as to minimize a weighted sum of versions' distortions.

Existing tile selection methods use Viewport PSNR (V-PSNR) as the performance metrics [37]. To provide satisfactory user experience, QoE-related factors should be taken into account [95–97].

Tile Delivery Options

There are several options to deliver the selected tiles' versions from the server to the client. Fig. 2.12 shows three typical tile delivery options. The most popular option is to deliver the

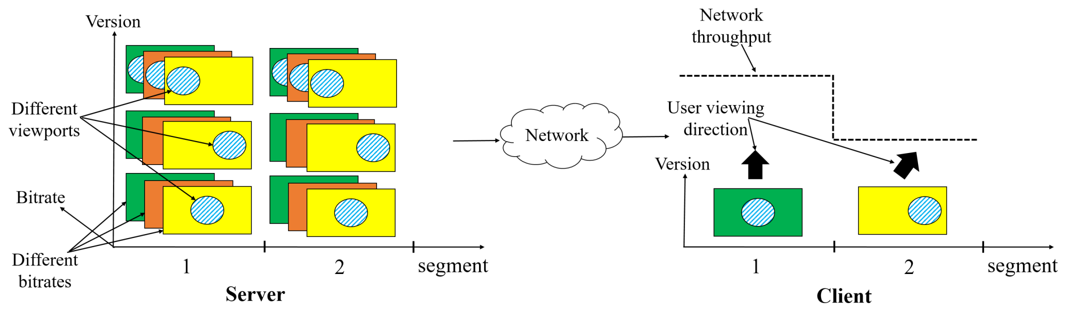


FIGURE 2.13: Viewport-dependent streaming approach.

selected tiles' versions separately to the client [44, 46, 77, 81, 98–100]. Furthermore, the tiles versions can be delivered in either sequential manner [44, 77, 81] or parallel manner [98]. Traditionally, each of the tiles' versions is delivered using one HTTP request-response transaction [44], which may results in high request overhead and low bandwidth utilization. Various features of HTTP/2 protocol [101] have been utilized to address the problems of HTTP-based delivery [102–104]. Specifically, HTTP/2's *server push* feature, which allows the server to push multiple responses given a single client's request, has been used to reduce the request overhead [81, 105]. HTTP/2's *stream priority* feature has been utilized to specify the delivery order of the tiles' versions, so that more important tiles will be delivered first [99, 100]. Moreover, delivery of tiles that will likely miss their playout deadline can be canceled on-the-fly using HTTP/2's *stream termination* feature [98].

In the second delivery option (see Fig. 2.12b), the tiles' versions are combined into a single video first. The combined video is then delivered to the client [37]. Since the combination of the tiles' versions into a single file is a resource-intensive task, this option should be used in server-based approach.

The third delivery option is illustrated in Fig. 2.12c. In the third option, an edge server located between the client and the server is responsible for receiving the tile' versions, rendering the viewport, and delivering the viewport to the client [21, 106, 107]. This option can help reduce the bandwidth usage and computational workload on HMD devices, especially for those with limited computational resources.

2.4.2 Viewport-dependent Streaming

This approach makes use of the viewport-dependent projection formats such as pyramid projection [71] and truncated square pyramid projection [73]. Specifically, multiple versions of a 360 video is generated, each corresponds to a pre-defined viewport. For each video version, the primary area (i.e., the visible area corresponding to the associated viewport) will have higher quality than the remaining area. Each version is further encoded into multiple representations of different bitrates. Given the user's viewing direction and the network conditions, the most suitable representation will be selected and delivered to the client. At the client, the representation is decoded and the viewport is extracted and displayed on the user device's screen [54]. Fig. 2.13 shows an example of viewport-dependent approach where three versions corresponding to three different viewports are available. Typically, the version whose the associated viewport is the closest to the user current viewing direction will be selected [54]. In previous studies, various version numbers have been used such as six [54] or thirty [71].

It can be noted that the key difference between this approach and the tiling-based approach is the way the video is prepared on the server. In the viewport-dependent approach, the system can only support a pre-defined number of viewports. The tiling-based approach can support every viewport. Thus, the tiling-based approach has higher scalability than the viewport-dependent approach. Also, viewport-dependent approach does not have the flexibility to deal with viewport estimation errors. The main advantage of this approach is that it requires minimal changes to the existing system. For example, MPEG-DASH based systems can be used by adding viewport-related information of each version to the metadata. Yet, this approach usually requires a large number of versions to cover all possible viewports, which can effectively increase the storage cost [108].

2.5 360-degree Video Rendering

To provide real-time VR experience to the user, the rendering process must produce high image quality with low delay. Even with advances in graphic hardware, computational resources required for real-time rendering are growing faster than ever. Especially, new Head-Mounted Displays for Virtual Reality have very high display resolutions and target refresh rate. Also, effective rendering methods are especially necessary to support low-power devices such as small phones and tablets.

Foveated rendering is a popular method to reduce the computational resources [18]. This method exploits the fact that human visual acuity decreases significantly from the retina center (i.e., the fovea) to the eye's periphery. Foveated rendering decreases rendering quality toward the periphery while maintaining high quality in the fovea [109]. It has been found that temporal stability and contrast preservation are key requirements for foveated renderers [18].

Another method to increase rendering speed is mipmapping [110]. A mipmap is a sequence of textures, each of which is a progressively lower resolution representation of the same image. Mipmapping can improve the rendering quality by using high-resolution mipmap for objects that are close to the viewer. Lower resolution are used as the objects appear farther away. A drawback of this method is that the mipmaps need to be prepared in advance, which increases memory usage.

To enable real-time stereoscopic rendering, multi-view rendering technology has been proposed [20]. The basic idea of multi-view rendering is that the similar information of images for left and right eyes is reused. To provide high quality rendering on lightweight devices, the rendering task can be offloaded to powerful cloud servers. The rendered images are then sent to the user devices. This technology is called cloud-based rendering [21].

In addition to Head-Mounted Display, another popular approach for viewing 360 video is projecting 360 video on full-dome screens. The full-dome screens can be very effective in a wide range of applications such as astronomical simulations [111] and earth science simulations [112].

2.6 Quality Metric

Conventionally, Peak Signal to Noise Ratio (PSNR) is the most popular objective metric to measure the video quality. PSNR measures the mean squared error between the compressed video and the original one. Firstly, the mean squared error (MSE) between the original and compressed videos is calculated by

$$MSE = \frac{1}{M \times N} \sum_{i=1}^{M} \sum_{j=1}^{N} (A(i,j) - A_0(i,j))^2. \tag{2.1}$$

Here, M and N are respectively the width and height of the video in pixels. A and A_0 respectively denotes the compressed video and original video. $A(i,j)$ is the value of pixel (i,j). The PSNR is computed from the MSE as follows.

$$PSNR = 10\log_{10}(\frac{255 \times 255}{MSE}) \tag{2.2}$$

It should be noted that the higher the PSNR is, the higher the video quality becomes. For good video quality, PSNR should be equal to or higher than 35 dB [96]. Because the user can only see the viewport when watching 360 video, we are interested in evaluating the quality of the viewport. Specifically, in this thesis, the performance of the proposed methods are evaluated using viewport PSNR (V-PSNR) metric, which is the PSNR value calculated on the viewport pixels.

Chapter 3

Server-based Adaptation Framework

3.1 Introduction

There have been some previous studies proposing methods for tile version selection. A simple method, which selects the highest possible version for the visible tiles and the lowest version for other tiles, is used in previous work [43–46]. The tile versions may be different in resolution [43, 45, 46] or Quantization Parameters [44]. However, this method may suffer significant quality degradations due to viewport estimation errors, which is unavoidable due to the randomness of the user head movements [49, 113]. To deal with viewport estimation errors, some previous studies propose to further deliver the tiles surrounding the viewport area at high quality [49, 80, 81]. However, no specific algorithm for selecting tiles' versions is given in [49]. Though two tile selection algorithms are presented in [80, 81], they use constant values for some important parameters such as the extension width [81] and the portion of bandwidth allocated for the visible tiles [80]. Such constant values make it difficult to apply these algorithms in different scenarios.

In addition, most of previous studies employ HTTP Adaptive Streaming (HAS) for video data delivery [43, 44, 49, 80]. Yet, HAS results in a high delay due to its long segment duration and large client buffer [114]. So far, the impact of the delay on the performances of tile selection methods has never been investigated [43, 44, 49, 80]. However, a majority of 360 video streaming applications require very low latency. For example, telepresence applications require an end-to-end delay lower than 5 ms [115]. In such a case, server driven adaptation is a better choice as the client-based approaches would result in unacceptable high latency.

Furthermore, most of the previous studies use bandwidth saving as the key performance metric [43, 44, 49]. However, the bandwidth saving cannot accurately reflect the quality perceived by the user. In [44], the average viewport PSNR is additionally used to demonstrate the effectiveness of a viewport-adaptive approach compared to conventional approaches. However, it is not possible to see how the video quality changes throughout a streaming session with the average viewport PSNR.

In this chapter, a new adaptation approach for tiling-based viewport-adaptive streaming that can systematically decide the version of each tile according to user head movements and network conditions is proposed. The proposed framework has the following key features:

- The tile selection problem is formulated as an optimization problem with a new quality objective that is based on the visible portion of each tile.
- Viewport estimation errors and user head movements in each adaptation interval are jointly considered in the optimization problem.
- Two solution options for the problem are devised.
- The experiments are carried out with an actual test-bed.
- The behaviors of the proposed and reference methods are analyzed under different user's head movement patterns.
- The impact of the segment duration on the performances of tile version selection methods is investigated. It is found that the proposed method outperforms the reference methods when the delay increases.

Experimental results show that the proposed approach can effectively adapt to the user head movements and varying bandwidth conditions. The proposed approach can improve the average viewport quality by up to 3.8 dB and reduce the standard deviation of the viewport quality by up to 1.1 dB compared to reference approaches. In addition, it is found that long segment durations and large buffer sizes cause severe impacts on the performances of tile selection methods.

The remainder of this chapter is structured as follows. The related work is presented in Section 3.2. The proposed framework is given in Section 3.3. The proposed approach is described in Section 3.4. The evaluations and discussions are drawn in Section 3.5. Finally, the chapter is summarized in Section 3.6.

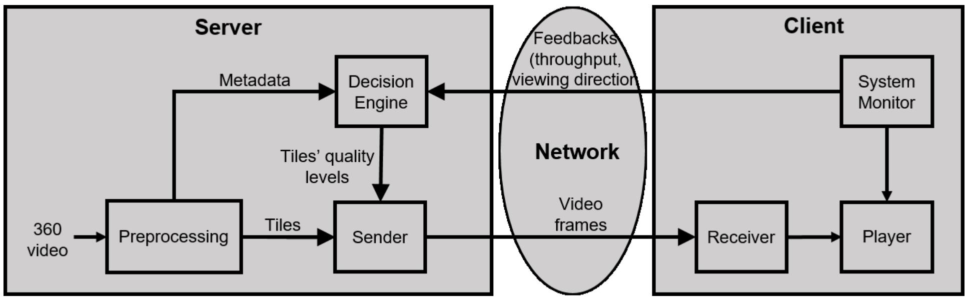


FIGURE 3.1: System architecture.

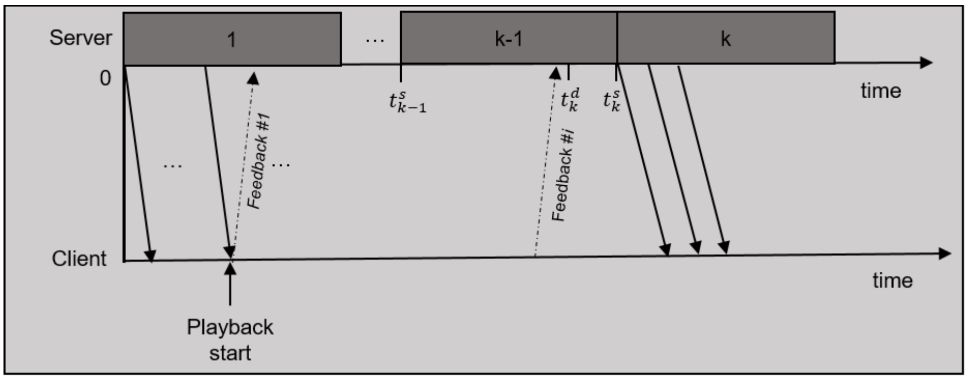


FIGURE 3.2: System data flows.

3.2 Related Work

In 360 video streaming, a 360 video is first projected onto a plane. In tile-based streaming, this projected video is then divided into tiles which are further encoded into multiple versions. In order to choose the optimal version of each tile, future viewport positions are estimated. Then, the selected tiles are transmitted to the client. Finally, some metrics are used to evaluate the performance of 360 video streaming. In this section, the related work to 360 video streaming is presented in detail.

Currently, typical planar formats for 360 videos include Cubemap (CMP), Equirectangular (ERP), Equal-Area (EAP), and Pyramid [116]. Those formats are different in size and the number/shape of faces [116]. Some previous studies have compared the performances of different projection formats. It is found that the EAP format yields around 8.3% bitrate saving relative to the ERP format [117], and the ERP and CMP formats outperform the Pyramid format [118]. Also, [54] suggests that the CMP format is better than the ERP and Pyramid formats. Nowadays, the ERP and CMP formats are the most widely used in practice [119].

In tiling-based approaches, a 360 video is spatially divided into tiles. The most popular tiling method is to partition every face of the video into tiles of equal size using a P×Q grid, such as 6×4 (ERP) [44], 8×8 (ERP) [76], 12×6 (ERP) [77], 2×2 (CMP) [53,78], and 4×4 (CMP) [78]. In [79], the optimal tiling scheme is determined for each segment based on the user viewing behaviors. Also, some non-uniform tiling schemes have been used to exploit the content characteristics [82] and/or the user access preferences [43,80,81]. Each tile is then encoded into multiple versions of different bitrates [76,80], QP values [44,45], or resolutions [46].

So far, some encoding techniques for 360 video have been proposed such as region adaptive distortion calculation [120], adaptive QP selection [121], weighted-based rate control [122], spherical geometry padding [123], and fast intra estimation [124]. In addition, evaluation frameworks for 360 video coding have been designed in [117,125].

In the literature, several viewing direction estimation methods have been proposed, such as average-based [49], linear regression-based [49], and neural network-based [113,126]. It is found that future viewing direction can be predicted well (accuracy of more than 80%) within the next 1sec [49,113,126]. Yet, the estimation accuracy drops significantly for longer estimation window [49].

With respect to tile version selection, one of the simplest methods is to select the highest possible version for the visible tiles and the lowest version for the other tiles according to the estimated viewport [43–46]. Alternatively, utilizing Scalable Video Coding, the client fetches the base layers of all tiles, whereas only the visible tiles' enhancement layers are further fetched to enhance the viewport quality [53]. This method, however, suffers significant quality degradations due to estimation errors, which are unavoidable due to the randomness of head movements [49]. Moreover, a problem formulation has not been presented in [43]. In [113], the authors propose to trim each video frame according to the predicted viewport. The trimmed frame is then transmitted to the client. This method is actually not a tiling-based method.

To cope with viewport estimation errors, some previous studies propose to deliver the other tiles rather than the visible tiles at high quality [49,80,81]. However, no specific algorithm for selecting tiles' versions is given in [49]. Two tile selection algorithms are presented in [81] and [80]. Yet, these algorithms use the constant values of some important parameters such as the extension width [81] and the portion of bandwidth allocated for the visible tiles [80].

Another method is to use the tiles' viewing probabilities to assist tile selection [77]. Yet, [77] is only tested under idealistic scenarios where 1) the head movements are assumed to be constant at a given angular speed and 2) the network delay and head movement speed are known beforehand. In addition, some tiles may not be delivered to the client. Thus the user will experience blank block within the viewport if those tiles turn out to be visible tiles. Such cases can severely impact the user experience.

Although the work of [94] and our proposed method use the same idea to calculate the objective quality (or expected quality), the method for calculating the objective quality in our method is different from that of [94]. Specifically, in our proposed method, the viewport quality at each frame is first calculated given the estimated viewport position at that frame. Then, the objective quality is calculated a function of the frames' viewport quality. Meanwhile, in [94], the expected quality is calculated for the whole segment. In addition, the weights of the tiles are obtained from the viewing preferences of other viewers watching the same video. Though the results in [94] also show the viewport PSNR values over time, the streaming test-bed used to produce those results was not described. In addition, the performances of tile selection methods under different user's head movement patterns have not been analyzed. In this work, an analysis of the behaviors of the proposed and reference methods when 1) the viewport positions are static and 2) the viewport positions change frequently is presented. It is shown that the considered methods behave differently under the two head movement patterns. Note that, such an analysis has not been performed in previous studies. The work in [94] has been extended to study the live 360 multicast in [127].

To deliver the tiles' versions over the networks in an efficient manner, important tiles can be rerouted to congestion-free network links using Software Defined Network [128] or first sent by the server using HTTP/2's priority feature [99]. Also, HTTP/2's server-push feature can be used to increase achieved throughput in high RTT networks [81].

In most previous studies, bandwidth saving is used as the key performance metric [43–46, 126]. Yet, the bandwidth saving cannot accurately reflect the video quality perceived by users. In [117], a new metric called viewport PSNR (V-PSNR) has been proposed. Some recent studies have employed V-PSNR for evaluation such as [44, 54, 77, 80, 94, 127]. Yet, only the average V-PSNR values of the entire video [44, 77] or the segments [80] are reported.

3.3 Proposed Framework

3.3.1 System Architecture

Fig. 3.1 shows the general architecture of our proposed system where a server streams a 360 video to a client running on the user's device. The server includes a preprocessing module, a decision engine module, and a sender module. The preprocessing module first converts the input 360 video into a planar format using the ERP format [116]. Then, it spatially divides the converted video into multiple tiles of equal size. The preprocessing module then encodes each tile into multiple versions. The bitrate/quality information of the tiles' versions and other description information are stored in the metadata, which is used by the decision engine. Given the metadata and the client's feedbacks, the decision engine makes decisions on the tiles' versions. The sender delivers the selected tiles' versions to the client. For low delay, the tiles of each frame are sent at the same time.

The client includes a receiver module, a player module, and a system monitor module. The receiver module receives and decodes the tiles' versions, then stitches the decoded tiles' versions to reconstruct the 360 video. The player extracts and displays the viewport corresponding to the user's current viewing direction. The system monitor is responsible for two tasks, which are 1) monitoring the network status (e.g., throughput) and the user viewing directions, and 2) periodically sending feedbacks to the server.

Fig. 3.2 shows the data flows in the proposed system. Similar to cloud gaming [129], the server sends video data to the client on frame basis. The adaptation is executed every adaptation interval (segment) that consists of L frames and contains τ seconds of the video. Segment k $(k \geq 1)$ is transmitted at time

$$t_k^s = (k-1) \times \tau. \tag{3.1}$$

To ensure that the determinations of tiles' versions of segment k are available before its transmission time, the decision engine decides the versions of the tiles of segment k at time

$$t_k^d = t_k^s - \delta t, \tag{3.2}$$

where δt is the maximum amount of time for making decisions, which depends on the server's processing speed as well as the complexity of the tile selection algorithm. At the client, as soon as the first B frames have completely received, the client starts playing the video. For every displayed video frame, the client (i.e., the system monitor) sends a feedback containing 1) the instant download throughput and 2) the user's current viewing direction (viewport position) to the server.

3.3.2 Problem Formulation

Suppose that, at a given time, the server needs to adapt a segment consisting of L frames to meet a bitrate constraint R^c. The segment is composed of M tiles, each is available in N versions where version 1 (N) has the lowest (highest) quality. Denote V_l the viewport position when the user watches the $l^{th}(1 \leq l \leq L)$ frame of the segment. Let v_m denote the version selected for tile m $(1 \leq m \leq M)$. The version v_m has a bitrate R_m and a distortion D_m. Note that, the bitrate and distortion are computed as the average values over all frames of the segment. The tile selection problem can be formulated as an optimization problem as follows.

Find $\{v_1, v_2, ..., v_M\}$ *to maximize a quality objective VQ which is a function of* $\{D_m\}_{1 \leq m \leq M}$ *and* $\{V_l\}_{1 \leq l \leq L}$

$$VQ = f\left(D_1, D_2, ..., D_M, V_1, V_2, ..., V_L\right) \tag{3.3}$$

and satisfy

$$\sum_{m=1}^{M} R_m \leq R^c. \tag{3.4}$$

To solve the above problem, our solution consists of the following aspects.

- Estimation of the bitrate constraint R^c (or throughput) and the viewport positions $V_l(1 \leq l \leq L)$. Note that in fact, any throughput and viewport estimation methods can be used in our framework.
- Computation of the quality objective VQ.
- Decision on the optimal version of each tile.

In the following section, each of these aspects will be addressed.

3.4 Proposed Approach

In this section, the proposed approach to determine tiles' versions is described. Note that, the decision engine on the server is responsible for all important tasks, including throughput estimation, viewport estimation, and tile version selection. The solution below is for adapting a segment k at the server. It is assumed that, at the decision making time, the server has just received the feedback of the l_{last}^{th} frame of the segment $k_{last} < k$.

3.4.1 Estimations of Throughput and Future Viewport Position

The estimated throughput $T^e(k)$ of the segment k is simply set to the throughput reported in the last client's feedback $T_{fb}(k_{last}, l_{last})$, i.e., $T^e(k) = T_{fb}(k_{last}, l_{last})$. The bitrate constraint R^c is then computed from $T^e(k)$ by

$$R^c = (1-\alpha) \times T^e(k), \tag{3.5}$$

where the safety margin α is in range $[0, 0.5]$ as shown in our previous work [130].

In this work, a linear regression method is used for viewport estimations. Different from [49], the estimation errors of past frames are taken into account. As mentioned above, the server needs to estimate the viewport position of each frame in segment k. Let $V^e(k,l)$ denote the estimated viewport position of the l^{th} $(1 \leq l \leq L)$ frame of the segment k. Let $E(k)$ denote the estimated viewport error of the segment k, which is set equal to the estimation error of the first frame of the segment k_{last}. Specifically, the value of $E(k)$ is given by

$$E(k) = V^e(k_{last}, 1) - V(k_{last}, 1). \tag{3.6}$$

Let $S_{avg}(k)$ denote the user's average head movement speed over the last L frames. The value of $S_{avg}(k)$ is given by

$$S_{avg}(k) = \frac{V(k_{last}, l_{last}) - V(k_{last}-1, l_{last})}{\tau}. \tag{3.7}$$

Note that, the values of $V(k_{last}, l_{last})$ and $V(k_{last}-1, l_{last})$ are obtained from the client's feedbacks. The estimated viewport position $V^e(k,l)$ of the l^{th} frame of the segment k is computed as follows.

$$V^e(k,l) = V(k_{last}, l_{last}) + (l-1) \times \frac{\tau}{L} \times S_{avg}(k) + \frac{l-1}{L-1} E(k). \tag{3.8}$$

3.4.2 Computation of Viewport Quality

In this part, the viewport quality value $VQ(k,l)$ of each frame of the segment k given the distortions $\{D_m(k)\}$ $(1 \leq m \leq M)$ and the estimated viewport positions $\{V^e(k,l)\}$ $(1 \leq l \leq L)$ are computed.

As the user only watches a portion of the full 360 video (i.e., the viewport) at each time instant, the contribution of a tile to the viewport quality is dependent on how that tile overlaps the viewport. Thus, the viewport quality distortion value $VD(k,l)$ of the l^{th} frame $(1 \leq l \leq L)$ is computed as a weighted sum of the distortion values of all tiles as follows.

$$VD(k,l) = \sum_{m=1}^{M} w_m(V^e(k,l)) \times D_m(k). \tag{3.9}$$

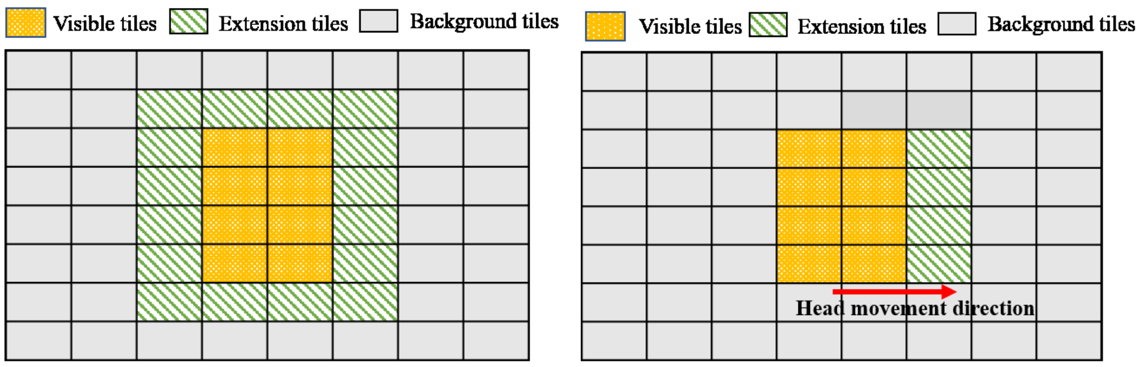
Here, $w_m(V)$ is the weight of tile m given the viewport position V. $w_m(V)$ is calculated as the fraction of the overlapped area of tile m to the viewport area given the viewport position V. If tile m does not overlap the viewport, then $w_m(V) = 0$. In this work, the distortion is measured in terms of the Mean Square Error (MSE), so the viewport quality value $VQ(k,l)$ of the l^{th} frame $(1 \leq l \leq L)$ is converted into PSNR as follows.

$$VQ(k,l) = 10 \times log_{10}\left(\frac{255^2}{VD(k,l)}\right). \tag{3.10}$$

The quality objective VQ in the above problem formulation can be computed from the obtained values of $VQ(k,l)$ $(1 \leq l \leq L)$.

3.4.3 Tile Selection Method

In this part, methods to select the version of each tile will be presented. The basic idea is to extend the viewport to deal with viewport estimation errors. However, different from previous studies, the proposed method can systematically decide 1) the coverage of the extension area and 2) the versions of the viewport and extension areas. Note that, the tiles in the background area always have the lowest version.



(A) Option 1: Extending in all directions.

(B) Option 2: Extending according to the predicted viewport positions.

FIGURE 3.3: Illustrations of two options for extending the viewport area.

If the user head movements are very difficult to estimate (or estimation error is high), the viewport should be extended in all directions so that any estimation errors could be tolerated. However, when the user head movements are highly predictable, extending the viewport in all directions will waste the available bandwidth since only a part of the extension area is actually useful. In this work, two options for extending the viewport area are considered, as illustrated in Fig. 3.3. In the following, the method to decide the versions of tiles for each option is described.

Option 1

In this option, the viewport will be extended in all directions. The width of the *extension area* should be dependent on the viewport estimation error. The higher the estimation error is, the wider the extension area should be. An extension area with the width of I $(1 \leq I \leq I_{max})$ is formed by extending the viewport area by I tiles in all directions. Note that, the extension area does not include the tiles of the viewport area. The maximum width of the extension area I_{max} is dependent on the tiling scheme. The extension area of the width of I is divided into I ranges. Specifically, Range 1 is identical to the extension area with the width of 1 tile. Range i $(2 \leq i \leq I)$ contains tiles that are in the extension area with the width of i but not in the extension area with the width of $(i-1)$. All tiles belonging to the same range have the same version.

Given the expected viewport quality value of every frame, the overall quality value of the segment can be estimated. As the viewport quality will change over the segment due to user head movements, the quality objective *VQ(k)* is computed as the average of the viewport quality values of the first and last frames as follows.

$$VQ(k) = \frac{1}{2} \times (VQ(k,1) + VQ(k,L)). \tag{3.11}$$

Let v_0 be the version selected for the visible tiles, v_i the version selected for extension range $i (1 \leq i \leq I)$. Note that, v_i can take values from 1 (lowest quality) to N (highest quality). The general procedure to decide the tiles' versions can be summarized as follows.

- **Step 1**
  - Compute bitrate constraint R^c using Eq. (3.5).
  - Compute the estimated viewport positions $\{V_k^e(l)\}_{1 \leq l \leq L}$ using Eqs.(3.6)(3.7)(3.8).
- **Step 2**

  For each extension width $I (1 \leq I \leq I_{max})$

  For each set of $(v_0, v_1, ..., v_I)$ so that $v_0 \geq v_1 \geq ... \geq v_I$

  1. Compute the quality objective *VQ(k)* using Eqs. (3.9)(3.10)(3.11).
  2. Compute the total bitrate of all tiles including that in the background area.
- **Step 3**: Select the extension width I and the set of $(v_0, v_1, ..., v_I)$ that result in the highest quality objective *VQ(k)* and satisfy the condition in Eq. (3.4).

In the above algorithm, the viewport area is always ensured to have the highest quality. For the extension ranges, the version is gradually decreasing when moving away from the viewport center. When the width of the extension area is equal to I_{max}, there is no background area.

Option 2

Similar to the option 1, the extension area is first formed by extending the viewport in all directions. However, each range of the extension area is further divided into a number of sub-areas. Fig. 3.4 shows an example where every extension range is divided into 4 sub-areas. The tiles belonging to the same sub-area will have the same version. Let v_0 the version selected for the visible tiles, $v_{i,j}$ the version selected for the sub-area $j (1 \leq j \leq J_i)$ of range $i (1 \leq i \leq I)$ where J_i denotes the number of sub-areas of the extension range i.

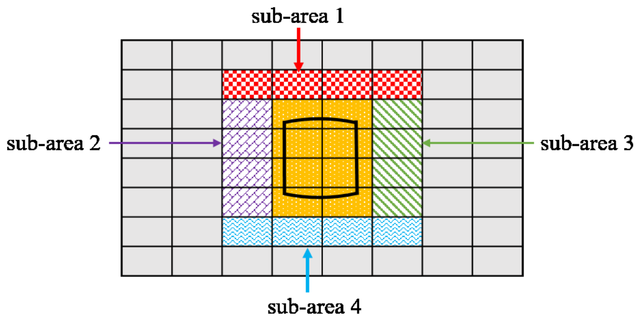


FIGURE 3.4: An example of the sub-areas.

If the number of sub-areas is high, searching over all possible selections of sub-areas' versions could be so time-consuming that is not feasible for real-time adaptation. Thus, we add two constraints regarding the version of each sub-area in order to reduce the processing time. First, the viewport area always has the highest possible version. Second, the tile version of any sub-area of an extension range i is higher than that of any sub-area of the range $(i+1)$. The two constraints are mathematically defined by equations (3.12) and (3.13).

$$v_0 \geq v_{i,j}, \text{for all } i, j \neq 0 \tag{3.12}$$

$$v_{i,j} \geq v_{i+1,j'} \text{for all } i, j, j' \tag{3.13}$$

These two constraints ensure that the quality is gradually decreasing from the viewport center to the periphery.

As for the objective quality, it is computed as the average of the estimated viewport quality values of all video frames of the segment k as follows.

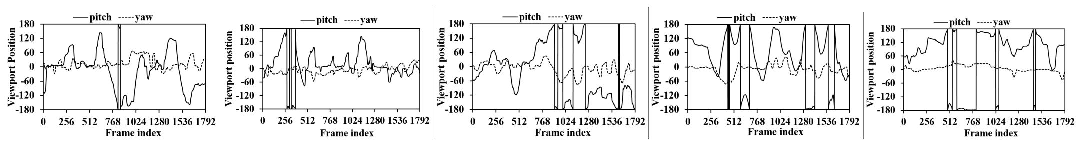
$$VQ(k) = \frac{1}{L} \times \sum_{l=1}^{L} VQ(k,l). \tag{3.14}$$

The general procedure to decide the versions of tiles can be summarized as follows.

- **Step 1**
  - Compute bitrate constraint R^c using Eq. (3.5).
  - Compute the estimated viewport positions $\{V_l^e\}_{1 \leq l \leq L}$ using Eqs. (3.6)(3.7)(3.8).
- **Step 2** For each set of $\{v_{i,j}\}_{1 \leq i \leq I_{max}, 1 \leq j \leq J}$ satisfying the constraints in Eqs. (3.12)(3.13)(3.4)

TABLE 3.1: Descriptions of five 360 videos used in the experiments.

Video	Name	YouTubeID	Start offset	Content Description
#1	Yakitori	pQyt6H7GlcY	40s	Exploring night streets in Tokyo. Medium moving camera. No main focus
#2	Diving	2OzlksZBTiA	40s	Diving scene. Slowly moving camera, no clear horizon. No main focus expected within the sphere.
#3	RollerCoaster	8lsB-P8nGSM	65s	Rollercoaster. Fast moving camera fxed in front of a moving roller-coaster. Strong main focus following the rollercoaster trail.
#4	Timelapse	CIw8R8thnm8	0s	Timelapse of city streets. Fixed camera, clear horizon with a lot of fast moving people/cars, many scene cuts. Focus expected along the equator line.
#5	Venice	s-AJRFQuAtE	0s	Virtual aerial reconstruction of Venice. Slowly moving camera. No main focus expected within the sphere.



(A) Video #1. (B) Video #2. (C) Video #3. (D) Video #4. (E) Video #5.

FIGURE 3.5: The head movement traces of the five videos.

1. Compute the quality objective *VQ(k)* using Eqs. (3.9)(3.10)(3.14).
2. Compute the total bitrate of all tiles including that in the background area.

- **Step 3**: Select the set of $(v_0, v_1, ..., v_{I_{max}*J})$ that results in the highest quality objective *VQ(k)*.

The above algorithm is implemented using nested *for* loops, each for a sub-area. Compared to the full-search approach, the proposed algorithm can reduce the computation time by approximately 99%. In this experiment, the processing time of the above algorithm is always less than 1msec on an Ubuntu 14.04 LTS 64bit machine with 8 GB RAM, Intel Core i5-2500 CPU 3.3GHz.

3.5 Evaluation

3.5.1 Experiment Settings

For experiments, five 360 videos from Youtube with different content types are used. Table 3.1 summarizes the content features of the four videos. The considered videos have 1792 frames, a resolution of 3840×1920 (4K), and a frame rate of 30 fps. The Field of View (FoV) of the viewport is 90 degrees both horizontally and vertically. The video is divided into $M = 64$ tiles (i.e., 8×8 tiling) as in [76], each has a resolution of 480×240. The maximum width of the extension area I_{max} is 3. Each tile is encoded into $N = 7$ versions corresponding to 7 QP

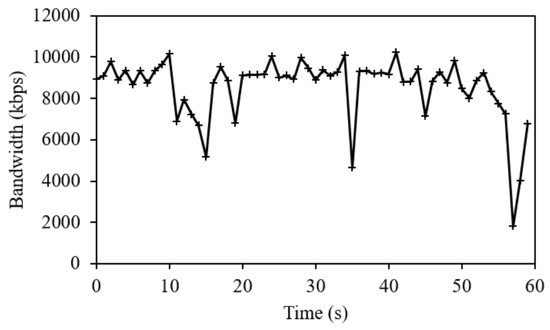
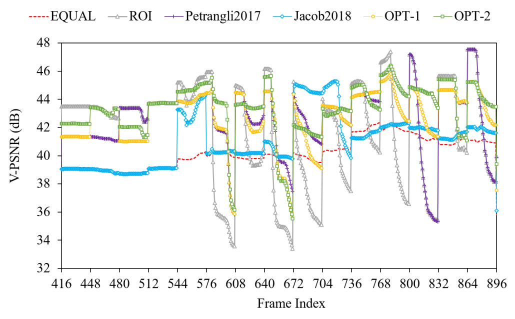


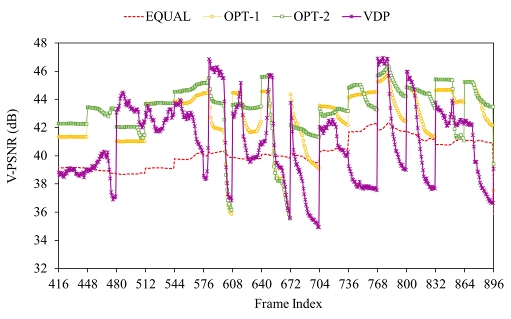
FIGURE 3.6: The used bandwidth trace.

values of 24, 28, 32, 36, 40, 44, and 48 using HEVC format. To enable fast processing time, the low-delay B profile with the Group of Picture (GoP) size of 4 frames is used. According to the previous study of [130], the value of the safety margin should be in [0, 0.5] range. In this work, the safety margin α in Eq. (3.5) is set to 0.2 according to [130]. Each streaming session lasts for 59.7 s. Each segment contains $L = 32$ frames. As aforementioned, the processing time of the proposed method is always less than 1 ms. Thus, in order to use the latest feedback information from the client, the value of δt in Eq. (3.2) is set to 33 ms, which is equal to the playback duration of one video frame at the frame rate of 30 fps. In the option 2 of the proposed method, the value of J is set to 4 to cover four main changes in viewing direction of left, right, top, and bottom. In the future work, other values of J will be considered. The buffer size B is set to 1. That means the client starts playing after it has fully received the first frame.

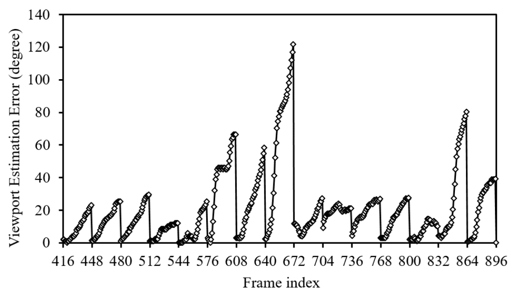
The proposed system is implemented based on Gaming Anywhere, an open-source cloud gaming platform [129]. The client is written in C++ and running on a Ubuntu 14.04 LTS 64bit machine with 4 GB RAM, Intel Core i5-3210M CPU 2.5 GHz. The server is running on another Ubuntu 14.04 LTS 64bit machine with 8 GB RAM, Intel Core i5-2500 CPU 3.3 GHz. The client is connected to the server via a router. The network conditions are emulated using DummyNet [131] in which the Round-Trip Time delay is set to 50 ms. Two head movement traces are used, one contains the periods of steady movements while the other has a lot of changes in moving directions. The used head movement traces are plotted in Fig. 3.5. One head movement trace is used for each video of the five videos. Here, the viewport position at a given time is determined by a (pitch, yaw) pair in degrees with $-180 \leq \text{pitch} \leq 180$ and $-90 \leq \text{yaw} \leq 90$ [116]. During a streaming session, the selected version of each tile and the corresponding viewport position are logged. After the streaming session finishes, the tiles' versions are combined to reconstruct the 360 video. Then the viewport is extracted, and the



(A) Frames' V-PSNR values(dB).



(B) Frames' V-PSNR values(dB).

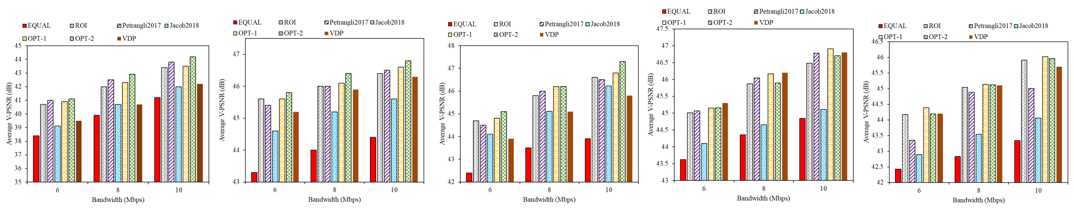


(C) Frames' V-PSNR valuse(dB).

FIGURE 3.7: V-PSNR values and viewport positions of video frames #416−896 of all considered methods when the network bandwidth is 8 Mbps under the head trace #1 (Video #1).

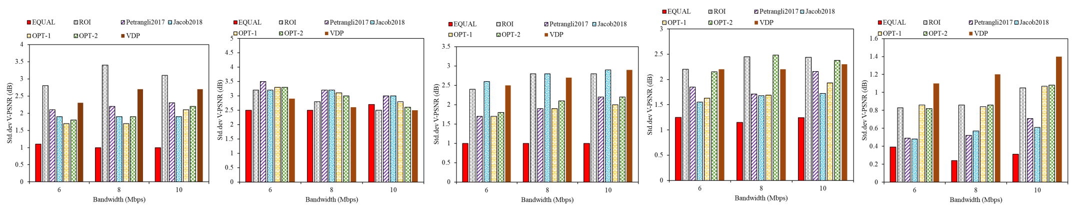
actual viewport PSNR (V-PSNR) of each frame is calculated.

The proposed approach is compared to five reference approaches. The first approach, denoted *ROI* (i.e., [43–45]), selects the lowest version for the background area and the highest possible version for the viewport area; no extension area is considered. The second approach, denoted *EQUAL*, selects the same version for all tiles. The third method, denoted *Petrangli2017*, classifies the tiles into three groups, namely *viewport*, *adjacent*, and *outside* [81]. Different from our proposed method, the viewport group in this method consists of the visible tiles corresponding to not only the estimated viewport postion but also the current viewport position. As the name suggests, the adjacent group consists of tiles that are adjacent to the viewport. It can be noted that the adjacent group is a special case of the extension area in our method when the extension width is one tile. This method always tries to maximize the version of the viewport group. The remaining bandwidth will be used to improve the version of the adjacent group, then the outside group. The fourth method, denoted *Jacob2018*, selects the versions of tiles so as to minimize a weighted sum of the distortions of tiles. The distortions of tiles are measured using the MSE metric. The weights of tiles are calculated from the viewing history [94]. In our implementation, five head traces from five different users watching the



(A) Video #1 (B) Video #2. (C) Video #3. (D) Video #4. (E) Video #5.

FIGURE 3.8: Average V-PSNR values of the proposed and four reference methods at three bandwidth values.



(A) Video #1 (B) Video #2. (C) Video #3. (D) Video #4. (E) Video #5.

FIGURE 3.9: Std.dev V-PSNR values of the proposed and four reference methods at three bandwidth values.

same video are used to calculate the weights of tiles. The fifth approach, denoted *VDP*, is an viewport-dependent approach with 36 pre-defined viewport locations. Given the estimated viewport position, the version with the associated viewport cloest to the estimated viewport will be selected. The two options of the proposed method are respectively denoted by *OPT-1* and *OPT-2*. For fairness, the reference approaches of *ROI*, *EQUAL*, *VDP*, and *Petrangli2017* are implemented using the same viewport estimation method as that used in the proposed method.

3.5.2 Constant Bandwidth Cases

In this part, the performances of the proposed method and the four reference methods under constant bandwidth scenarios will be investigated. Specifically, three bandwidth values of 6 Mbps, 8 Mbps, and 10 Mbps are considered.

Fig. 3.7 shows the V-PSNR values of the video frames #416-896 of the considered methods under the head trace #1 of video #1 when the network bandwidth is 8 Mbps.

It can be seen that when the viewport positions are quite static (i.e., frames #416−576), the V-PSNR values of the *ROI*, *Petrangli2017*, *OPT-1*, and *OPT-2* methods are similar. This is because that the viewport positions can be well estimated during this interval (i.e., Fig. 3.7b). As a result, the proposed method (both two options) behaves like the *ROI* method by selecting

TABLE 3.2: Average performance improvements of the proposed method compared to the four reference methods V-PSNR at different bandwidth values in terms of average V-PSNR (dB).

(A) The *OPT-1* option.

	BW=6 Mbps					BW=8 Mbps					BW=10 Mbps				
Video	EQUAL	ROI	Petrangli2017	Jacob2018	VDP	EQUAL	ROI	Petrangli2017	Jacob2018	VDP	EQUAL	ROI	Petrangli2017	Jacob2018	VDP
#1	2.5	0.2	-0.1	1.9	1.4	2.4	0.3	-0.2	2.1	1.6	2.3	0.1	-0.3	2.3	1.3
#2	2.4	0.1	0.3	0.7	0.4	2.7	0.4	0.2	1.1	0.2	2.9	0.2	0.3	0.6	0.3
#3	2.3	0.0	0.2	1.0	0.9	2.1	0.1	0.1	0.9	1.1	2.2	0.2	0.1	1.0	1.0
#4	1.5	0.1	0.1	1.1	-0.1	1.8	0.3	0.1	1.5	0.0	2.1	0.4	0.1	1.8	0.1
#5	2.0	0.2	1.0	1.5	0.2	2.3	0.1	0.3	1.6	0.0	2.7	0.1	1.0	2.0	0.3
Average	**2.1**	**0.1**	**0.3**	**1.2**	**0.6**	**2.3**	**0.2**	**0.1**	**1.4**	**0.6**	**2.4**	**0.2**	**0.2**	**1.5**	**0.6**
Max	**2.5**	**0.2**	**1.0**	**1.9**	**1.4**	**2.7**	**0.4**	**0.3**	**2.1**	**1.6**	**2.9**	**0.4**	**1.0**	**2.3**	**1.3**

(B) The *OPT-2* option.

	BW=6 Mbps					BW=8 Mbps					BW=10 Mbps				
Video	EQUAL	ROI	Petrangli2017	Jacob2018	VDP	EQUAL	ROI	Petrangli2017	Jacob2018	VDP	EQUAL	ROI	Petrangli2017	Jacob2018	VDP
#1	2.7	0.4	0.1	2.1	1.6	3.0	0.9	0.4	2.7	2.2	3.0	0.8	0.4	3.0	2
#2	2.7	0.4	0.6	1.0	0.6	2.7	0.4	0.2	1.1	0.5	3.4	0.7	0.8	1.1	0.5
#3	2.5	0.2	0.4	1.2	1.2	2.4	0.4	0.4	1.2	1.1	2.4	0.4	0.3	1.2	1.5
#4	1.5	0.1	0.1	1.1	-0.1	1.5	0.0	-0.1	1.2	-0.3	1.9	0.2	-0.1	1.6	-0.1
#5	1.8	0.0	0.8	1.3	0.0	2.3	0.1	0.2	1.6	0.0	2.6	0.1	1.0	1.9	0.3
Average	**2.2**	**0.2**	**0.4**	**1.3**	**0.7**	**2.4**	**0.4**	**0.2**	**1.6**	**0.7**	**2.7**	**0.4**	**0.5**	**1.8**	**0.8**
Max	**2.7**	**0.4**	**0.8**	**2.1**	**1.6**	**3.0**	**0.9**	**0.4**	**2.7**	**2.2**	**3.4**	**0.8**	**1.0**	**3.0**	**1.3**

TABLE 3.3: Average performance improvements of the proposed method compared to the four reference methods V-PSNR at different bandwidth values in terms of std.dev V-PSNR (dB).

(A) The *OPT-1* option.

	BW=6 Mbps					BW=8 Mbps					BW=10 Mbps				
Video	EQUAL	ROI	Petrangli2017	Jacob2018	VDP	EQUAL	ROI	Petrangli2017	Jacob2018	VDP	EQUAL	ROI	Petrangli2017	Jacob2018	VDP
#1	-0.6	1.1	0.4	0.8	0.6	-0.7	1.7	0.5	1.2	1	-1.1	1.0	0.2	1.2	0.6
#2	-0.7	0.7	0.0	0.9	0.4	-0.9	0.9	0.0	0.9	0.5	-1.0	0.8	0.2	0.9	0.3
#3	-0.8	-0.1	0.2	-0.1	0.8	-0.6	-0.3	0.1	0.1	0.8	-0.1	-0.3	0.2	0.2	0.9
#4	-0.4	0.6	0.2	-0.1	0.6	-0.5	0.8	0.0	0.0	0.5	-0.7	0.5	0.2	-0.2	0.4
#5	-0.5	0.0	-0.4	-0.4	0.2	-0.6	0.0	-0.3	-0.3	0.4	-0.8	0.0	-0.4	-0.5	0.3
Average	**-0.6**	**0.4**	**0.1**	**0.2**	**0.5**	**-0.7**	**0.6**	**0.1**	**0.4**	**0.6**	**-0.7**	**0.4**	**0.1**	**0.3**	**0.5**
Max	**-0.4**	**1.1**	**0.4**	**0.9**	**0.8**	**-0.5**	**1.7**	**0.5**	**1.2**	**1.0**	**-0.1**	**1.0**	**0.2**	**1.2**	**0.9**

(B) The *OPT-2* option.

	BW=6 Mbps					BW=8 Mbps					BW=10 Mbps				
Video	EQUAL	ROI	Petrangli2017	Jacob2018	VDP	EQUAL	ROI	Petrangli2017	Jacob2018	VDP	EQUAL	ROI	Petrangli2017	Jacob2018	VDP
#1	-0.7	1.0	0.3	0.7	0.5	-0.8	0.6	-0.1	0.8	0.8	-0.8	-0.1	0.2	-0.1	0.5
#2	-0.8	0.6	-0.1	0.8	0.4	-0.8	-0.1	0.2	-0.1	0.4	-0.9	0.1	-0.3	-0.6	0.1
#3	-0.8	-0.1	0.2	-0.1	0.7	-0.9	0.1	-0.3	-0.6	0.6	-0.4	0.0	-0.3	-0.3	0.7
#4	-0.9	0.1	-0.3	-0.6	0.1	-0.4	0.0	-0.3	-0.3	-0.3	0.0	0.0	0.0	0.0	-0.1
#5	-0.4	0.0	-0.3	-0.3	0.3	0.0	0.0	0.0	0.0	0.3	0.0	0.0	0.0	0.0	-0.3
Average	**-0.7**	**0.3**	**0.0**	**0.1**	**0.4**	**-0.6**	**0.1**	**-0.1**	**0.0**	**0.4**	**-0.4**	**0.0**	**-0.1**	**-0.2**	**0.2**
Max	**-0.4**	**1.0**	**0.3**	**0.8**	**0.7**	**0.0**	**0.6**	**0.2**	**0.8**	**0.8**	**0.0**	**0.1**	**0.2**	**0.0**	**0.7**

the highest possible version for the visible tiles and the lowest version for the extension tiles. The current and estimated viewport positions will be very close to each other when the viewport positions are stable. Consequently, the viewport group of the *Petrangli2017* method and the viewport area of the proposed method will be almost identical. Meanwhile, the *EQUAL* method has significantly lower V-PSNR values since it selects the same version for all tiles. As a significant amount of bandwidth is consumed by the invisible tiles, the versions of the visible tiles are significantly reduced. It can be noted that the *Jacob2018* method results in the same V-PSNR curve as that of the *EQUAL* method for frames #416-544. This is because that none of the reference traces starts those segments at the same tile as that of the considered trace. Thus, it is not possible to determine the navigation likelihood of tiles using the reference traces. In this case, all tiles are assigned the same weight. As a result, the *Jacob2018* method selects the same version for all tiles. From frame #544 to frame #575, it can be seen that the frame V-PSNR of the *Jacob2018* method is only slightly lower than that of the *OPT-1* option of the proposed method. This is because that the user looks at a similar viewing direction to those of the reference traces. This result indicates that the performance of the *Jacob2018* method is strongly dependent on the reference traces. Especially, it seems that this method does not work well with a small number of reference traces.

When the viewport positions start changing quickly (i.e., frames #576-896), the *ROI* method experiences drastic viewport quality reductions within each segment. The V-PSNR values are usually high at the beginning of each segment, but decrease significantly later. This can be explained as follows. As can be seen in Fig. 3.7b, the viewport estimation error are very small at the beginning of each segment. Yet, it increases significantly during each segment. The high viewport estimation errors cause some invisible tiles, which is associated with the estimated viewport position, become visible. Because the *ROI* method selects the lowest version for those tiles, the viewport quality is reduced significantly. It can be seen that the *Petrangli2017* method can achieve similar frame V-PSNR values to that of the *OPT-1* option of the proposed method for most of the frames. Yet, the frame V-PSNR of this method drops drastically during segment #25 (frames #800-831) and segment #27 (frames #864-895).

It can be seen that, similar to the *ROI* method, the *VDP* method also results in drastic viewport quality fluctuations throughout the session. It is because that this method cannot cope with high viewport estimation errors. Even when the viewport positions are very stable, the performance of the *VDP* method is still very low. This results indicate that viewport-dependent approach is not effective for 360 video streaming. Of course, one can use a huge number of pre-defined viewports to improve the performance of this method. Yet, this would require a huge amount of data storage at the server side.

Meanwhile, the proposed method can provide much more stable V-PSNR values than those of the *ROI* method. This indicates that our proposed method is effective in dealing with the viewport estimation errors. It can also be noted that, the *OPT-2* option achieves higher and more stable V-PSNR values than those of the *OPT-1* option for most of the segments. This is because that the viewport position can be well predicted in this case (i.e., Fig. 3.7c). In addition, there are some segments in which the *OPT-1* option is comparable or better than the *OPT-2* option (e.g., segment #20 (frames #640-671) and segment #26 (frames #832-863)). This is due to the fact that the head movement direction suddenly changes at those segments.

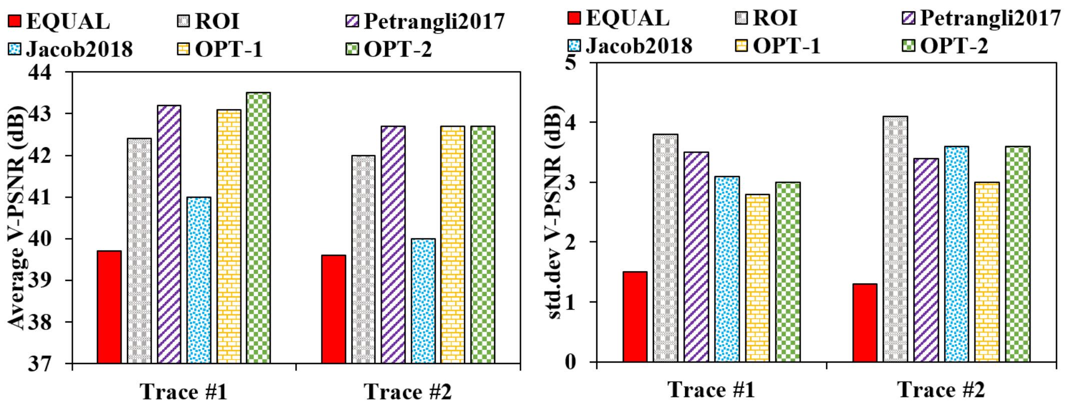
As a result, the estimated viewport positions are very inaccurate as can be seen in Fig. 3.7c. In such a case, the *OPT-2* option suffers from significant quality degradations since it chooses the extension tiles according to the estimated viewport. Meanwhile, the *OPT-1* option can tolerate the errors as the viewport is extended in all directions, thereby achieving higher V-PSNR.

Fig. 3.8 and Fig. 3.9 compare the proposed method with the reference methods, in terms of average and standard deviation (std.dev) V-PSNR, using five videos and at three different bandwidth values of 6 Mbps, 8 Mbps, and 10 Mbps. It can be seen that the proposed method always achieve the highest V-PSNR values while providing stable viewport quality. This result indicates that the proposed method is effective in improving the viewport quality for various types of content in different network conditions.

Table 3.2 and Table 3.3 summarize the improvements of the proposed method over the four reference methods. The last two rows in Table 3.2 and Table 3.3 show the average and maximum gains over all videos at each bandwidth value. In particular, the *OPT-1* of the proposed method can improve the average V-PSNR by up to 2.9 dB compared to the *EQUAL* method, up to 0.4 dB compared to the *ROI* method, up to 1.0 dB compared to the *Petrangli2017* method, and up to 2.3 dB compared to the *Jacob2018* method. The *OPT-2* option can improve the average V-PSNR by up to 3.4 dB compared to the *EQUAL* method, 0.9 dB compared to the *ROI* method, 1dB compared to the *Petrangli2017* method, and 3.0 dB compared to the *Jacob2018* method. As shown in Table 3.3, the proposed method can reduce the std.dev V-PSNR compared to all reference methods except for the *EQUAL* method. Note that, despite of having the lower std.dev V-PSNR values than the proposed method, the *EQUAL* method results in very low average V-PSNR values. It can be noted that the *OPT-2* option has a slightly higher performance than the *OPT-1* option in terms of average V-PSNR, but has lower improvement in terms of std.dev V-PSNR. The reason is that the *OPT-1* extends the viewport area in all directions, whereas the viewport area is extended in only one specific direction in the *OPT-2* option.

3.5.3 Variable bandwidth case

In this subsection, the performance of our proposed method under a real bandwidth trace will be evaluated. For that purpose, a bandwidth trace recorded using a mobile client under 4G network [132] is used. The bandwidth trace is shown in Fig. 3.6. Here, only the Video #1 is



(A) Average V-PSNR (dB).

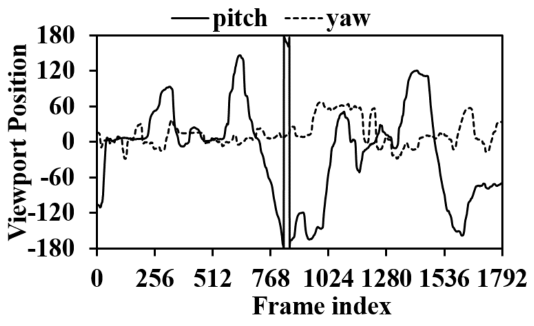
(B) Std.dev V-PSNR (dB).

FIGURE 3.10: A comparison of the considered methods under a real bandwidth trace (Video #1).

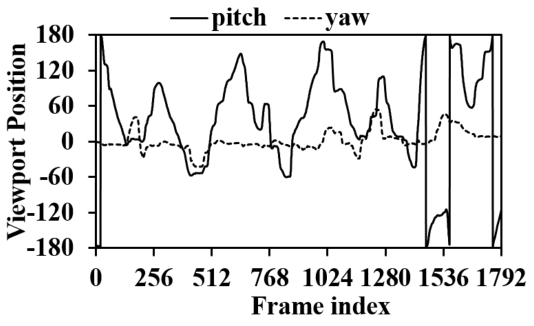
TABLE 3.4: Performance improvements of the proposed method under a real bandwidth trace (Video #1).The gains in case of the trace #1 are of the *OPT-2* option. The gains in case of the trace #2 are of the *OPT-1* option.

Trace	Metrics		EQUAL	ROI	*Petrangli2017*	*Jacob2018*	OPT-1	OPT-2
Trace #1	Average V-PSNR	Value (dB)	39.7	42.4	43.2	41	43.1	43.5
		Gain (dB)	+3.8	+1.1	+0.3	+2.5	-	-
	Std.dev V-PSNR	Value (dB)	1.5	3.8	3.5	3.1	2.8	3
		Gain (dB)	-1.5	+0.8	+0.5	+0.1	-	-
Trace #2	Average V-PSNR	Value (dB)	39.6	42	42.7	40	42.7	42.7
		Gain (dB)	+3.1	+0.7	0.0	+2.7	-	-
	Std.dev V-PSNR	Value (dB)	1.3	4.1	3.4	3.6	3	3.6
		Gain (dB)	-1.7	+1.1	+0.4	+0.6	-	-

considered. Beside the head movement trace shown in Fig. 3.5, an additional head movement trace of the Video #1 is used. The two head movement traces used in this experiment are shown in Fig. 3.11 The average V-PSNR and std.dev V-PSNR values of the proposed and four reference methods are shown in Fig. 3.10. Similar to the constant bandwidth cases, the *OPT-2* option achieves the highest average V-PSNR value under the head trace #1, whereas the *OPT-1* option performs the best under the head trace #2. The performance improvements of the proposed method are summarized in Table 3.4. Under the head trace #1, the *OPT-2* option can increase the average V-PSNR by up to 3.8 dB. Also, it can reduce the std.dev V-PSNR by 0.8 dB compared to the *ROI* method. The gains in terms of average V-PSNR compared to the *Petrangli2017* and *Jacob2018* methods are respectively 0.3 dB and 2.5 dB. Under the head trace #2, the *OPT-1* option achieves the highest average V-PSNR value, which is 3.1 dB higher than that of the *EQUAL* method. Besides, the *OPT-1* option can reduce the std.dev V-PSNR by 1.1 dB compared to the *ROI* method. These results indicate that the prosed method outperforms the reference methods under the variable bandwidth trace.



(A) Trace #1.



(B) Trace #2.

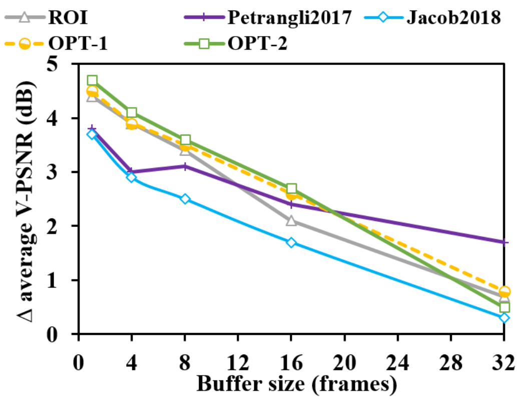
FIGURE 3.11: The two head movement traces of Video #1.

3.5.4 Impacts of Content Characteristics

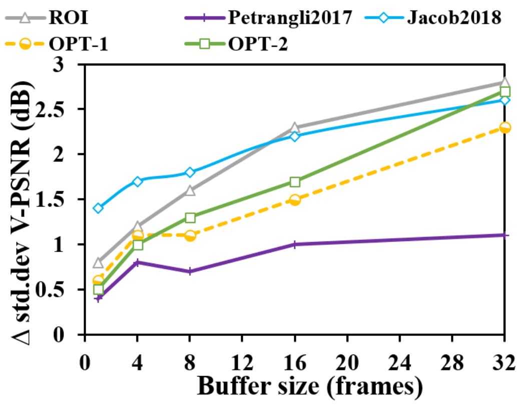
In this subsection, the impacts of content characteristics to the performance of the proposed method will be investigated.

It can be seen that different content types have different head movement characteristics that in turn affect the performances of the proposed and reference methods. Specifically, the proposed method can clearly improve the viewport quality for Video #2 and Video #3. However, the improvement gains compared to the reference methods become smaller in case of Video #4 and Video #5. This can be explained as follows. As shown in Fig. 3.5, the user usually changes the viewing directions while watching Videos #2 and Video #3. Thus, it is necessary to deliver a number of extension tiles at high quality. As a result, the two options of the proposed method can improve the viewport quality compared to that of the *ROI* method that does not consider any extension tiles. Moreover, it can be seen that the *OPT-2* option achieves higher average V-PSNR than the *OPT-1* option for all the cases except when the bandwidth is 8 Mbps with Video #2. It is interesting to see that the *Petrangli2017* method has lower average V-PSNR values than that of the *ROI* method when the bandwidth is 6 Mbps and 10 Mbps with Video #2. In addition, the average V-PSNR of the *Jacob2018* method is the second lowest among the considered methods/options. This is because that the *Jacob2018* method calculates the navigation likelihood of the tiles using prior navigation traces. As a consequence, this method does not work well under the head movement traces that contain a significant number of new viewing directions.

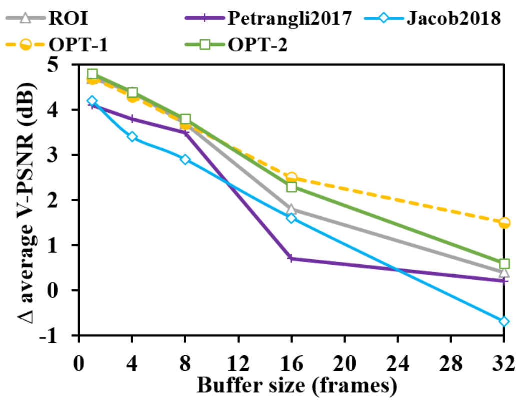
On the other hand, the performances of the *ROI* method, the *OPT-1* and *OPT-2* options of the proposed method are comparable under Video #5 as shown in Fig. 3.8d. The reason is that



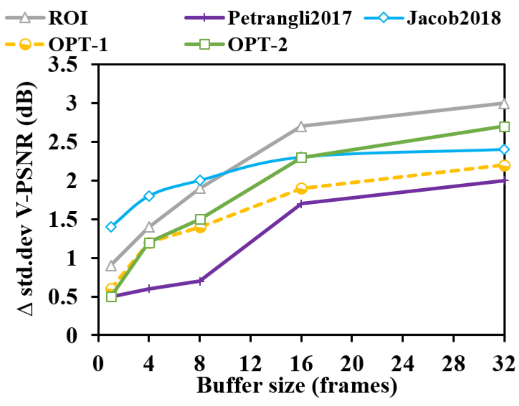
(A) Trace #1, Δ*V-PSNR*$_{avg}$.



(B) Trace #1, Δ*V-PSNR*$_{std}$.



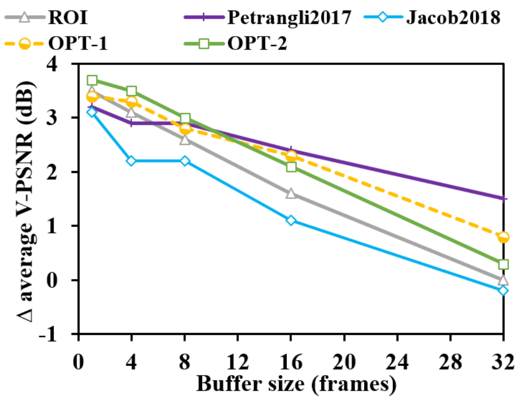
(C) Trace #2, Δ*V-PSNR*$_{avg}$.



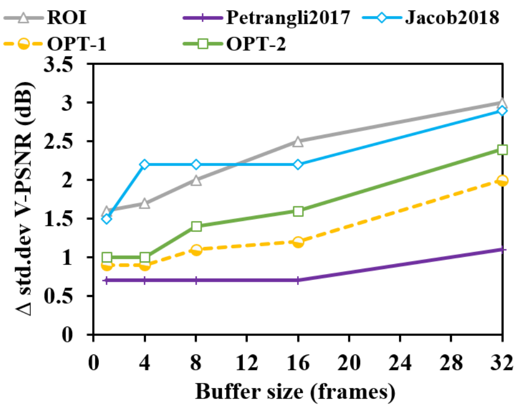
(D) Trace #2, Δ*V-PSNR*$_{std}$.

FIGURE 3.12: The impacts of the buffer size and the segment duration when $L = 4$ frames (Video #1).

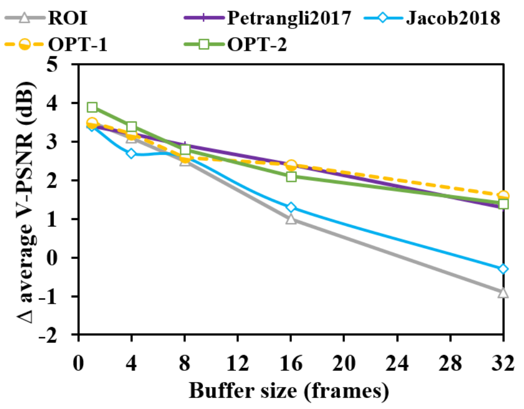
the user' viewing directions are mostly stable throughout the viewing session of this video as can be seen in Fig. 3.5d. Thus, the *ROI* method can achieve good performance since there is almost no need for the extension tiles. This also explains why the two options of the proposed methods have very similar performances. In contrast, it can be noted that the *Petrangli2017* method has quite lower average V-PSNR values than those of the *ROI* method for all three bandwidth values though the viewport area of this method is very similar to that of the *ROI* method. The lower performance of the *Petrangli2017* method is caused by the fact that this method requires all tiles in the viewport area must have the same version. This may cause a significant amount of bandwith be unnecessarily allocated to the adjacent area. It can be seen that the *OPT-1* option of the proposed method achieves higher average V-PSNR values than the *OPT-2* option for Video #4. This result is similar to that of the head trace #2 of Video #1. Again, that the reason is that the head movement trace of Video #4 contains many changes in viewing directions that results in high viewport estimation errors.



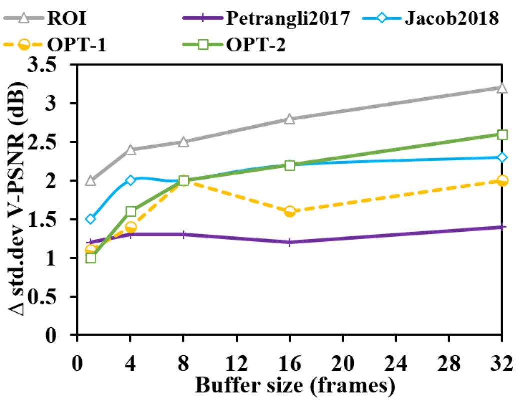
(A) Trace #1, Δ*V-PSNR*$_{avg}$.



(B) Trace #1, Δ*V-PSNR*$_{std}$.



(C) Trace #2, Δ*V-PSNR*$_{avg}$.

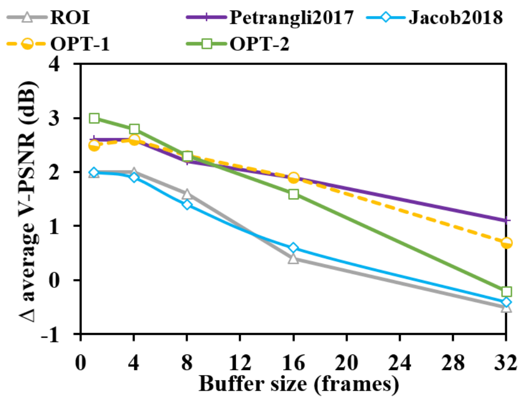


(D) Trace #2, Δ*V-PSNR*$_{std}$.

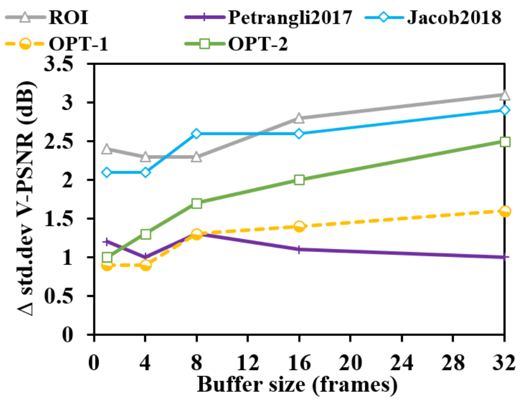
FIGURE 3.13: The impacts of the buffer size and the segment duration when $L = 16$ frames (Video #1).

3.5.5 Impacts of Segment Duration and Buffer Size

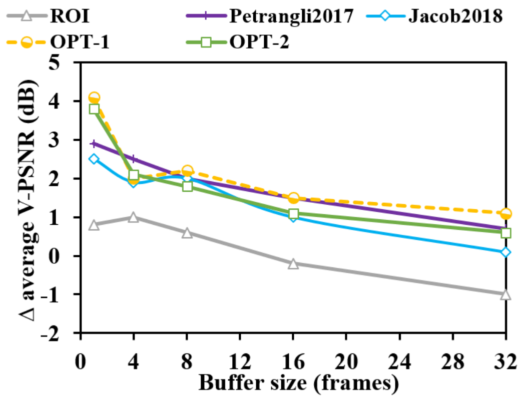
In this part, the performances of the proposed, *Petrangli2017*, *Jacob2018* and *ROI* methods under the different settings of segment durations and buffer sizes will be investigated. For that purpose, 3 segment durations of 4, 16, and 32 frames, and 5 buffer sizes of 1, 4, 8, 16, and 32 frames are considered. Note that, the *EQUAL* method is not affected by viewport estimation errors. Thus, its performance will be independent of the segment duration and the buffer size. Again, the two head movement traces shown in Fig. 3.11 are used for the Video #1. To clearly show the results, two performance metrics namely 1) delta average V-PSNR and 2) delta std.dev of V-PSNR are used. The delta average V-PSNR, denoted by Δ*V-PSNR*$_{avg}$, is the difference in the average V-PSNR between a given option/method and the *EQUAL* method. The delta std.dev V-PSNR, denoted by Δ*V-PSNR*$_{std}$, is the difference in the std.dev V-PSNR between a given option/method and the *EQUAL* method.



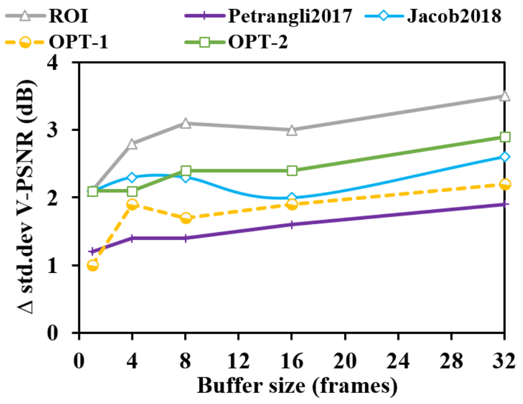
(A) Trace #1, Δ*V-PSNR*$_{avg}$.



(B) Trace #1, Δ*V-PSNR*$_{std}$.



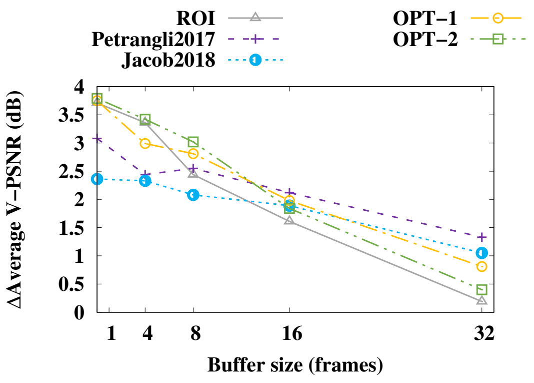
(C) Trace #2, Δ*V-PSNR*$_{avg}$.



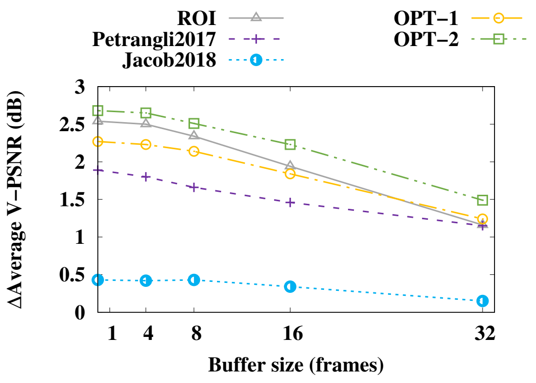
(D) Trace #2, Δ*V-PSNR*$_{std}$.

FIGURE 3.14: The impacts of the buffer size and the segment duration when $L = 32$ frames (Video #1).

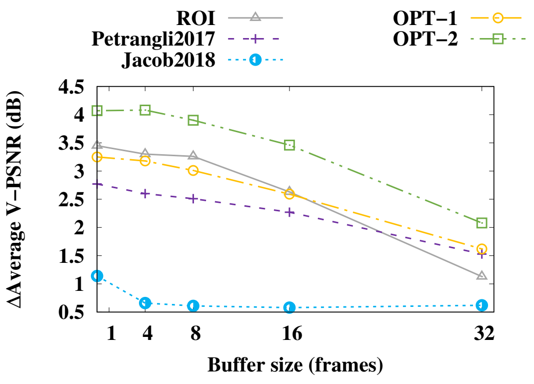
Fig. 3.12, Fig. 3.13, and Fig. 3.14 show the Δ*V-PSNR*$_{avg}$ and Δ*V-PSNR*$_{std}$ values at different settings of segment durations and buffer sizes of the proposed and *ROI* methods when the bandwidth is 8 Mbps under the two head movement traces. It can be seen that, given a segment duration, the higher the buffer size is, the lower the Δ*V-PSNR*$_{avg}$ value would become. Also, the Δ*V-PSNR*$_{std}$ values tend to increase with the buffer size. This can be explained that, longer buffer size leads to higher delay, that in turn increases the viewport estimation errors. It can also be noted that, the *ROI* method experiences the strongest quality degradation as the buffer size increases. For $L = 16$ and $L = 32$ cases, the Δ*V-PSNR*$_{avg}$ values of the *ROI* method become lower than zero when $B = 32$ frames. Hence, the *ROI* method is likely worse than the *EQUAL* method as it has much higher variations as shown in Fig. 3.13b, Fig. 3.13d, Fig. 3.14b, and Fig. 3.14d. Meanwhile, the two options of the proposed method can still achieve higher average *V-PSNR* values than those of the *EQUAL* method (i.e., Fig. 3.13a and Fig. 3.14a)



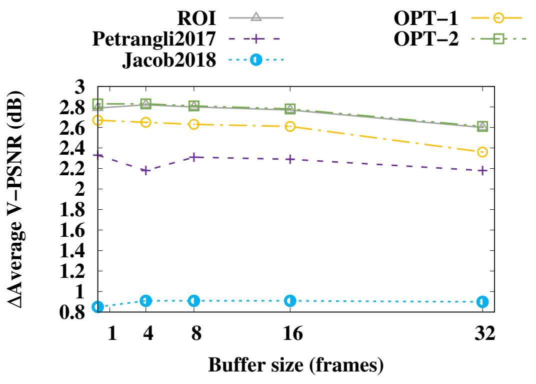
(A) Video #2.



(B) Video #3.



(C) Video #4.

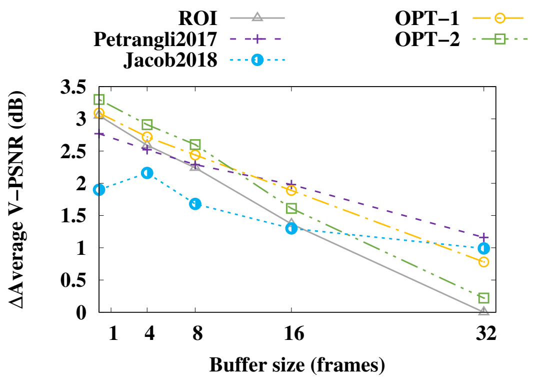


(D) Video #5.

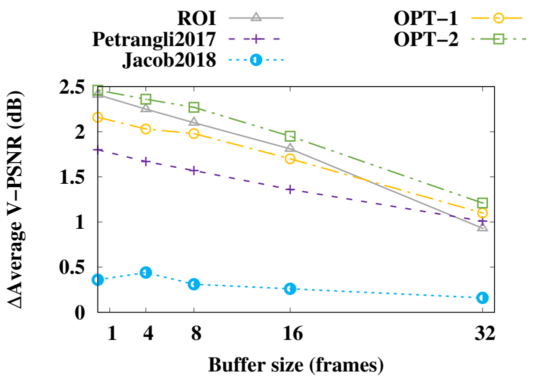
FIGURE 3.15: The impacts of the buffer size and the segment duration when $L = 4$ frames of Videos #2, #3, #4, and #5.

The results of videos #2, #3, #4, and #5 are shown in Fig. 3.15, Fig. 3.16, and Fig. 3.17. Similar to the Video #1, the trend is that the higher the buffer size is, the lower the $\Delta V\text{-}PSNR_{avg}$ becomes. It can be seen that when the buffer size increases the $\Delta V\text{-}PSNR_{avg}$ reduces quickly in case of Video #2. This is because the user frequently changes the viewing direction while watching this video. This makes the estimation errors increase rapidly when the delay (buffer size) increases. On the other hand, the $\Delta V\text{-}PSNR_{avg}$ decreases very slowly in case of Video #5. The reason is that the estimation errors are small across different buffer sizes, thanks to the stable viewport potions during the head movement trace. It can also be seen that the proposed method outperforms the reference methods when the delay increases for most of the cases.

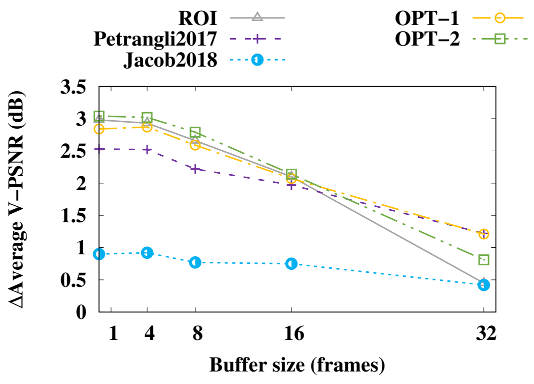
In this study, a server-based scenario in which the bitrate and quality information of all tile versions are available to the decision engine beforehand is considered. In HAS paradigm, the decision engine resides at the client. At the beginning of a streaming session, the client retrieves a metadata containing tile description information from the server. Thus, the proposed algorithm can work seamlessly in HAS paradigm if the bitrate and quality information of



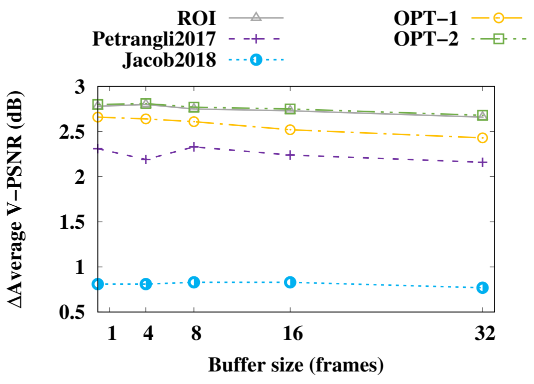
(A) Video #2.



(B) Video #3.



(C) Video #4.



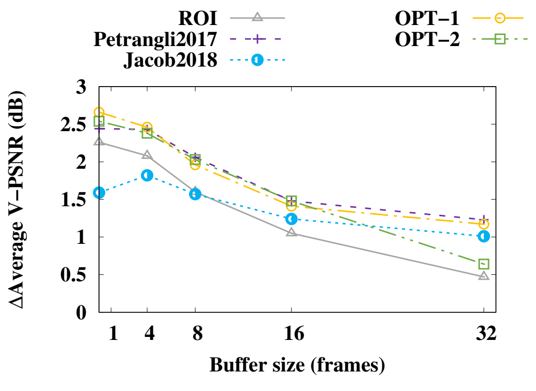
(D) Video #5.

FIGURE 3.16: The impacts of the buffer size and the segment duration when $L = 16$ frames of Videos #2, #3, #4, and #5.

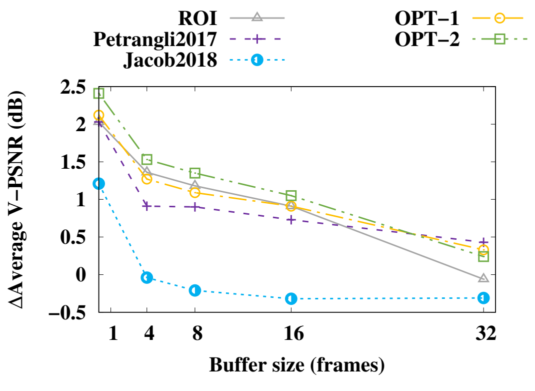
all tiles versions is provided in the metadata. In [77], the authors propose to include bitrate and quality information of tiles' versions in the metadata. Otherwise, the bitrate and quality information can be estimated at the client as proposed in [50]. In this case, the models for bitrate and quality estimation must be sent to the client in advance, e.g., in the metadata.

3.6 Summary

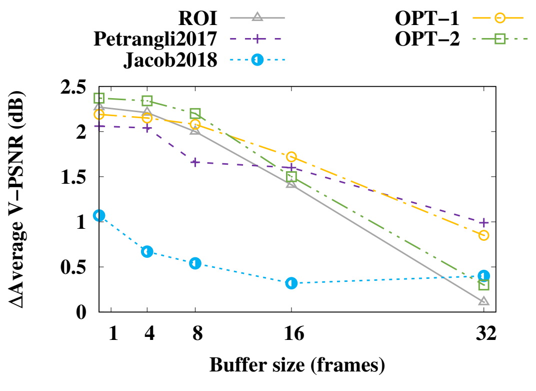
In this chapter, a low-delay system for viewport-adaptive streaming of 360-degree videos is proposed. Our proposed approach decides the versions of tiles based on estimation errors and user head movements during each segment duration. Though experiments, it is found that the proposed approach can provide not only high but also stable quality. Specifically, the proposed approach can improve the average viewport quality by up to 3.8 dB while reducing the standard deviation by up to 1.1 dB compared to the two reference approaches when the segment duration is 32 frames. The impacts of the segment duration and buffer size on the



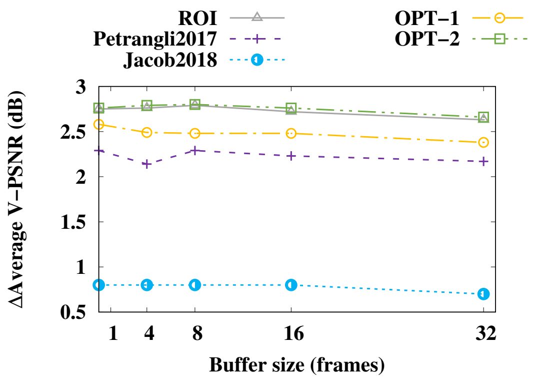
(A) Video #2.



(B) Video #3.



(C) Video #4.



(D) Video #5.

FIGURE 3.17: The impacts of the buffer size and the segment duration when $L = 32$ frames of Videos #2, #3, #4, and #5.

performance of the proposed approach is also investigated. It is found that the performance of the proposed approach decreases as the segment duration and the buffer size increases.

Chapter 4

Client-based Adaptation Framework

4.1 Introduction

For client-based adaptation, the existing HAS standards provide little information to optimize the adaptation of 360 videos. For example in MPEG DASH standard, the quality of a version is provided as a rank/order, not an actual quality value. Also, as discussed in our previous work [37], the current HAS standards define the bitrate of a version as the maximum bitrate of all segments (intervals) in that version. That is the reason why instant bitrate estimation is shown to be useful to improve adaptation performance in [50].

As for evaluation, most of the previous studies use bandwidth saving as the key performance metric [49]. However, the bandwidth saving is not able to reflect the quality seen by users. In [44], average viewport PSNR is used to demonstrate the effectiveness of some viewport-adaptive strategies compared to a non-viewport-adaptive strategy. However, it is not possible to see how the video quality changes throughout a streaming session with the average PSNR. Note that the system delay is not described in these studies.

For client-based viewport-adaptive streaming, the key research issues in our opinion are as follows.

- How to represent the bitrate information of tiles.
- How to represent the quality information of tiles.
- How to make decisions on tiles' quality levels based on the obtained bitrate and quality information.
- How to evaluate the performance of a streaming method.

In this chapter, a client-based framework for 360 video streaming is proposed. To the best of our knowledge, this is the first work that addresses all the above key issues in client-based viewport-adaptive streaming. The basic option of the framework is compatible with the existing metadata of HAS, which is used by a client to make decisions on tiles' quality levels. Moreover, our framework supports some advanced features. First, instant values of bitrate and quality of each tile's versions could be estimated and used in a similar manner to [50]. Second, a set of adaptation methods (and their options) is defined to support different usage scenarios. Then, experiments are carried out with an actual test-bed to measure viewport PSNR values over time, which has not been provided in most previous studies. The experimental results show that, using either 1) estimated bitrate or 2) both estimated bitrate and estimated quality, the adaptation performance could be significantly improved. In fact, the performance is nearly the same as the case of knowing all bitrates and quality values in advance. Moreover, the impacts of buffering delay and projection formats in 360 video streaming are also investigated.

It should be noted that the bitrate estimation related aspect in this work is different from that of [50] in some points. First there are many tiles in a 360 video, while there is only one "tile" in [47]. Second, the use of bitrate in the adaptation problem of this work is new. And finally the coding format considered here is HEVC rather than AVC [50].

The rest of this chapter is organized as follows. Related work is given in Section 4.2. Section 4.3 presents the proposed adaptation framework in detail. The evaluation and its results are described in Section 4.4. Finally, the chapter is summarized in Section 4.5.

4.2 Related Work

In [49], key issues and potential solutions for tile-based viewport-adaptive streaming of 360 videos over cellular networks are discussed. To implement viewport-adaptive streaming, most previous studies use the so-called tiling-based approach in which a 360 video is spatially divided into non-overlapping tiles [37, 44]. Each tile is then encoded into different versions corresponding to different quality levels. Given a user viewport position, high quality versions will be selected for tiles overlapping the viewport while lower quality versions will be selected for the other tiles. In [43], a 360 video is divided into six parts, each is encoded into 4 versions corresponding to four different resolutions. In [44], an end-to-end tiling-based viewport-adaptive streaming system of 360 videos over HTTP is presented. For content preparation, the authors suggest that 6x4 tiling scheme should be used for the best tradeoff between encoding overhead and adaptability. Yet, other studies employ different tiling schemes such as 8x4 [49], 8x8 [37], and 12x6 [52]. Thus, it is still unclear which tiling scheme is the best one. In [45], an HEVC-compliant tile encoding method for 360 videos is proposed, where the main goal is to support a single decoder of thin clients. In [76], an ROI-like delivery mechanism is presented for 360 videos. This study focuses on server-based approaches and employs JPEG image format for content coding.

In [54], the authors propose to encode an entire 360 video into different versions, each corresponds to a specific viewing direction. For each version, the video parts corresponding to its associated viewing direction are encoded at high quality while the remaining parts are encoded at lower quality. In adaptation, the version that is the closest to the current viewing direction is selected for transmission.

As 360 videos are delivered to the user over the Internet, there is a delay from when the system makes decisions until the viewport is displayed on a user device. Hence viewport-adaptive streaming systems have to estimate future viewport positions. In [49], the authors investigate three viewport estimation methods and find that future viewport positions in near-term (i.e., less than 1sec) could be effectively estimated with accuracy of more than 90%. A neuron network-based viewport estimation method is proposed in [126]. This method takes into account not only viewer orientations but also video characteristics when estimating future viewport positions. Experiment results show that their method can achieve prediction accuracy of more than 80%.

For content delivery, most of the existing studies use HTTP as the transport protocol [44]. In [37], a server-based method for viewport-adaptive streaming that employs push protocols of RTP/UDP has been proposed.

As conventional video encoders cannot encode 360 videos directly, 360 videos must be converted into the planar type using some projection formats such as Cubemap, Equirectangular, and Pyramid [116]. Some previous studies have compared performance of different projection formats for 360 videos. In [117], it is found that Equal-area projection format yields around 8.3% bitrate saving relative to Equirectangular projection format. In [118], the authors show that Cubemap and Equirectangular outperform Pyramidal projection format. The work in [54] reveals that Cubemap projection format is the best choice for representing 360 videos. Nowadays, Equirectangular and Cubemap projection formats are the most popular in practice.

In most previous studies, bandwidth saving is used as the key performance metric for viewport-adaptive streaming systems. Nevertheless, the bandwidth saving is not able to reflect the video quality perceived by users. [44] is the first study that uses viewport PSNR to compare the video quality of various delivery strategies. Yet, only average viewport PSNR values are presented. Also, in these studies, it is not clear 1) which type of bitrate (e.g. average or maximum) is used in making decisions and 2) how buffering delay is set. In the proposed system, for performance evaluation, the instant PSNR value of the viewport is measured for every video frame. This is the actual quality perceived by the user over time, which has not been provided in previous studies.

Compared to the previous studies, the contributions of this chapter are as follows.

- A client-based adaptation framework for viewport-adaptive streaming that can support different application scenarios, of which the basic scenario (i.e. using maximum bitrates and no quality information) is similar to the existing setting of HAS, is proposed.
- Instant bitrate of a version could be estimated by the client and then used instead of the maximum bitrate specified by the server.
- Version instant quality could be estimated and used together with estimated bitrates.
- A set of adaptation methods and their various options are presented and evaluated to show the benefits of bitrate/quality estimation.
- Experiments are carried out with an actual low-delay test-bed and results are shown by viewport PSNR values of all frames in streaming sessions. This provides insights into the actual behaviors of adaptation methods.
- Especially, the impacts of some VR-specific settings such as buffering delay and projection formats (Cubemap and Equirectangular) are investigated. Such an investigation has never been done in previous work.

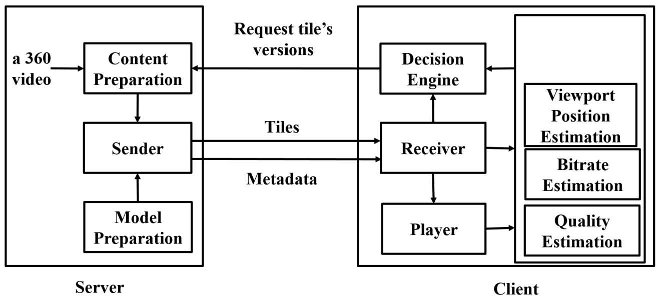


FIGURE 4.1: Tile-based viewport-adaptive streaming architecture.

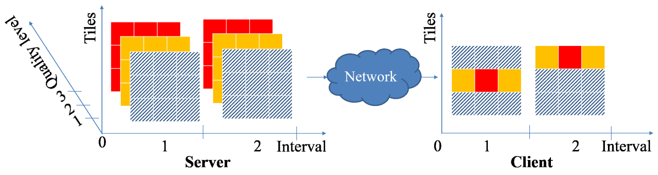


FIGURE 4.2: An example of the viewport-adaptive streaming.

4.3 Client-based Adaptation Framework for Viewport-adaptive Streaming

4.3.1 Overview

Figure 4.1 shows the general architecture of the proposed client-based framework for viewport-adaptive streaming. The server includes a content preparation module, a sender module, and a model preparation module. At the content preparation module of the server, a 360 video is first converted into a planar format using a mapping method such as Equirectangular map and Cube map [116]. Then, the converted video in the planar format is spatially divided into multiple non-overlapping *tiles*. Each tile, which is equivalent to a video in HAS, is further encoded at multiple quality versions. In practice, each version is divided, either physically or logically, into short temporal intervals. In HAS, an interval is called a segment [47]. This is also called an adaptation interval as an adaptation decision is made for a whole interval. The model preparation module is responsible for calculating the bitrate and quality estimation models which are used for decision making at the client. The sender is responsible for delivering 1) the tiles' versions requested by the client and 2) the metadata containing tile description information and the bitrate/quality estimation models to the client. In the proposed system, the tile description structure has three levels. The first level includes the basic information of the 360 video, such as projection format and the spatial size of the planar format. The

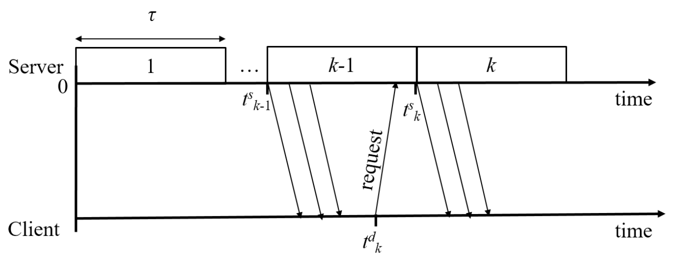


FIGURE 4.3: System data flows.

second level describes the tile arrangement, such as location and size of each tile. The third level describes the characteristics, such as may look different and coding parameters, of each tile version. In general, the first and second levels can be described by MPEG-DASH Spatial Relation Description (SRD) [133], and the third level can be described by the MPEG-DASH Media Presentation Description (MPD) [47]. In addition, the bitrate and quality estimation models are included in the first description level, and full information of bitrates and quality values (the third option in the following) are included in the third level. However, these special data are represented by our proprietary metadata.

The client includes a decision engine, a receiver, a player, and four estimation modules. At the client, the available throughput and the user's viewing direction (or viewport position) are estimated for bitrate adaptation by the throughput and viewport position estimation modules, respectively. Tiles' information (such as bitrate and quality of the versions) is either provided by the server via the metadata or estimated by the bitrate and quality estimation modules of the client. Note that, the bitrate/quality estimation modules obtain information about the bitrate/quality models from the metadata sent by the server. Based on the information of throughput, viewing direction, and tiles' bitrates and quality levels, the decision engine of the client will decide the most appropriate version for each tile in the next adaptation interval. Then, the client will send a request for the selected tiles to the server. Upon receiving a request, the sender module of the server sends the corresponding tiles to the receiver module of the client. The receiver module receives the tiles' versions from the server and forwards them to the player. The player module decodes the received tiles and then reconstructs the 360 video. According to the current viewing direction, a viewport is extracted and displayed to the user.

In video streaming, buffering delay is a big problem as analyzed in [134]. To avoid this big delay, currently the framework is designed for low-delay mode. Specifically, the media data is sent on frame basis, where a frame contains data of all component tiles. Also, the client will start playout as soon as it has completely received the first frame from the server. For content delivery, the proposed system uses two separate connections, one for signaling metadata and one for video data.

Fig. 4.2 shows an example of the viewport-adaptive streaming. At the server, the planar format of a 360 video is spatially divided into 9 (i.e., 3x3) tiles. Each tile is encoded at three quality versions, where version 3 (red color) means the highest quality. At the client, the quality versions of tiles are adaptively decided. In this example, the highest-quality tile is at the center of the frame in the first interval, and then at the top of the frame in the second interval.

In our framework, the bitrate information of a version can be either

1. only the maximum bitrate (as in HAS)
2. instantly estimated by the client, or
3. a full bitrate set of all intervals, which is provided by the server in advance.

Also, the quality information of a version can be either

1. unavailable (as in HAS),
2. instantly estimated by the client, or
3. a full set of quality values for all intervals, provided by the server.

Note that the case of having full information about bitrate/quality needs very complex metadata sent from the server to the client, so it is usually not enabled in client-based approaches. In fact, this case can be considered as the benchmark for the other cases. Interestingly, as shown in the experiments, the performance with estimated bitrate/quality is nearly the same as the performance with full information of bitrate/quality.

In the next subsections, the details of the proposed framework will be presented. Specifically, the following items will be described:

- A problem formulation for determining tiles' versions in the decision engine.
- Estimation of bitrate and quality of versions of tiles.
- Methods for deciding the optimal version of each tile.

TABLE 4.1: Notations and definitions.

Notation	Definition
V	The number of available tile versions
M	The number of tiles
L	The number of frames in an adaptation interval
τ	The duration of an interval
S_{VP}	The number of pixels in the viewport
N_m	The number of pixels in the viewport sampled from tile m
t_k^d	The time when the client makes decision for interval k
t_k^s	The time when the server starts to transmit interval k
P^e	The estimated viewport position for the next interval
$P(t_k^d)$	The viewport position at the time of making decision t_k^d
$v_{k,m}^s$	The index of the version which is chosen for tile m of interval k
Err_{VP}^{cur}	The viewport position estimation error
Err_{VP}^{thr}	The threshold of position estimation error
FoV_{ori}^v	The actual vertical Field of View of a user
FoV_{ori}^h	The actual horizontal Field of View of a user
FoV^v	The vertical Field of View used in adaptation algorithm to avoid viewport estimation errors
$R^{max}(m,v)$	The maximum bitrate of version v of tile m for all intervals.

4.3.2 Adaptation problem formulation

Fig. 4.3 shows the data flows in the proposed system. The server sends sequentially video frames to the client. The video is adapted every adaptation interval that consists of L frames corresponding to τ seconds of video.

The data of the k^{th} interval $(k \geq 1)$ is transmitted at time t_k^s given by

$$t_k^s = (k-1) \times \tau. \tag{4.1}$$

Suppose that the client needs to make decisions for next adaptation interval $k+1$ to meet a bitrate constraint R^c. To ensure that the decision of tile versions is available before the server starts sending the video data of interval $k+1$, the client should make a decision at time t_{k+1}^d given by

$$t_{k+1}^d = t_{k+1}^s - \delta t. \tag{4.2}$$

The value of δt will depend on 1) the time required to make decisions on tiles' versions and 2) the time required to send the decisions to the server.

Suppose that there are totally M tiles. Each tile is available in V versions, where version 1 (V) has the lowest (highest) quality. Denote $R(k+1,m,v)$ and $D(k+1,m,v)$ the bitrate and the distortion of version v of tile m of interval $k+1$, with $(1 \leq v \leq V, 1 \leq m \leq M)$. Also denote $v^s_{k+1,m}$ the selected version for tile m for the next interval. The version selection problem can be formulated as an optimization problem as follows.

Find the set $\{v^s_{k+1,m} | 1 \leq m \leq M\}$ *to minimize the overall distortion* VD *which is a function of* $\{D(k+1,m,v^s_{k+1,m})\}$ *while satisfying the bitrate constraint*

$$\sum_{m=1}^{M} R(k+1,m,v^s_{k+1,m}) \leq R^c. \tag{4.3}$$

To solve this problem, the system should estimate the throughput and viewing direction. The throughput measure is computed as the ratio of a given segment's data size over the transport duration of that segment [6]. In addition, the bitrate and quality of tiles'versions are required. Existing studies estimate only future viewport positions and network throughputs, while the bitrate and quality information of tiles' versions is available in advance. As our focus is on adaptation methods with bitrate and quality estimations, currently the estimated throughput and viewing direction are simply based on the latest measurements during a session to avoid making the problem complex. Anyway, the use of last measurements as the estimates of throughput and viewport position have been shown to be effective in [37, 135]. Advanced viewport position and throughput estimation methods will be investigated in our future work.

Specifically, the estimated throughput $T^e(k+1)$ for the $(k+1)^{th}$ adaptation interval is simply set to the measured throughput of the latest τ seconds of the receiver. The throughput measure is computed as the ratio of a given segment's data size over the transport duration of that segment [47]. The bitrate constraint R^c is then computed from $T^e(k+1)$ by

$$R^c = (1-\alpha) \times T^e(k+1), \tag{4.4}$$

where α is a safety margin that takes a value in the range $[0, 0.5]$ [130].

Also, the viewport position for the next interval P^e is simply estimated using the current viewport position $P(t^d_{k+1})$ at the time of making decision t^d_{k+1}. Advanced viewport position and throughput estimations will be investigated in our future work.

4.3.3 Estimations of tiles' bitrate and quality

In this subsection, methods to estimate bitrates and quality values of versions of tiles will be presented, when such information for all tiles is not sent to the client. In the literature, there exists several metrics for video quality such as SSIM and PSNR. According to our recent work [96], SSIM is shown to be not the best for assessing quality of 360 video. In contrast, PSNR achieves a little better performance than SSIM. Also, PSNR is adopted as the evaluation metric in JVET's common test conditions for 360 videos [136]. Thus, the quality metric in this work is PSNR. Let $PSNR(k,m,v)$ denote the PSNR of version v of tile m of interval k. It is assumed that that each tile version is encoded using a quantization parameter (QP) value [25]. Also, the information of bitrate/quality is included in the header of every media interval of each version. Within each interval, given a received version, this quality/bitrate information will be used to estimate the quality/bitrate of any other versions.

The estimations of bitrate and quality are divided into two parts, namely intra-interval estimation and inter-interval estimation. The former means estimating the bitrates and quality values in the same interval across different versions of a tile. The later indicates estimating the bitrates and quality values of a tile in the next interval within the same version. The estimation models presented below are specific to HEVC (High Efficient Video Coding) [51] that is used in our streaming system. Although HEVC is more complex than its predecessor such as AVC (Advanced Video Coding), HEVC has the highest coding efficiency needed to reduce the bitrates of 360-degree videos. In addition, the computing power of consumer devices is increasing rapidly. Thus, HEVC is expected to be widely supported in the near future [51].

In video encoding, the bitrate and PSNR measure of an encoded video stream are well related to the quantization parameter QP. Regarding the bitrate model, it is well-known that encoded video bitrate is roughly halved as QP increases by 6-unit [50]. In this part, a generalized form of the model in the previous work [50] is used. The normalized bitrate of version v of tile m is estimated using the following model.

$$\frac{R(k,m,v)}{R(k,m,V)} = a_1 \times 2^{b_1 \times (\frac{QP_v - QP_V}{6})} + c_1, \tag{4.5}$$

where QP_v denotes the QP of version v $(1 \le v \le V)$, and a_1, b_1, c_1 are model parameters.

When receiving the tile with selected version $v^s_{k,m}$, the client knows the bitrate $R(k,m,v^s_{k,m})$, so the estimated bitrates of other versions of tile m can be computed from Eq. (4.5) as follows.

$$R^e(k,m,v) = R(k,m,v^s_{k,m}) \times \frac{a_1 \times 2^{b_1 \times (\frac{QP_v - QP_V}{6})} + c_1}{a_1 \times 2^{b_1 \times (\frac{QP_{v^s_{k,m}} - QP_V}{6})} + c_1}. \tag{4.6}$$

Similar to the bitrate, the PSNR values of versions of a tile are estimated as follows. First, the normalized PSNR value of a version is estimated using the following model.

$$\frac{PSNR(k,m,v)}{PSNR(k,m,V)} = a_2 \times \frac{QP_v}{QP_V} + b_2. \tag{4.7}$$

where a_2 and b_2 are model parameters.

The estimated PSNR value of version v of tile m is therefore given by

$$PSNR^e(k,m,v) = PSNR(k,m,v^s_{k,m}) \times \frac{a_2 \times \frac{QP_v}{QP_V} + b_2}{a_2 \times \frac{QP_{v^s_{k,m}}}{QP_V} + b_2}. \tag{4.8}$$

The model parameters (a_1,b_1,c_1,a_2,b_2) are obtained by minimizing the squared error between the modeled values and measured values. Some other functional forms, including linear, logarithmic, power, and exponential functions are also tried. It was found that the above functions yielded the least fitting error. Especially, the bitrate and QP values of the tiles' versions are used to obtain the parameters of the bitrate model. As for the PSNR estimation model, the PSNR and QP values of the tiles' versions are used. In practice, as bitrate and quality information is available after video encoding, these two models can be obtained at the server for each video in advance and then sent to any client requesting that video.

Because two adjacent intervals usually have similar characteristics, the bitrate/quality of the current interval can be used as the estimated bitrate/quality of the next interval as follows.

$$R^e(k+1,m,v) = \begin{cases} R(k,m,v) & \text{if } v = v^s_{k,m} \\ R^e(k,m,v) & \text{if } v \neq v^s_{k,m} \end{cases}. \tag{4.9}$$

$$PSNR^e(k+1,m,v) = \begin{cases} PSNR(k,m,v) & \text{if } v = v^s_{k,m} \\ PSNR^e(k,m,v) & \text{if } v \neq v^s_{k,m} \end{cases}. \tag{4.10}$$

In a long video, scenes may have very different characteristics over time. So the model parameters may need to be recomputed and updated to the client accordingly. Obviously, the update mechanism should depend on the availability of the bitrate and quality of tile versions. In case of on-demand streaming, the model parameters in different scenes (i.e., MPEG-DASH periods) can be computed in advance and updated each period. As for live streaming, using some methods such as machine learning, the model parameters for a period could be calculated on the fly and sent to the client during a streaming session. In our current system, the estimation models are updated as in the on-demand case. Detailed mechanisms for updating the estimation models are reserved for our future work.

4.3.4 Version Selection Methods

Given a specific viewport position, the objective of a version selection method is to select the version of each tile so as to maximize the quality for the user. It can be seen that the proposed framework is rather complicated with various options of bitrate and quality information. So, to 1) understand clearly the benefits of bitrate and quality estimations and also 2) avoid making the investigation too complex, in our current framework, three version selection methods, called *Equal, ROI*, and *Weighted* methods, are deployed. The *Equal* and *ROI* methods are basic methods employed in previous work, and are now enhanced with bitrate estimation. The *Weighted* method is an improved ROI method when having quality information to show the benefits of both bitrate and quality estimations. Note that, the estimated information can actually be used to improve any existing or future version selection methods at the client. As the method names imply, the *Equal* method treats all tiles equally, the *ROI* method emphasizes the tiles overlapping the viewport, and the *Weighted* method assigns different weights to different tiles in the viewport. Note that the basic idea of the *ROI* method has long been used in ROI-based or panoramic image/video viewing. The new point here is the use of estimated/specified instant bitrates in making decisions.

Equal method

As mentioned, in the *Equal* method, all tiles have the same quality version. The selected version is the highest level while the sum of tiles' bitrates is still not greater than the bitrate constraint. The advantages of this method include implementation simplicity and effectiveness in coping with viewport position variations.

ROI method

In this method, visible tiles (i.e., tiles overlapping the viewport) are all assigned the same highest possible version, while the other (invisible) tiles are assigned a low quality level which is allowed by the bitrate constraint. This method is investigated in various studies such as [43,44]. An implementation of this version selection method can be specified by two general steps as follows.

- *Step 1:* The version of invisible tiles is set to 1. Find the highest quality version for all visible tiles, while still satisfying the bitrate constraint.
- *Step 2:* Find the highest quality version for all invisible tiles, while still satisfying the bitrate constraint.

Obviously, no quality information should be considered in the two methods above. In either the *Equal* method or the *ROI* method, there are three options of bitrate information for a version, namely 1) only the maximum bitrate (i.e. basic scenario), 2) estimated instant bitrate at the client, and 3) a full set of bitrates provided by the server. In the first option, the bitrate of a version in any interval is simply set to the maximum bitrate $R^{max}(m,v)$ of that version.

$$R(k,m,v) = R^{max}(m,v)\ \forall k. \tag{4.11}$$

Weighted method

It can be seen that the *Equal* method treats all tiles equally, and the *ROI* method treats visible tiles equally. However, the contribution of a tile to the viewport quality will be mainly dependent on how the tile overlaps the viewport. In particular, if only a small part of a tile is actually in the viewport, the contribution of that tile in the viewport quality is small, and so selecting a low quality level for that tile does not affect the viewport quality much. Therefore, in this method the quality level of a tile is decided according to the contribution (weight) of that tile.

In order to take into account the contribution of a tile, the objective VD for optimization is proposed as follows. Because only a portion (i.e., viewport) of the entire 360 video is actually watched by viewers, rectilinear projection is used to obtain a viewport given a viewport position (ϕ, θ) with ϕ being the latitude and θ being the longitude. By this way,

Algorithm 1: *Weighted* version selection method

```
Input: $M, S_{VP}, V, D(k,m,v^s_{k,m}), N_m, R(k,m,v^s_{k,m}), Err^{cur}_{VP}, Err^{thr}_{VP}$
Output: $v^s_{k+1,m}$
 1: Estimate $D(k+1,m,v), R(k+1,m,v)$ using Eqs.(4.6), (4.8)~(4.10);
 2: $TR^s \leftarrow \sum_{m=1}^{M} R_{k+1,m,1}$                # Total bitrate corresponding to $v^s_{k+1,m}$
 3: $v^s_{k+1,m} \leftarrow 1 \ \forall m \in [1,M]$;
 4: $flag \leftarrow 1$;
 5: if $Err^{cur}_{VP} > Err^{thr}_{VP}$ then               # Extending the Field of View
 6:   $FoV^h \leftarrow FoV^h_{ori} + \Delta FoV$;
 7:   $FoV^v \leftarrow FoV^v_{ori} + \Delta FoV$;
 8:   Update $N_m$ based on $FoV^h$ and $FoV^v$;
 9: end if
10: while $flag = 1$ do                                  # Deciding tiles' selected versions.
11:   Initialize: $\hat{D}^{opt}$, $m^{opt}$, $TR^{opt}$, $flag \leftarrow 0$
12:   for $m = 1$ to $M$ do
13:     if $v^s_{k+1,m} + 1 \leq V$ then
14:       Calculate $\Delta D$, $\Delta R$, and $\hat{D}$ using Eqs.(4.13)(4.14)(4.15);
15:       if $\hat{D} \geq \hat{D}^{opt}$ and $\Delta R \leq R^c - TR^s$ then
16:         $\hat{D}^{opt} \leftarrow \hat{D}$;
17:         $TR^{opt} \leftarrow TR^s + \Delta R$;
18:         $m^{opt} \leftarrow m$;
19:         $flag \leftarrow 1$;
20:       end if
21:     end if
22:   end for
23:   if $flag = 1$ then
24:     $v^s_{k+1,m^{opt}} \leftarrow v^s_{k+1,m^{opt}} + 1$;
25:     $TR^s \leftarrow TR^{opt}$;
26:   end if
27: end while
```

the number of pixels $\{N_m | 1 \leq m \leq M\}$ in the viewport which are sampled from tile m can be determined. N_m is then used as the weight of tile m in the overall distortion. Note that the sum of $\sum_{m=1}^{M} N_m$ is equal to the viewport's total number of pixels $S_{VP} = W_{VP} \times H_{VP}$ with H_{VP} being the viewport's height and W_{VP} being the viewport's width.

Given $v^s_{k+1,m}$ the version selected for tile m of interval $k+1$, the overall distortion VD at interval $k+1$ is calculated by

$$VD = \frac{\sum_{m=1}^{M} D(k+1,m,v^s_{k+1,m}) \times N_m}{S_{VP}}. \tag{4.12}$$

So, this version selection method can be based on the optimization problem of Subsection 4.3.2. In each interval, the distortion value of a version $D(k+1,m,v^s_{k+1,m})$ is the MSE value, which is obtained from the estimated or received PSNR value of that version.

When there are too many combinations of versions and tiles, a full search to find optimal versions for all tiles could be time-consuming. To deal with this, an efficient loop of determining optimal versions is conducted as follows.

Assume that the version $v_{k+1,m}$ of tile m increases by 1 each time, then the overall distortion value consequently decreases by

$$\Delta D = \frac{(D(k+1,m,v_{k+1,m}) - D(k+1,m,v_{k+1,m}+1)) \times N_m}{S_{VP}}. \tag{4.13}$$

Correspondingly, the total bitrate of all tiles is increased by

$$\Delta R = R(k+1,m,v_{k+1,m}+1) - R(k+1,m,v_{k+1,m}). \tag{4.14}$$

Denote by

$$\hat{D} = \frac{\Delta D}{\Delta R} \tag{4.15}$$

the decrease degree of the overall distortion value per a bitrate unit. Hence, to minimize the overall distortion while meeting the bitrate constraint, the optimal version increase should have the highest $\hat{D}$ and meet the bitrate constraint. The process to determining optimal versions is repeated until it is impossible to find any version increase for all tiles while meeting the bitrate constraint.

As user head may be continously moving, the predicted viewport positions may not be the same as the actual viewport position, resulting in prediction errors. To cope with this problem, one may extend the viewport to compensate prediction errors. Firstly, the prediction error is computed by the prediction error of the last playout frame. Assume that at the time of making decision t^d_{k+1}, the last playout frame is the l^{th} frame, and the estimated and actual corresponding viewport positions are $P^e_l = (\phi^e_l, \theta^e_l)$ and $P^a_l = (\phi^a_l, \theta^a_l)$ respectively. The prediction error is obtained by

$$\begin{aligned} Err^{cur}_{VP} &= P^e_l - P^a_l \\ &= \sqrt{(\phi^e_l - \phi^a_l)^2 + (\theta^e_l - \theta^a_l)^2}. \end{aligned} \tag{4.16}$$

Currently, to cope with this prediction error, a simple solution is employed. Specifically, if the prediction error of the last playout frame exceeds a predefined threshold Err^{thr}_{VP}, the FoV of the viewport will be extended by ΔFoV (degree) both horizontally and vertically. It

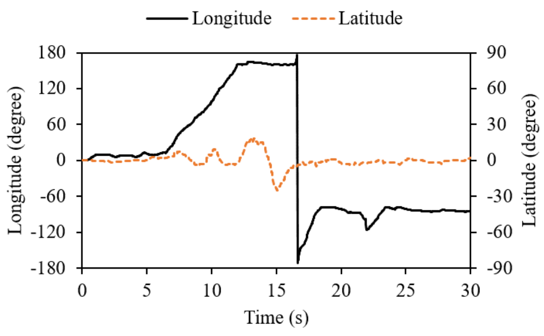


FIGURE 4.4: The used head movement trace.

should be noted that this change of FoV is just for computing the weights of tiles, not for content display. The details of the *Weighted* method are shown in Algorithm 1. Firstly, the bitrate and distortion of tile versions of the next interval are estimated using the birate and quality estimation models described in Subsection 3.3, i.e., line 1. Then, the lowest quality version is assigned for every tile and the total bitrate of all tiles is computed, i.e., lines 2-3. Next, the field of view is extended by ΔFoV if the last estimation error exceeds the threshold Err_{VP}^{thr}, i.e., lines 5-9. Finally, in lines 10-27, the version of each tile is determined as follows. The algorithm first determines the tile of which the decrease degree of the overall distortion value per a bitrate unit $\hat{D}$ is highest, i.e., lines 12-22. Then, the version of that tile is increased by one if there is enough unallocated throughput, i.e., lines 23-26. Essentially, this version selection step increases the version of the tile that has the highest contribution in the overall quality. This process is repeated over all tiles. In our test-bed, the processing time of this method is always less than 1 ms.

Finally, in this method, two options are considered. The first option, called *Weighted-full*, is that the bitrate and quality information of versions is fully available at the client. The second option, called *Weighted-estimate*, is that the bitrate and quality information of versions is estimated at the client as described in the previous subsection.

4.4 Evaluation

4.4.1 Experimental Settings

The proposed framework is implemented based on Gaming Anywhere, an open-source cloud gaming platform [129]. The client is written in C++ and running on a Ubuntu 14.04 LTS 64bit machine with 4 GB RAM, Intel Core i5-3210M CPU 2.5 GHz (4 cores). The server

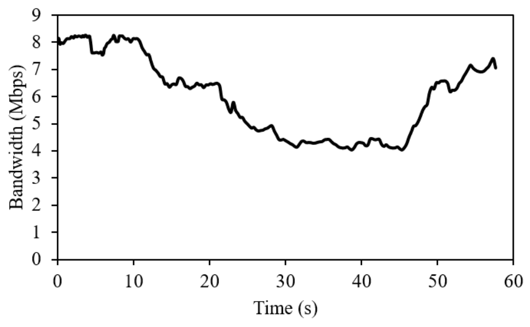


FIGURE 4.5: The time-varying bandwidth trace.

is running on another Ubuntu 14.04 LTS 64bit machine with 8 GB RAM, Intel Core i5-2500 CPU 3.3 GHz (4 cores). The client is connected to the server via a router. During a streaming session, the selected quality level of each tile and the viewport position are logged. After a streaming session finishes, video tiles are combined to reconstruct a 360 video using FFMPEG [137], then the viewport is extracted and the actual viewport PSNR is calculated. The horizontal and vertical Field of Views (FoV) (i.e., FoV^h_{ori} and FoV^v_{ori}) are both set to 90 degrees. The viewport has a resolution of 960×960. The 360Lib software [138] is used to convert the projection formats of 360 videos. An adaptation interval contains $L = 16$ (frames). The values of $\delta t, \alpha, Err^{thrr}_{VP}$ and ΔFoV are respectively set to 50 ms, 0.2, 8 degrees, and 30 degrees. Network conditions are emulated using DummyNet [131] where the round-trip time delay is set to 20 ms. A real head movement trace shown in Fig. 4.4 is used in the experiment. The head movement trace is obtained by using OpenTrack [139] to record viewport positions of a user watching the videos.

4.4.2 Estimation accuracy

In this part, the proposed bitrate and PSNR estimation models presented in Subsection 4.3.3 will be evaluated. For tiling, two popular projection formats, namely Equirectangular (ERP) and Cubemap (CUBE) are employed. In the experiment, three videos from YouTube: *Tokyo's Yakitori Alley: Shinjuku 360*[1] , *Dust Meat Bots*[2] and *Roller Coaster*[3] with different content characteristics are used. Video #1 is a 30s-long video about a reporter traveling bustling streets in Tokyo. Video #2 is a 10s-long video of the game CounterStrike. Video #3 is a 30s-long video recording a ride on a rollercoaster. These videos have a frame rate of 30 fps and a

[1]https://www.youtube.com/watch?v=pQyt6H7GlcY, 80s-110s
[2]https://www.youtube.com/watch?v=jWbLLKdO7k4, 40s-50s
[3]https://www.youtube.com/watch?v=8lsB-P8nGSM, 0s-30s

TABLE 4.2: Parameters of the bitrate estimation model (Eq.4.6).

Parameters	Video #1		Video #2		Video #3	
	CUBE	**ERP**	**CUBE**	**ERP**	**CUBE**	**ERP**
a_1	8.19	6.74	10.00	9.30	10.0	10.0
b_1	-0.18	-0.17	-0.20	-0.19	-0.13	-0.14
c_1	-0.014	0.005	-0.006	-0.014	-0.02	0.04

TABLE 4.3: Performances of the bitrate estimation model.

Metric	Video #1		Video #2		Video #3	
	CUBE	**ERP**	**CUBE**	**ERP**	**CUBE**	**ERP**
PCC	0.99	0.99	0.99	0.99	0.98	0.97
RMSE	20.02	19.19	12.57	11.73	40.7	52.9

resolution of 3840x1920 in the original ERP projection format. For ERP, each video is divided into $M = 64$ tiles (i.e., 8×8 tiles). For CUBE, each face of a video is divided into 9 tiles (i.e., 3×3 tiles), resulting in a total of $M = 54$ tiles (i.e., 6 faces × 9 tiles). Using HEVC, each tile is encoded into $V = 6$ versions corresponding to 6 QP values of 28, 32, 36, 40, 44, 48, which are similar to the QP value range used in [44]. The low-delay B profile with Group of Picture (GoP) size of 4 is used. The model parameters of bitrate and PSNR estimation models are determined by means of curve-fitting for each pair of video and projection format.

The values of model parameters and the performance of bitrate estimation model (4.6) are respectively shown in Table 4.2 and Table 4.3. Here, two performance metrics, namely Pearson Correlation Coefficient (PCC) and Root Mean Square Error (RMSE) are considered. Generally, the higher the value of PCC or the lower the value of RMSE is, the more accurate the model becomes. Fig. 4.6 shows the scatter diagram of the estimated and actual tile bitrates. As shown in Table 4.3, PCC values are very high (i.e., ≥ 0.97) for all cases. Also, RMSE values are also very small (i.e., less than 53 kbps). These results imply that the proposed model can predict well the bitrates of tiles of three considered videos with both projection formats.

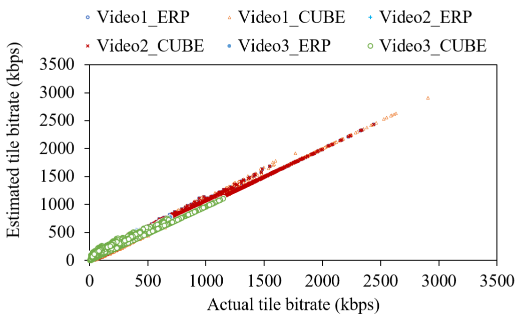


FIGURE 4.6: Scatter diagram of estimated and actual tile bitrates.

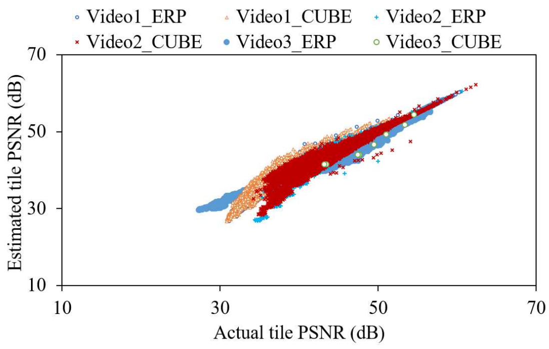


FIGURE 4.7: Scatter diagram of estimated and actual tile PSNR values.

It can be seen that though the PCC values in all cases are similar, the RMSE values of Video #1 are always higher than those of Video #2, whereas the RMSE values of Video #3 are highest. Also, the model parameters of each (video, projection format) pair are different from the others as shown in Table 4.2. This can be explained that the characteristics of a video have impacts on the prediction performance of the bitrate model. Specifically, Video #1 is a real video that has higher complexity and thus more difficult to predict than Video #2 which is a computer-generated video. In addition, Video #3 has a much higher camera motion than Video #1.

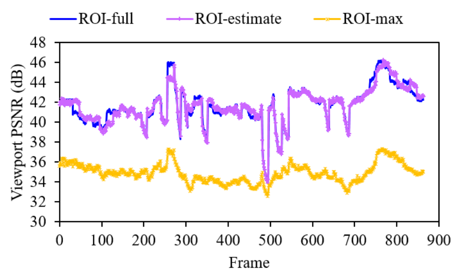
TABLE 4.4: Parameters of the PSNR estimation model (4.8).

Parameters	**Video #1**		**Video #2**		**Video #3**	
	CUBE	**ERP**	**CUBE**	**ERP**	**CUBE**	**ERP**
a_2	-0.22	-0.22	-0.17	-0.19	-0.33	-0.34
b_2	1.22	1.22	1.18	1.19	1.33	1.34

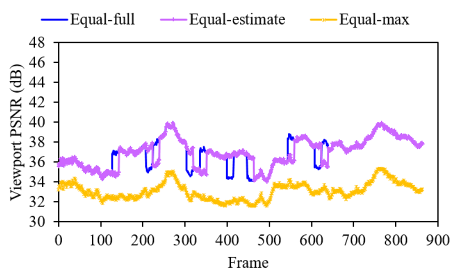
TABLE 4.5: Performances of the PSNR estimation model.

Metric	**Video #1**		**Video #2**		**Video #3**	
	CUBE	**ERP**	**CUBE**	**ERP**	**CUBE**	**ERP**
PCC	0.99	0.99	0.97	0.96	0.98	0.98
RMSE	1.23	1.42	1.12	1.45	1.5	1.37

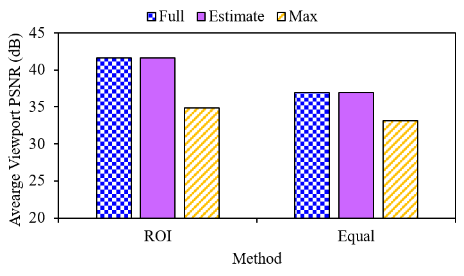
Table 4.4 and Table 4.5 respectively show the model parameters and estimation performance of the PSNR estimation model (4.8). The scatter diagram of the estimated and actual tile PSNR values is shown in Fig. 4.7. From Table 4.5, it can be seen that the proposed PSNR estimation model has very high PCC values (i.e., 0.96 or higher) and low RMSE values (i.e., 1.5 dB) for all three videos and both projection formats. This means that the PSNR values of tiles can be accurately predicted using the proposed model.



(A) ROI.



(B) Equal.



(C) Average Viewport PSNR (dB)

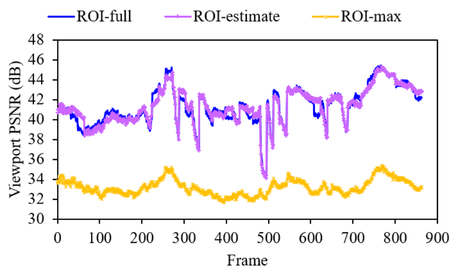
FIGURE 4.8: Comparisons of three options of the *Equal* and *ROI* methods under a constant bandwidth of 6 Mbps (CUBE).

It can also be noted that the PCC values of Video #1 are slightly higher than that of Video #2 and Video #3. And, for Video #1 and Video #2, the RMSE value of ERP is a little higher than that of CUBE. On the other hand, the RMSE of CUBE is a little higher than that of ERP in case of Video #3. From Table 4.4, it can be seen that there is only a small difference of the model parameters in the six cases. This implies that the prediction performance of the PSNR estimation model is only slightly affected by the content characteristics and projection formats.

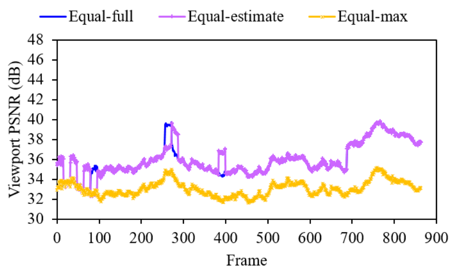
4.4.3 Benefits of bitrate estimation

In this subsection, the benefits of bitrate estimation will be investigated. For that purpose, the *Equal* and *ROI* methods are considered. From now on, only Video #1 is used to evaluate the proposed framework. The total maximum bitrates of the whole video (i.e. composed of all tiles) are shown in Table 4.6. The length of each streaming session is 54 intervals, or 864 frames. The setting of content preparation is the same as described in the previous subsection.

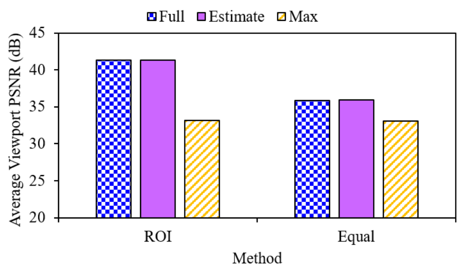
As mentioned, three options of bitrate information are considered. The first option, called *Max*, uses the maximum bitrate as the representative bitrate of a version. The second option,



(A) ROI



(B) Equal.



(C) Average Viewport PSNR (dB)

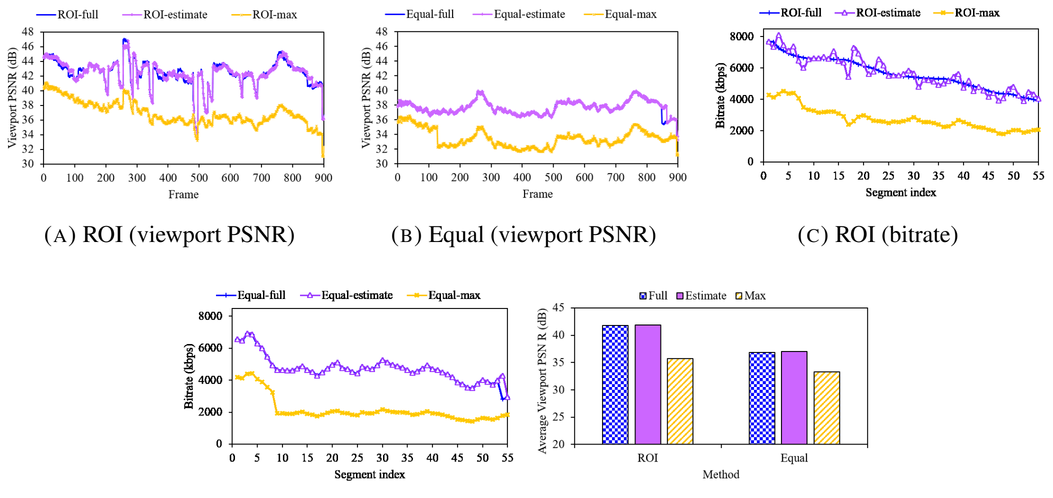
FIGURE 4.9: Comparisons of three options of the *Equal* and *ROI* methods under a constant bandwidth of 6 Mbps (ERP).

TABLE 4.6: Maximum bitrate of versions.

Version	QP	Max bitrate (kbps)	
		CUBE	ERP
#1	48	2832	3294
#2	44	3682	4220
#3	40	6034	6792
#4	36	9393	10431
#5	32	14405	15900
#6	28	22352	24346

called *Estimate*, uses estimated bitrates at the client. The third option, called *Full*, uses actual bitrates as fully provided by the server. In current client-based methods, the client currently knows only the version ranking and max bitrate of each version. Though there are a number of client-based methods (i.e., [43–45]) in the literature, they are actually the *ROI* method with the *Max* option.

To clearly explain client behavior under each of these options, the experiment is firstly run under a constant bandwidth of 6 Mbps. Fig. 4.8(a) and Fig. 4.8(b) respectively show frames' viewport PSNR values of the three options of the *ROI* and *Equal* methods when using the CUBE format. Fig. 4.8(c) compares the average viewport PSNR of the three options of each method.



(A) ROI (viewport PSNR) (B) Equal (viewport PSNR) (C) ROI (bitrate)

(D) Equal (bitrate) (E) Average Viewport PSNR (dB)

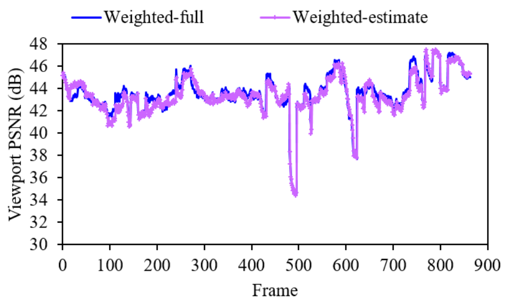
FIGURE 4.10: Comparisons of three options of the *Equal* and *ROI* methods under a real bandwidth trace (CUBE)

It can be seen that, interestingly, the viewport PSNR of the *Estimate* option is almost identical to that of the *Full* option. This implies that the proposed bitrate estimation can enable the client to behave as if it knows all information about tiles' bitrates.

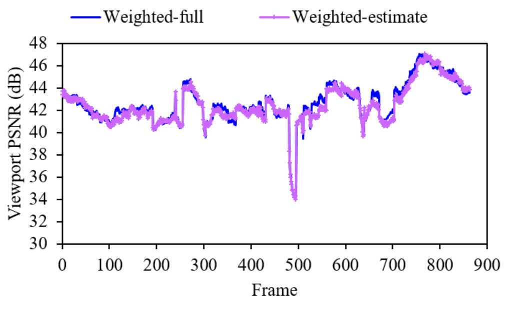
With the *Max* option, the client decides tiles' versions based on the maximum bitrates of versions. Meanwhile, when bitrate estimation is enabled (i.e., the *Estimate* option), the client can dynamically switch between versions depending on the estimated bitrate. This results in significantly higher viewport quality than that of the *Max* option. Specifically, the average viewport PSNR increases by 3.8 dB and 6.7 dB for the *Equal* and *ROI* methods, respectively.

Fig. 4.9 shows the results when using the ERP format. Similar to the results of the CUBE format, bitrate estimation can help the client improve considerably the viewport quality compared to using the maximum bitrates. In particular, the average PSNR gains for the *Equal* and *ROI* methods are respectively 2.9 dB and 8.1 dB. Also, the viewport PSNR of the *Estimate* option is very similar to that of the *Full* option. It can also be noted that the resulting viewport PSNR using ERP is similar to that using CUBE.

Next, a time-varying bandwidth trace is used to see how bitrate estimation can help the client in this case. The used bandwidth trace is shown in Fig. 4.5. As this trace's duration is 58 seconds while the obtained video is only 30 s long, the whole video is repeated in this experiment. The viewport PSNR values of the three options of the *Equal* and *ROI* methods are shown in Fig. 4.10. It can be seen that selecting tile versions based on bitrate estimation



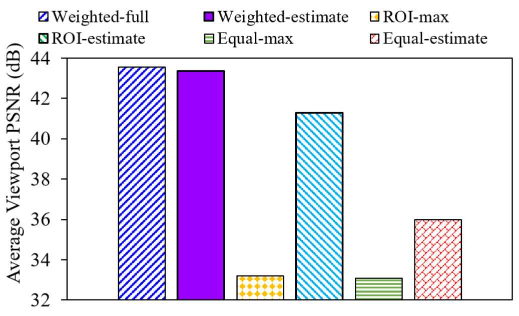
(A) ERP.



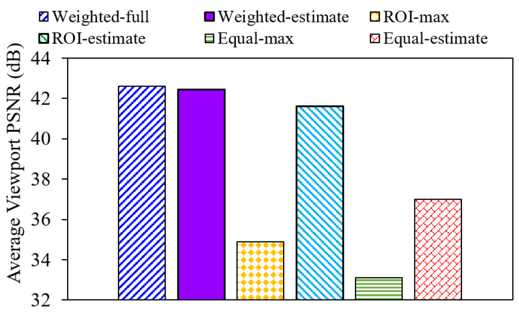
(B) CUBE.

FIGURE 4.11: Frame Viewport PSNR of two options of the *Weighted* method under a constant bandwidth.

results in similar viewport quality as that of the *Full* option. It can be noted that the *Estimate* option has a slightly higher average viewport PSNR than that of the *Full* option in case of the *Equal* method. The reason is that at the 55th segment (frames 848−863), the estimated bitrates of tiles' versions are lower than their actual values. As a result, the *Estimate* option selects version 3 for all tiles. Meanwhile, the Full option can only select version 2 for all tiles. Consequently, the viewport PSNR values of the *Estimate* option are higher than those of the Full option for frames 848−863. In addition, the *Estimate* option achieves significantly higher viewport quality compared to that of the *Max* option for both *Equal* and *ROI* methods. Specifically, the average viewport PSNR is improved by 6 dB and 3.7 dB for the *ROI* and *Equal* methods, respectively. Also, the advantage of bitrate estimation is significant across the entire range of bandwidth. As the PSNR value under time-varying bandwidth is much more fluctuating, constant bandwidth will be used for comparing adaptation methods.



(A) ERP.



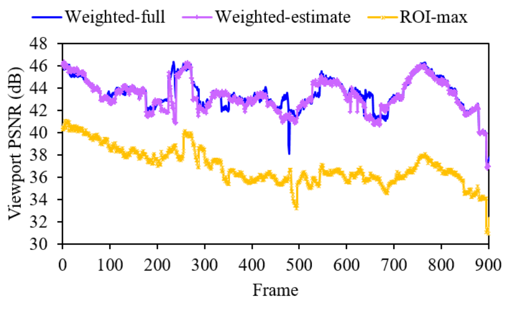
(B) CUBE.

FIGURE 4.12: Average Viewport PSNR of two options of the *Weighted* method and the *Max* option of the *ROI* and *Equal* methods under a constant bandwidth.

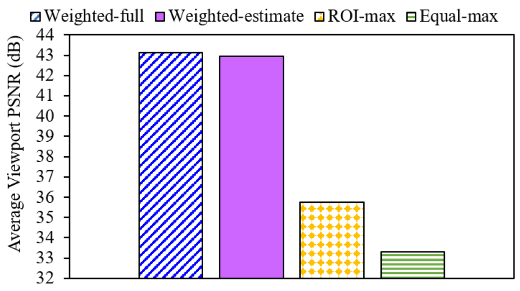
4.4.4 Benefits of both bitrate and quality estimations

In this subsection, the advantage of both bitrate and quality estimations will be investigated using the *Weighted* method, which decides tiles' versions based on bitrate and PSNR information. In server-based methods, the server has full information about quality and bitrate of each version [37]. So, though the context is different, the *Full* option is somewhat similar to server-based methods. Fig. 4.11 compares the *Estimate* and *Full* options of the *Weighted* method under a constant bandwidth of 6 Mbps for two projection formats. It can be seen that, the frame viewport PSNR of the *Estimate* option is again almost similar to that of the *Full* option. This means that the proposed bitrate and PSNR estimation models are very effective, so that the version selection method can determine tiles' versions as if it fully knows the bitrate and PSNR values of tiles.

Fig. 4.12 compares the average viewport PSNR of the two options of the *Weighted* method and the *Estimate* and *Max* options of the *ROI* and *Equal* methods under a constant bandwidth of 6 Mbps. As for the *Weighted* method, the average viewport PSNR of the *Estimate* option is only $0.1-0.2$ dB lower than that of the *Full* option. Compared to the *ROI* and *Equal* method,



(A) Frame Viewport PSNR



(B) Average Viewport PSNR

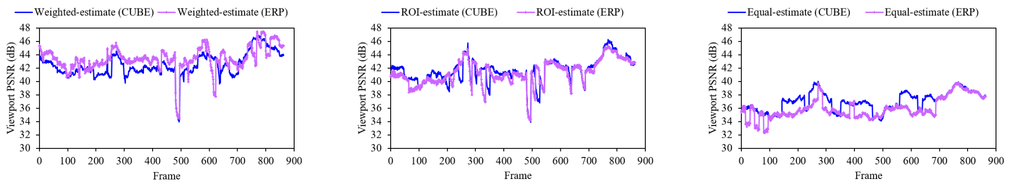
FIGURE 4.13: Viewport PSNR of two options of the *Weighted* method under a real bandwidth trace (CUBE).

the *Weighted* method with the *Estimate* option results in significantly higher average viewport PSNR. Specifically, the average viewport PSNR is respectively improved by 10.2 dB and 10.3 dB compared to the *Max* options of the *ROI* and *Equal* methods in case of ERP projection. In case of CUBE projection, the *Weighted* method increases the average viewport PSNR by 7.5 dB and 9.3 dB compared to the *Max* options of the *ROI* and *Equal* methods, respectively.

Fig. 4.13 shows the results under a real bandwidth trace using the CUBE projection. Similar to the constant bandwidth case, the proposed bitrate and quality estimation can help achieve similar viewport quality compared to the *Full* option and significantly improve viewport quality compared to the *ROI* and *Equal* methods. Specifically, the average viewport PSNR of the *Estimate* option is 0.2 dB lower than that of the *Full* option. Meanwhile, the *Weighted* method with the *Estimate* option can improve the average viewport PSNR by 7.2 dB and 9.6 dB compared to the *ROI* and *Equal* methods, respectively.

4.4.5 Discussion on system-related settings

In this subsection, the impacts of some important settings on the performance of the proposed framework will be discussed. For simplicity, only the *Estimate* option is used.



(A) Weighted method (B) ROI method. (C) Equal method

FIGURE 4.14: Comparisons of CUBE and ERP for the three methods.

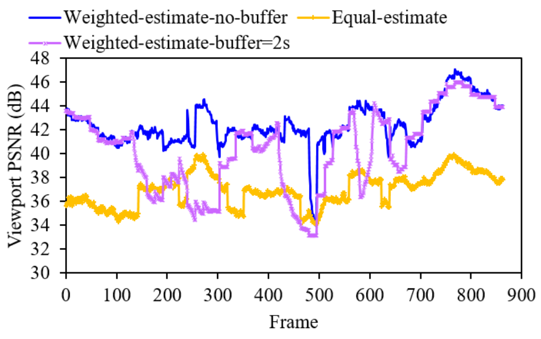


FIGURE 4.15: Impact of client buffering on the performance of the *Weighted* method under a constant bandwidth (CUBE).

Regarding the impact of projection formats, Fig. 4.14 compares the viewport PSNR values of CUBE and ERP for all three methods under the constant bandwidth of 6 Mbps. It can be seen that, with the *Weighted* method, the viewport PSNR values when using CUBE are mostly higher than that using ERP. Yet, the average difference is small (about 0.4 dB). With the *ROI* method, ERP and CUBE also perform similarly. However, with the *Equal* method, using CUBE results in significant higher viewport PSNR than using ERP. This is because CUBE always requires less bitrate than ERP does. Consequently, given the same network bandwidth, the client can select higher quality version when using CUBE. Specifically, in this example, the client selects mostly version #3 when using CUBE, while it selects mostly version #2 when using ERP.

As mentioned above, client buffering as in HAS could be a big problem in 360 video streaming. In the proposed framework, the client starts playout right after having received the first frame. For this factor, an additional experiment is carried out where the client buffer is set to 2 s. Fig. 4.15 compares the viewport PSNR of the *Weighted* method in two cases 1) without client buffering and 2) with a client buffering of 2 s. It can be seen that the viewport PSNR is drastically reduced under the presence of the client buffering, especially when the viewport position changes quickly due to user head movements. The viewport PSNR is even lower than that of the *Equal* method, which may greatly reduce the viewing experience of users.

The negative impact of client buffering can be explained as follows. The delay from when the client decides the versions of tiles until the corresponding viewport is displayed will be increased because the frames should wait (for 2 s in this experiment) in the client buffer. The longer the client buffering is, the longer the delay becomes. A long delay may lead to significant difference between the estimated and actual viewports if the user moves his/her head quickly. As a result, the viewport quality may be reduced significantly. This result indicates that using large client buffers as in DASH for viewport-adaptive streaming may severely impact user experience. In case long buffering is unavoidable, obviously the *Equal* method should be the best choice.

Currently, the transport layer of the proposed framework is based on that of cloud gaming, which employs UDP/TCP to directly deliver media data. This is because VR services need a very low response delay. A recent trend in low-delay HAS is using the server-push feature of the new HTTP/2 protocol [140]. Using the server-push feature, very short media segments can be used without reducing bandwidth utilization and increasing the number of client requests. In the future work, the use of HTTP/2 to support the low-delay requirement of VR services will be investigated.

4.5 Summary

In this chapter, a client-based adaptation framework for viewport-adaptive streaming of 360 videos is proposed. In the proposed framework, a problem formulation for tile version selection was presented as the basis for designing adaptation methods. Especially, two estimation models that could effectively estimate instant bitrate and quality of video tiles were proposed. From the above experimental results and discussions, some conclusions can be summarized as follows.

- The use of estimated bitrate/quality can help improve the viewport quality significantly. The improvements were seen in all three tile selection methods of the framework.
- The performance with estimated bitrate/quality is nearly the same as the performance with full information of bitrate/quality. This means the complexity of sending full bitrate/quality information of tiles' versions could be avoided.

- The presented estimation solutions and adaptation methods can be applied to both Equirectangular and Cube projection formats.
- Among these two popular projection formats, the latter provides a little better performance.
- Long client buffering could have severe impacts on the visual quality in VR. In this context, ultra low-delay solutions are necessary, which is in line with the new trend of Tactile Internet.

Chapter 5

Adaptive Tiling Selection

5.1 Introduction

Tiling-based approach is the most popular approach for realizing viewport-adaptive streaming [37, 44]. In tiling-based viewport-adaptive streaming, the original video is spatially partitioned into regions called *tiles*. Each tile is further encoded into several versions of different quality levels. Given the user's viewport, the tiles overlapping the viewport (called visible tiles) are streamed at high quality while the other tiles at lower quality [37]. Fig. 5.1 shows a tiling example and the visible tiles corresponding to a specific viewport.

In the current literature, typical tiling schemes include 4×3 [45], 6×4 [44], 8×4 [52], and 8×8 [37]. Some studies investigate good tiling schemes from the server's point of view, by considering the tradeoff between coding efficiency and the number of tiles [44, 53]. However, no metric to decide the optimal tiling scheme has been considered.

In our opinion, the existing studies on optimal tiling scheme miss two important issues. First, the optimal tiling scheme should be mainly considered from the client's point of view. That is, it should be based on the quality performance measured at the client, not at the server. Second, a tiling scheme so far is fixed during a whole streaming session. Intuitively, when the head-moving speed is small, one should use high tiling granularity (i.e., large number of tiles) as it can reduce the amount of redundant pixels, which are the pixels belonging to high quality tiles but not in the viewport. Meanwhile, redundant pixels in case of low tiling granularity (i.e., small number of tiles) can help cope with a high head-moving speed. Since the user head movement is generally varying throughout a streaming session, using a fixed tiling scheme as in existing studies might lead to non-optimal viewport quality.

In this context, it is important to answer some related questions such as "what is the benefit

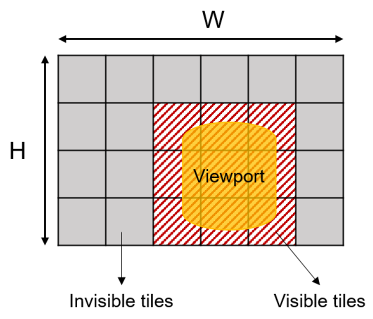


FIGURE 5.1: Illustrations of tiling scheme, viewport, and visible tiles.

of adaptive tiling compared to fixed tiling?", "which tiling should be selected given a speed of head movement?", or "if fixed tiling is preferred in a given context, what is the best tiling scheme for the client?".

For that purpose, a general problem for adaptive tiling in 360 video streaming is fist formulated. Then, correspondingly a simple solution to that problem is devised. Experiment results show that adaptive tiling can improve the average viewport quality by up to 2.3 dB compared to a fixed tiling solution. It is also found that among fixed tiling schemes, 4×3 tiling achieves the lowest viewport quality and thus should not be used.

The remainder of the chapter is organized as follows. Related work is given in Section 5.2. Section 5.3 presents the proposed method. The proposed method is evaluated in Section 5.4. Finally, the chapter is summarized in Section 5.5.

5.2 Related work

In the literature, 360 video is usually divided into equal-sized, non-overlapping tiles by a grid. Various tiling schemes have been used such as 6×4 (ERP) [44], 8×8 (ERP) [76], 12×6 (ERP) [77], 2×2 (CMP) [53, 78], and 4×4 (CMP) [78]. Typically, the tiles are independently encoded. However, this will require multiple decoders at the receiver side. In [46], motion-constrained HEVC tiles are utilized to allows usage of a single decoder instance on the user device. In addition, some variable-sized tiling schemes have been proposed, utilizing the user viewing behaviors [43, 79–81], the content characteristics [79, 82], and the visual attention model [83]. The tiles can also be overlapping as proposed in [84]. In previous studies, tiling schemes are usually selected based on either coding efficiency [44] or content feature [79].

5.3 Adaptive Tiling Selection Method

In this section, the problem formulation of adaptive tiling selection is first presented. Based on that, a solution to the problem is described.

5.3.1 General Problem Formulation

In tiling-based viewport-adaptive streaming system, the tiling scheme is decided every adaptation interval. Each adaptation interval consists of L video frames. The original 360 video is represented as a rectangular video with a width of W (pixels) and a height of H (pixels) using Equirectangular projection [116]. There are K available tiling schemes. The tiling scheme $C_k (1 \leq k \leq K)$ is defined as a grid partition of $T_k = M_k \times N_k$ equally sized tiles (i.e. N_k rows and M_k columns). Each rectangular tile has a width of W/M_k and a height of H/N_k. C_k is also denoted by $M_k \times N_k$. Each tile is encoded into V versions. Version v $(1 \leq v \leq V)$ of tile t $(1 \leq t \leq T_k)$ of tiling scheme C_k $(1 \leq k \leq K)$ of the l^{th} frame $(1 \leq l \leq L)$ has a bitrate of $R_t^k(v,l)$ and a distortion of $D_t^k(v,l)$. In this study, the distortion is measured by the Mean Square Error (MSE). MSE and bitrate values of tiles can be provided as metadata [37]. It should be noted that, as shown in the previous study of [96], PSNR (which is convertible from/to MSE) is still very effective to represent the viewport quality for users.

Suppose that, at a given time, the server needs to adapt an adaptation interval to meet a bandwidth constraint R^c. Denote $\boldsymbol{P}$ the set of the viewport positions when the user watches the frames of the considered adaptation interval. The tiling selection problem can be formulated as follows.

Find a tiling scheme C_k and a version v_t of each tile t so as to minimize the quality objective VQ which is a function of tiles' distortions and viewport positions.

$$VQ = f(\{D_t^k(v,l), 1 \leq t \leq T_k, 1 \leq l \leq L\}, \boldsymbol{P}) \tag{5.1}$$

and satisfy the bitrate constraint

$$\sum_{l=1}^{L} \sum_{t=1}^{T_k} R_t^k(v_t,l) \leq R^c. \tag{5.2}$$

5.3.2 Optimal Solution

In the following, the computation of the quality objective and the method to decide the tiling scheme and the version of each tile will be described. The quality objective VQ is computed as follows. First, the viewport distortion $VQ(l)$ of the l^{th} frame $(1 \leq l \leq L)$ is calculated as the weighted average distortion of the visible tiles as follows.

$$VQ(l) = \sum_{t=1}^{T_k} w_t(l) \times D_t^k(v_t, l). \tag{5.3}$$

Here, the weight $w_t(l)$ indicates how much tile t $(1 \leq t \leq T_k)$ overlaps the viewport at the l^{th} frame. Denote $N(t,l)$ the area of the overlapped area of tile t and N_{vp} the total area of the viewport, the value of $w_t(l)$ is computed as follows.

$$w_t(l) = \frac{N(t,l)}{N_{vp}}. \tag{5.4}$$

It can be note that the value of $w_t(l)$ depends on the head-moving speed. The quality objective *VQ* is then computed as the average viewport distortion over all frames of the adaptation interval as follows.

$$VQ = \frac{1}{L} \sum_{l=1}^{L} VQ(l). \tag{5.5}$$

For selecting the version of each tile, a simple tile selection procedure is applied. Basically, the procedure selects the lowest version for the invisible tiles, and selects the highest possible version for visible tiles. Here, the invisible tiles are also delivered to the client because we have found that some users suddenly changes his/her viewing direction (to the left/right or the back). If the invisible tiles are not sent, the user might experience blank blocks in the viewport. Currently, similar to the previous studies of [44, 45], the visible tiles are determined using the viewport at the first frame of the adaptation interval.

As the problem space is small, a full-search procedure is used to find the optimal tiling scheme as follows.

- **Step 1**: For each tiling scheme
  - Classifying visible tiles and non-visible tiles of the interval.
  - Assigning the lowest version to all non-visible tiles.

  - Finding the highest possible version for all visible tiles that is permitted by the bandwidth constraint R^c.
  - Caclulating the quality objective VQ using (5.3)(5.4)(5.5).

- **Step 2**:
  - Selecting the tiling scheme that achieves the lowest quality objective.
  - Recording the tile versions decided in Step 1 for that tiling scheme.

With the current implementation, the average calculation time of the proposed method is less than 1 ms. Thus, it is able to apply in real-time adaptation.

5.4 Experimental Results

In this experiment, a 360-degree video named *Timelapse*, which is provided in [141], is considered. The video is of Equirectangular format, having a duration of 60 seconds, a resolution of 3840×1920 (4K), and a frame rate of 24 fps. The Field of View (FoV) of the viewport is 90 horizontal degrees × 90 vertical degrees. $K = 4$ tiling schemes of 4×3, 6×4, 8×4, and 8×8 are considered. Each tile is encoded into 7 versions corresponding to 7 quantization parameter values of 24, 28, 32, 36, 40, 44, and 48 using HEVC. The adaptation interval and the buffer size are both set to 1 second. Ten head movement traces which are recorded from 10 different users watching the considered video [141] are used. The CDFs of the angular speed per interval of each trace are shown in Fig. 5.2. It can be seen that the head movements vary among the users. To clearly see the effect of tiling, it is assumed that the viewport positions during each adaptation interval are known in advance. Also, the network bandwidth is constant during each streaming session. Three bandwidth values of 2 Mbps, 4 Mbps, and 6 Mbps are used. The network delay is set to 10 ms.

The proposed method is compared to the conventional method (essentially Step 1 above) in which the tiling scheme is fixed during the streaming session. As PSNR has been proved as most suitable metric for evaluating 360 video [96], viewport PSNR, which measures the quality of a rendered viewport in the sphere domain, is adopted as the performance metric in this study. It is calculated as the PSNR between the rendered viewport and the original viewport. Besides, it is possible that the boundaries between tiles might be visible and cause negative impacts to user experience. This issue will be reserved for future work.

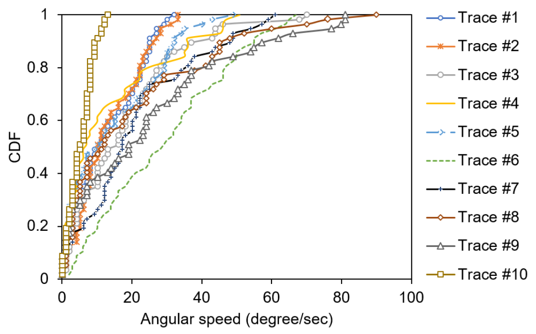


FIGURE 5.2: CDFs of the angular speed per interval of 10 head traces.

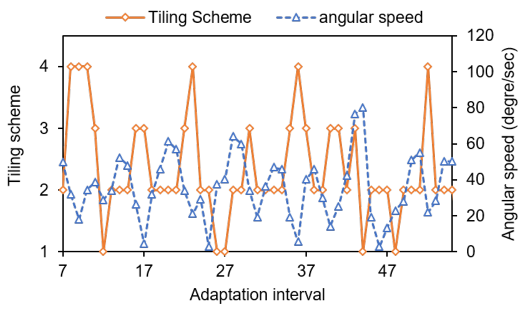


FIGURE 5.3: Selected tiling schemes (1: 4×3, 2: 6×4, 3: 8×4, 4: 8×8) of the proposed method under trace #6 when the bandwidth is 4 Mbps.

Fig. 5.3 shows the selected tiling schemes and the average angular speeds of $7^{th} - 55^{th}$ adaptation intervals of the proposed method under trace #6 when the bandwidth is 4 Mbps. It can be seen that the proposed method can dynamically adapt the tiling scheme. Specifically, a higher number of tiles is selected when the user head movement speed decreases (e.g., $7^{th} - 10^{th}$ intervals). On the other hand, a small number of tiles is used when the head movement speed is high. Fig. 5.4 shows the percentage of tiling schemes decided by the proposed method with the 10 head traces when the bandwidth is 4 Mbps. It can be seen that the selected tiling schemes are strongly correlated to the movement of each trace. For example, 4×3 tiling is not selected in case of traces #10 and #2 as these traces have very low movement speed. Meanwhile, in case of trace #6 where the angular speed is mostly in the range of 30 – 70 (degree/sec), 6×4 tiling is the most selected scheme. When the movement speed spreads out evenly between 0 and 90 (degree/sec) as in case trace #9, the portions of the tiling schemes are very similar.

Tables 5.1, 5.2, and 5.3 show the gain of adaptive tiling compared to fixed tilings when the bandwidth is 2 Mbps, 4 Mbps, and 6 Mbps respectively. Note that the last two rows in these tables summarize the average and maximum values of improvement among the 10 traces.

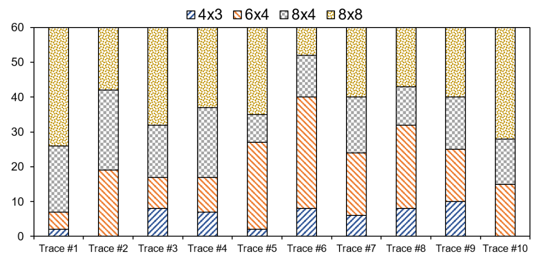


FIGURE 5.4: Percentage of each selected tiling scheme of the proposed method when the bandwidth is 4 Mbps.

TABLE 5.1: Quality gain of adaptive tiling over fixed tiling when the bandwidth is 2 Mbps. The last two rows summarize the average and max values of improvement.

Trace	**Quality gain (dB)**			
	vs. 4×3	**vs. 6×4**	**vs. 8×4**	**vs. 8×8**
#1	**1.6**	**0.7**	**0.6**	**0.3**
#2	**1.6**	**0.7**	**0.3**	**0.4**
#3	**1.2**	**0.7**	**0.4**	**0.4**
#4	**0.9**	**0.7**	**0.3**	**0.4**
#5	**1.4**	**0.5**	**0.5**	**0.6**
#6	**1.1**	**0.3**	**0.4**	**0.6**
#7	**1.3**	**0.5**	**0.5**	**0.6**
#8	**1.9**	**0.5**	**0.5**	**0.8**
#9	**1.4**	**0.7**	**0.5**	**0.6**
#10	**1.9**	**0.5**	**0.5**	**0.4**
Average	**1.4**	**0.6**	**0.4**	**0.5**
Max	**1.9**	**0.7**	**0.6**	**0.8**

TABLE 5.2: Quality gain of adaptive tiling over fixed tiling when the bandwidth is 4 Mbps.

Trace	**Quality gain (dB)**			
	vs. 4×3	**vs. 6×4**	**vs. 8×4**	**vs. 8×8**
#1	**1.9**	**0.9**	**0.7**	**0.3**
#2	**2.1**	**0.8**	**0.4**	**0.3**
#3	**1.6**	**1.0**	**0.6**	**0.4**
#4	**1.2**	**0.8**	**0.4**	**0.4**
#5	**1.8**	**0.6**	**0.8**	**0.7**
#6	**1.3**	**0.4**	**0.7**	**0.9**
#7	**1.7**	**0.8**	**0.8**	**0.9**
#8	**2.2**	**0.6**	**0.7**	**0.8**
#9	**1.6**	**0.8**	**0.7**	**0.7**
#10	**2.3**	**0.8**	**0.4**	**0.2**
Average	**1.8**	**0.8**	**0.6**	**0.5**
Max	**2.3**	**1.0**	**0.8**	**0.9**

TABLE 5.3: Quality gain of adaptive tiling over fixed tiling when the bandwidth is 6 Mbps.

Trace	**Quality gain (dB)**			
	vs. 4×3	**vs. 6×4**	**vs. 8×4**	**vs. 8×8**
#1	**1.5**	**0.8**	**0.6**	**0.2**
#2	**1.9**	**0.8**	**0.4**	**0.3**
#3	**1.2**	**0.8**	**0.5**	**0.4**
#4	**1.0**	**0.6**	**0.3**	**0.4**
#5	**1.5**	**0.5**	**0.7**	**0.7**
#6	**1.2**	**0.3**	**0.7**	**0.8**
#7	**1.3**	**0.7**	**0.7**	**0.8**
#8	**1.6**	**0.5**	**0.5**	**0.6**
#9	**1.2**	**0.7**	**0.6**	**0.6**
#10	**1.9**	**0.9**	**0.3**	**0.1**
Average	**1.4**	**0.7**	**0.5**	**0.5**
Max	**1.9**	**0.9**	**0.7**	**0.8**

Figure 5.5 shows the average viewport PSNR of adaptive tiling and fixed tilings, where the average is computed over the three bandwidth values.

It can be seen that, by adapting the tiling scheme during a streaming session, the proposed method always achieves higher viewport quality than the conventional method. Especially, the improvement is consistent over all three bandwidth values. In general, the improvement compared to 4×3 tiling is highest, while improvements compared to 6×4, 8×4, and 8×8 tilings are similar. As seen in Table 5.2, the proposed method can improve the average viewport PSNR by up to 2.3 dB compared to 4×3 tiling, and up to $0.8 - 1.0$ dB compared to other tilings.

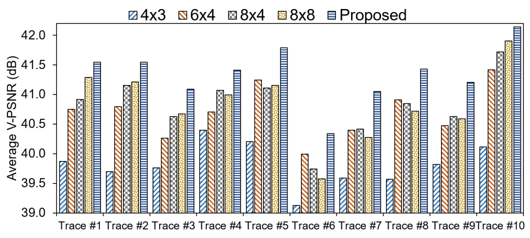


FIGURE 5.5: Average viewport PSNR per trace of the proposed method and the conventional method (averaged over three bandwidth values).

TABLE 5.4: Number of traces in which a fixed tiling scheme achieves the highest performance compared to other fixed tilings.

Tiling Scheme	4×3	6×4	8×4	8×8
Number of Traces	0	3	3	4

Table 5.4 shows the number of head movement traces in which a fixed tiling scheme achieves the highest performance compared to the other fixed tilings. It can be noted that, though having the highest coding efficiency, 4×3 tiling scheme does not achieve the highest viewport PSNR for any traces. This suggests that 4×3 tiling is not effective and thus should not be used. This finding in fact cannot be found if the tiling is considered from the server's point of view. It can also be seen that the tiling schemes of 6×4, 8×4, and 8×8 have similar number of traces where they achieve the highest viewport PSNR. From this result, 6×4 tiling seems to be the best choice of fixed tiling, due to its high PSNR value and lowest number of tiles. A related tiling approach is using overlapped tiles (e.g. [142]) to cope with user head movements. Here, one may adjust both the size and the overlapped parts of tiles. This approach is reserved for our future work.

5.5 Summary

In this chapter, an adaptation method for viewport-adaptive streaming of 360 video is presented that is able to dynamically adapt the tiling scheme based on the user's head movements and the network bandwidth. It was shown that adaptive tiling can improve the average viewport quality by up to 2.3 dB. For future work, optimal tiling selection scheme when applying other tile selection methods will be investigated.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this dissertation, three key aspects of tiling-based viewport-adaptive streaming over networks which are 1) server-based adaptation, 2) client-based adaptation, 3) adaptive tiling selection have been investigated.

First, in Chapter 3, a server-based adaptation framework is proposed. The proposed approach decides the tiles' versions based on viewport estimation errors and user head movements within each video segment. Experimental results show that the proposed approach can achieve high and stable viewport quality. Especially, the average viewport quality is improved by up to 3.8 dB and the standard deviation viewport quality is reduced by up to 1.1 dB when the segment duration is 32 frames. In addition, the impacts of the client buffer size and segment duration is also investigated. It is found that long segment duration and large buffer size negatively impact the performance of the proposed approach.

Second, in Chapter 4, a client-based adaptation framework for tiling-based viewport-adaptive streaming of 360 video is proposed. The proposed framework can support different adaptation scenarios. Especially, two estimation models that can effectively estimate instant bitrate and quality of tiles' versions were proposed. It is found that the proposed bitrate/quality estimation models can help improve the viewport quality significantly. Especially, the performance with estimated bitrate/quality is nearly the same as the performance with full bitrate/quality information. Thus, the complexity of sending full bitrate/quality information could be avoided.

Third, in Chapter 5, a novel adaptation method for viewport-adaptive streaming of 360 video that can dynamically adapt the tiling scheme according to the user's head movements is proposed. Experimental results show that the proposed method can enhance the average viewport quality by up to 2.3 dB compared to the conventional fixed-tiling-based methods.

6.2 Future work

In future work, performance improvements for the proposed frameworks/method will be investigated. For the server-based adaptation framework, the proposed tile version selection methods will be enhanced to better cope with high viewport estimation errors. In the current framework, tiles belonging to the same area have the same version. As the contribution of each tile to the viewport quality is different, it may be better if tile version selection is done for individual tiles.

The current client-based framework still suffers from significant quality degradation under long delay settings. To deal with this issue, advanced methods for long-term viewport estimation are highly desired. In the literature, Recurrent Neural Network (RNN) such as Long-Short Term Memory (LSTM) is found to be effective for long-term estimation [143]. Combination of RNN networks with content features such as saliency map will be considered to improve viewport estimation accuracy.

To improve the performance in client-based adaptation, another approach is to use Scalable Video Coding [53], which can reduce the buffer size to as low as one segment duration while still achieving smooth playback under varying network conditions. SVC can also be combined with new protocol such as HTTP/2 to improve network resource utilization and viewport quality.

As aforementioned, the current work only considers monscopic 360 video, which is the most simple form of immersive video. Other immersive content such as stereoscopic 360 video [144] and volumetric video [145] have much higher bitrate and different characteristics. Thus, effective adaptation/transmission methods for other immersive content types will be another topic in our future work.

Bibliography

[1] Qualcomm. (2016) Making immersive virtual reality possible in mobile. [Online]. Available: https://www.qualcomm.com/media/documents/files/whitepaper-making-immersive-virtual-reality-possible-in-mobile.pdf

[2] J. Lin, Y. Lee, C. Shih, S. Lin, H. Lin, S. Chang, P. Wang, L. Liu, and C. Ju, "Efficient projection and coding tools for 360° video," *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 1, pp. 84–97, March 2019.

[3] M. Chen, K. Hu, I. Chung, and C. Chou, "Towards VR/AR Multimedia Content Multicast over Wireless LAN," in *IEEE Annual Consumer Communications Networking Conf. (CCNC)*, Las Vegas, US, Jan 2019, pp. 1–6.

[4] L. Freina and M. Ott, "A Literature Review on Immersive Virtual Reality in Education: State Of The Art and Perspectives," in *Proc. Int. Conf. eLearning and Software for Education*, Bucharest, Romania, Apr. 2015, pp. 133–141.

[5] J. Blascovich and J. Bailenson, *Infinite reality: The hidden blueprint of our virtual lives*. William Morrow New York, 2012.

[6] N. Vaughan, V. N. Dubey, T. W. Wainwright, and R. G. Middleton, "A review of virtual reality based training simulators for orthopaedic surgery," *Medical engineering & physics*, vol. 38, no. 2, pp. 59–71, 2016.

[7] D. Roberts, A. Fairchild, S. Campion, A. Garcia Jimenez, and R. Wolff, "Bringing the client and therapist together in virtual reality telepresence exposure therapy," in *Proc. Int. Conf. on Disability, Virtual Reality and Associated Technologies*, LA, US, Sep. 2016, pp. 157–163.

[8] J. Gugenheimer, D. Wolf, G. Haas, S. Krebs, and E. Rukzio, "Swivrchair: A motorized swivel chair to nudge users' orientation for 360 degree storytelling in virtual reality," in *Proc. CHI Conf. on Human Factors in Computing Systems*, San Jose, CA, USA, 2016, pp. 1996–2000.

[9] Huewei. (2017) Virtual reality/augmented reality white paper. [Online]. Available: http://www-file.huawei.com/-/media/CORPORATE/PDF/ilab/vr-ar-en.pdf

[10] Facebook. Introduction to Virtual Reality Audio. Accessed: 2019-03-08. [Online]. Available: https://developer.oculus.com/documentation/audiosdk/latest/concepts/audio-intro-overview/

[11] A. Maimone, A. Georgiou, and J. S. Kollin, "Holographic Near-eye Displays for Virtual and Augmented Reality," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 85:1–85:16, Jul. 2017.

[12] D. Dunn, C. Tippets, K. Torell, P. Kellnhofer, K. Akşit, P. Didyk, K. Myszkowski, D. Luebke, and H. Fuchs, "Wide field of view varifocal near-eye display using see-through deformable membrane mirrors," *IEEE Trans. Vis. Comput. Graphics*, vol. 23, no. 4, pp. 1322–1331, 2017.

[13] E. Whitmire, L. Trutoiu, R. Cavin, D. Perek, B. Scally, J. Phillips, and S. Patel, “EyeContact: scleral coil eye tracking for virtual reality,” in *Proc. ACM Int. Symp. on Wearable Computers*, Heidelberg, Germany, Sep. 2016, pp. 184–191.

[14] D. T. Han, M. Suhail, and E. D. Ragan, “Evaluating Remapped Physical Reach for Hand Interactions with Passive Haptics in Virtual Reality,” *IEEE Trans. Vis. Comput. Graphics*, vol. 24, no. 4, pp. 1467–1476, 2018.

[15] P.-W. Lee, H.-Y. Wang, Y.-C. Tung, J.-W. Lin, and A. Valstar, “TranSection: hand-based interaction for playing a game within a virtual reality game,” in *Proc. Annual ACM Conf. on Human Factors in Computing Systems*, Seoul, Republic of Korea, Apr. 2015, pp. 73–76.

[16] D. Hong, T.-H. Lee, Y. Joo, and W.-C. Park, “Real-time sound propagation hardware accelerator for immersive virtual reality 3D audio,” in *Proc. ACM SIGGRAPH Symp. on Interactive 3D Graphics and Games*, San Francisco, California, Feb. 2017, pp. 20:1–20:2.

[17] C. Pike, R. Taylor, T. Parnell, and F. Melchior, “Object-based 3D audio production for virtual reality using the audio definition model,” in *AES Int. Conf. on Audio for Virtual and Augmented Reality*, LA, US, Sep. 2016, p. 2:1.

[18] A. Patney, M. Salvi, J. Kim, A. Kaplanyan, C. Wyman, N. Benty, D. Luebke, and A. Lefohn, “Towards foveated rendering for gaze-tracked virtual reality,” *ACM Trans. on Graphics (TOG)*, vol. 35, no. 6, p. 179, 2016.

[19] A. Patney, J. Kim, M. Salvi, A. Kaplanyan, C. Wyman, N. Benty, A. Lefohn, and D. Luebke, “Perceptually-based foveated virtual reality,” in *ACM SIGGRAPH*, Anaheim, California, Jul. 2016, pp. 17:1–17:2.

[20] F. De Sorbier, V. Nozick, and H. Saito, “Multi-view Rendering using GPU for 3-D Displays,” *GSTF Journal on Computing (JoC)*, vol. 1, no. 1, pp. 1–5, 2018.

[21] S. Mangiante, G. Klas, A. Navon, Z. GuanHua, J. Ran, and M. D. Silva, “VR is on the Edge: How to Deliver 360-degree Videos in Mobile Networks,” in *Proc. Workshop on Virtual Reality and Augmented Reality Network*, Los Angeles, CA, USA, 2017, pp. 30–35.

[22] S. Sukhmani, M. Sadeghi, M. Erol-Kantarci, and A. El Saddik, “Edge Caching and Computing in 5G for Mobile AR/VR and Tactile Internet,” *IEEE MultiMedia*, vol. 26, no. 1, pp. 21–30, Jan 2019.

[23] L. Shi, F.-C. Huang, W. Lopes, W. Matusik, and D. Luebke, “Near-eye light field holographic rendering with spherical waves for wide field of view interactive 3d computer graphics,” *ACM Trans. on Graphics (TOG)*, vol. 36, no. 6, p. 236, 2017.

[24] M. Shafi, A. F. Molisch, P. J. Smith, T. Haustein, P. Zhu, P. De Silva, F. Tufvesson, A. Benjebbour, and G. Wunder, “5G: A tutorial overview of standards, trials, challenges, deployment, and practice,” *IEEE JSAC*, vol. 35, no. 6, pp. 1201–1221, 2017.

[25] S. Bindhaiq, A. S. M. Supa, N. Zulkifli, A. B. Mohammad, R. Q. Shaddad, M. A. Elmagzoub, and A. Faisal, “Recent development on time and wavelength-division multiplexed passive optical network (TWDM-PON) for next-generation passive optical network stage 2 (NG-PON2),” *Optical Switching and Networking*, vol. 15, pp. 53–66, 2015.

[26] H. A. Omar, K. Abboud, N. Cheng, K. R. Malekshan, A. T. Gamage, and W. Zhuang, "A survey on high efficiency wireless local area networks: Next generation WiFi," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2315–2344, 2016.

[27] K. He, E. Rozner, K. Agarwal, Y. J. Gu, W. Felter, J. Carter, and A. Akella, "AC/DC TCP: Virtual congestion control enforcement for datacenter networks," in *Proc. ACM SIGCOMM*, Florianopolis, Brazil, Aug. 2016, pp. 244–257.

[28] R. Mittal, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "TIMELY: RTT-based Congestion Control for the Datacenter," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, 2015, pp. 537–550.

[29] R. Trivisonno, R. Guerzoni, I. Vaishnavi, and A. Frimpong, "Network resource management and QoS in SDN-enabled 5G systems," in *IEEE Global Communications Conf. (GLOBECOM)*, San Diego, CA, USA, Dec. 2015, pp. 1–7.

[30] C. Xu, B. Chen, and H. Qian, "Quality of service guaranteed resource management dynamically in software defined network," *Journal of Communications*, vol. 10, no. 11, pp. 843–850, 2015.

[31] E. Hossain and M. Hasan, "5G cellular: key enabling technologies and research challenges," *IEEE Instrumentation Measurement Magazine*, vol. 18, no. 3, pp. 11–21, June 2015.

[32] B. Bross, "Versatile video coding (draft 1)," in *Joint Video Experts Team (JVET) of ITU-T SG 16 WP3 and ISO/IEC JTC1 SC29/WG11, 10th meeting*, San Diego, CA, USA, April 2018.

[33] H. T. Le, D. V. Nguyen, N. P. Ngoc, A. T. Pham, and T. C. Thang, "Buffer-based bitrate adaptation for adaptive http streaming," in *Int. Conf. on Advanced Technologies for Communications*, Ho Chi Minh City, Vietnam, Oct 2013, pp. 33–38.

[34] T. Vu, H. T. Le, D. V. Nguyen, N. Pham Ngoc, and T. C. Thang, "Future buffer based adaptation for vbr video streaming over http," in *IEEE Int. Workshop on Multimedia Signal Processing (MMSP)*, Xiamen, China, Oct 2015, pp. 1–5.

[35] T. Nguyen, T. Vu, D. V. Nguyen, N. P. Ngoc, and T. C. Thang, "QoE optimization for adaptive streaming with multiple VBR videos," in *Int. Conf. on Communications, Management and Telecommunications (ComManTel)*, DaNang, Vietnam, Dec 2015, pp. 189–193.

[36] D. V. Nguyen, D. M. Nguyen, H. T. Tran, , A. T. Pham, and T. C. Thang, "Quality-delay tradeoff optimization in multi-bitrate adaptive streaming," in *IEEE Int. Conf. on Consumer Electronics (ICCE)*, Las Vegas, USA, Jan 2015, pp. 66–67.

[37] D. V. Nguyen, H. T. T. Tran, A. T. Pham, and T. C. Thang, "A new adaptation approach for viewport-adaptive 360-degree video streaming," in *Proc. IEEE Int. Symp. on Multimedia (ISM)*, Taichung, Taiwan, Dec 2017, pp. 38–44.

[38] H. T. T. Tran, N. P. Ngoc, A. T. Pham, and T. C. Thang, "A Multi-Factor QoE Model for Adaptive Streaming over Mobile Networks," in *IEEE Globecom Workshops (GC Wkshps)*, Washington, DC, USA, Dec. 2016, pp. 1–6.

[39] H. T. Tran, N. P. Ngoc, Y. J. Jung, A. T. Pham, and T. C. Thang, "A Histogram-Based Quality Model for HTTP Adaptive Streaming," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 100, no. 2, pp. 555–564, 2017.

[40] H. T. T. Tran, N. P. Ngoc, T. Hoßfeld, and T. C. Thang, "A Cumulative Quality Model for HTTP Adaptive Streaming," in *Int. Conf. on Quality of Multimedia Experience (QoMEX)*, Cagliari, Italy, May 2018, pp. 1–6.

[41] 360-degree video. Accessed: 2019-03-09. [Online]. Available: https://en.wikipedia.org/wiki/360-degree-video

[42] M. Domański, O. Stankiewicz, K. Wegner, and T. Grajek, "Immersive visual media — MPEG-I: 360 video, virtual navigation and beyond," in *Int. Conf. on Systems, Signals and Image Processing (IWSSIP)*, Poznań, Poland, May 2017, pp. 1–9.

[43] M. Hosseini and V. Swaminathan, "Adaptive 360 VR Video Streaming : Divide and Conquer!" in *IEEE Int. Symp. on Multimedia (ISM)*, San Jose, US, 2016, pp. 107–110.

[44] M. Graf, C. Timmerer, and C. Mueller, "Towards Bandwidth Efficient Adaptive Streaming of Omnidirectional Video over HTTP," in *Proc. ACM Multimedia Systems Conf. (MMSys'17)*, Taipei, Taiwan, June 2017, pp. 261–271.

[45] A. Zare, A. Aminlou, M. M. Hannuksela, and M. Gabbouj, "HEVC-compliant Tile-based Streaming of Panoramic Video for Virtual Reality Applications," in *Proc. IEEE Picture Coding Symposium (PCS)*, Nuremberg, Germany, Dec. 2016, pp. 601–605.

[46] R. Skupin, Y. Sanchez, D. Podborski, C. Hellge, and T. Schierl, "HEVC tile based streaming to head mounted displays," in *Proc. IEEE Annual Consumer Communications Networking Conf. (CCNC)*, Las Vegas, USA, Jan. 2017, pp. 613–615.

[47] T. C. Thang, Q. D. Ho, J. W. Kang, and A. T. Pham, "Adaptive streaming of audiovisual content using MPEG DASH," *IEEE Trans. Consum. Electron.*, vol. 58, no. 1, pp. 78–85, 2012.

[48] T. M. Bae, T. C. Thang, D. Y. Kim, Y. M. Ro, J. W. Kang, and J.-G. Kim, "Multiple ROI support in Scalable Video Coding," *ETRI Journal*, vol. 28, no. 2, pp. 239–242, Apr 2006.

[49] F. Qian, L. Ji, B. Han, and V. Gopalakrishnan, "Optimizing 360 video delivery over cellular networks," in *Proc. 5th Workshop on All Things Cellular: Operations, Applications and Challenges (ATC'16)*, NY, USA, October 2016, pp. 1–6.

[50] T. C. Thang, H. T. Le, H. X. Nguyen, A. T. Pham, J. W. Kang, and Y. M. Ro, "Adaptive video streaming over HTTP with dynamic resource estimation," *Journal of Communications and Networks*, vol. 15, pp. 635–644, 2013.

[51] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, Dec 2012.

[52] H. Ahmadi, O. Eltobgy, and M. Hefeeda, "Adaptive Multicast Streaming of Virtual Reality Content to Mobile Users," in *Proc. Thematic Workshops of ACM Multimedia*, Mountain View, California, USA, Oct. 2017, pp. 170–178.

[53] A. T. Nasrabadi, A. Mahzari, J. D. Beshay, and R. Prakash, “Adaptive 360-degree video streaming using Scalable Video Coding,” in *Proc. ACM Int. Conf. on Multimedia*, Mountain View, California, USA, Oct. 2017, pp. 1689–1697.

[54] X. Corbillon, A. Devlic, G. Simon, and J. Chakareski, “Viewport-Adaptive Navigable 360-Degree Video Delivery,” in *Proc. IEEE ICC*, Paris, France, May. 2017, pp. 1–7.

[55] H. T. T. Tran, D. V. Nguyen, N. P. Ngoc, and T. C. Thang, “A tile-based solution using cubemap for viewport-adaptive 360-degree video delivery,” *IEICE Trans. on Communications*, vol. advpub, 2019.

[56] D. V. Nguyen, H. T. T. Tran, A. T. Pham, and T. C. Thang, “An optimal tile-based approach for viewport-adaptive 360-degree video streaming,” *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 1, pp. 29–42, March 2019.

[57] D. Nguyen, H. T. Tran, and T. C. Thang, “A client-based adaptation framework for 360-degree video streaming,” *Journal of Visual Communication and Image Representation*, vol. 59, pp. 231 – 243, 2019.

[58] D. V. Nguyen, H. T. T. Tran, and T. C. Thang, “Adaptive tiling selection for viewport adaptive streaming of 360-degree video,” *IEICE Trans. on Information and Systems*, vol. E102.D, no. 1, pp. 48–51, 2019.

[59] GoPro. Here is Odyssey. Accessed: 2019-03-08. [Online]. Available: https://jp.gopro.com/news/here-is-odyssey

[60] Ricoh. Ricoh Theta Z1. Accessed: 2019-06-20. [Online]. Available: https://theta360.com/en/about/theta/z1.html

[61] Samsung. Gear 360. Accessed: 2019-03-08. [Online]. Available: https://www.samsung.com/global/galaxy/gear-360/

[62] Kodak, “SP360 4K,” accessed: 2019-03-08. [Online]. Available: https://kodakpixpro.com/cameras/360-vr/sp360-4k

[63] W.-T. Lee, H.-I. Chen, M.-S. Chen, I.-C. Shen, and B.-Y. Chen, “High-resolution 360 Video Foveated Stitching for Real-time VR,” *Computer Graphics Forum*, vol. 36, no. 7, pp. 115–123, 2017.

[64] J.-Y. Shieh, Y.-C. Liao, G.-J. Lioao, and Y.-T. Liou, “Dynamic image stitching for panoramic video,” *Int. J. of Engineering and Technology Innovation*, vol. 4, no. 4, pp. 260–268, Oct. 2014.

[65] Q. Liu, X. Su, L. Zhang, and H. Huang, “Panoramic video stitching of dual cameras based on spatio-temporal seam optimization,” *Multimedia Tools and Applications*, pp. 1–19, Jul. 2018.

[66] R. M. A. Silva, B. Feijó, P. B. Gomes, T. Frensh, and D. Monteiro, “Real Time 360: Video Stitching and Streaming,” in *ACM SIGGRAPH*, Anaheim, California, 2016, pp. 70:1–70:2.

[67] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, “Overview of the H.264/AVC video coding standard,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 560–576, 2003.

[68] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, 2012.

[69] C. Grunheit, A. Smolic, and T. Wiegand, "Efficient representation and interactive streaming of high-resolution panoramic views," in *Int. Conf. on Image Processing*, Rochester, NY, USA, Sep. 2002, pp. III–209–III–212.

[70] N. King-To, C. Shing-Chow, and S. Heung-Yeung, "Data compression and transmission aspects of panoramic videos," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 1, pp. 82–95, Jan 2005.

[71] Facebook. Next-generation video encoding techniques for 360 video and vr. Accessed: 2019-03-11. [Online]. Available: https://code.fb.com/virtual-reality/next-generation-video-encoding-techniques-for-360-video-and-vr/

[72] Google. Bringing pixels front and center in vr video. Accessed: 2019-03-18. [Online]. Available: https://blog.google/products/google-ar-vr/bringing-pixels-front-and-center-vr-video/

[73] G. V. der Auwera, M. Coban, Hendry, and M. Karczewicz, "Ahg8: Truncated square pyramid projection (tsp) for 360 video," in *Joint Video Exploration Team of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, JVET-D0071*, Oct. 2016.

[74] P. Hanhart, X. Xiu, Y. He, and Y. Ye, "360° video coding based on projection format adaptation and spherical neighboring relationship," *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 1, pp. 71–83, March 2019.

[75] X. Sun, F. Wu, S. Li, and W. Gao, "In-loop deblocking filter for block based video coding," in *Proc. Int. Conf. on Signal Processing*, Beijing, China, Aug. 2002, pp. 33–36.

[76] P. R. Alface, J.-F. Mac, and V. Nico, "Interactive Omnidirectional Video Delivery: A Bandwidth-Effective Approach," *Bell Labs Technical Journal*, vol. 16, no. 4, pp. 135–148, 2012.

[77] L. Xie, X. Zhang, and Z. Guo, "360ProbDASH: Improving QoE of 360 Video Streaming Using Tile-based HTTP Adaptive Streaming," in *Proc. ACM Multimedia Conf.*, CA, USA, Oct. 2017, pp. 315–323.

[78] Y. Sanchez, R. Skupin, C. Hellge, and T. Schierl, "Random access point period optimization for viewport adaptive tile based streaming of 360 video," in *Proc. IEEE Int. Conf. on Image Processing (ICIP)*, Beijing, China, Sept 2017, pp. 1915–1919.

[79] M. Xiao, C. Zhou, Y. Liu, and S. Chen, "OpTile: Toward Optimal Tiling in 360-degree Video Streaming," in *Proc. ACM Multimedia Conf.*, Mountain View, California, USA, Oct. 2017, pp. 708–716.

[80] C. Ozcinar, A. De Abreu, and A. Smolic, "Viewport-aware adaptive 360° video streaming using tiles for virtual reality," in *IEEE Int. Conf. on Image Processing (ICIP)*, Beijing, China, Sep. 2017, pp. 2174–2178.

[81] S. Petrangeli, V. Swaminathan, M. Hosseini, and F. De Turck, "An HTTP/2-based adaptive streaming framework for 360° virtual reality videos," in *ACM Multimedia Conf.*, 2017, pp. 1–9.

[82] Z. Tu, T. Zong, X. Xi, L. Ai, Y. Jin, X. Zeng, and Y. Fan, "Content adaptive tiling method based on user access preference for streaming panoramic video," in *IEEE Int. Conf. on Consumer Electronics (ICCE)*, Las Vegas, NV, USA, Jan 2018, pp. 1–4.

[83] C. Ozcinar, J. Cabrera, and A. Smolic, "Visual attention-aware omnidirectional video streaming using optimal tiles for virtual reality," *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 1, pp. 217–230, March 2019.

[84] F. Duanmu, E. Kurdoglu, Y. Liu, and Y. Wang, "View direction and bandwidth adaptive 360 degree video streaming using a two-tier system," in *IEEE Int. Symp. on Circuits and Systems (ISCAS)*, May 2017, pp. 1–4.

[85] H. T. Tran, H. T. Le, N. P. Ngoc, A. T. Pham, and T. C. Thang, "Quality improvement for video on-demand streaming over http," *IEICE Trans. on Information and Systems*, vol. 100, no. 1, pp. 61–64, 2017.

[86] H. Tran, N. Pham Ngoc, T. Cong Thang, and Y. Man Ro, "Real-time quality evaluation of adaptation strategies in vod streaming," in *Digital Media Industry Academic Forum (DMIAF)*, Santorini, Greece, July 2016, pp. 217–221.

[87] H. T. Le, N. P. Ngoc, and C.-T. Truong, "Bitrate adaptation for seamless on-demand video streaming over mobile networks," *Signal Processing: Image Communication*, vol. 65, pp. 154 – 164, 2018.

[88] H. T. Le, H. N. Nguyen, N. Pham Ngoc, A. T. Pham, H. Le Minh, and T. C. Thang, "Quality-driven bitrate adaptation method for HTTP live-streaming," in *IEEE Int. Conf. on Communication Workshop (ICCW)*, London, UK, June 2015, pp. 1771–1776.

[89] D. M. Nguyen, L. B. Tran, H. T. Le, N. P. Ngoc, and T. C. Thang, "An evaluation of segment duration effects in http adaptive streaming over mobile networks," in *NAFOSTED Conf. on Information and Computer Science (NICS)*, Ho Chi Minh City, Vietnam, Sep. 2015, pp. 248–253.

[90] H. N. Nguyen, T. Vu, H. T. Le, N. P. Ngoc, and T. C. Thang, "Smooth quality adaptation method for vbr video streaming over http," in *Int. Conf. on Communications, Management and Telecommunications (ComManTel)*, DaNang, Vietnam, Dec 2015, pp. 184–188.

[91] S. Petrangeli, G. Simon, and V. Swaminathan, "Trajectory-based viewport prediction for 360-degree virtual reality videos," in *IEEE Int. Conf. on Artificial Intelligence and Virtual Reality (AIVR)*, Taichung, Taiwan, Dec. 2018, pp. 157–160.

[92] D. V. Nguyen and H. T. T. Tran and T. C. Thang, "Impact of delays on 360-degree video communications," in *TRON Symposium (TRONSHOW)*, 12 2017, pp. 1–6.

[93] L. Xie, Z. Xu, Y. Ban, X. Zhang, and Z. Guo, "360ProbDASH: Improving QoE of 360 Video Streaming Using Tile-based HTTP Adaptive Streaming," in *Proc. ACM Multimedia Conf.*, CA, USA, Oct. 2017, pp. 315–323.

[94] J. Chakareski, R. Aksu, X. Corbillon, G. Simon, and V. Swaminathan, "Viewport-driven rate-distortion optimized 360° video streaming," in *IEEE Int. Conf. on Communications (ICC)*, MO, USA, May 2018, pp. 1–7.

[95] H. T. T. Tran, N. P. Ngoc, C. T. Pham, Y. J. Jung, and T. C. Thang, "A subjective study on QoE of 360 video for VR communication," in *Proc. IEEE MMSP*, Luton, UK, Oct 2017, pp. 1–6.

[96] H. T. T. Tran, C. T. Pham, N. P. Ngoc, A. T. Pham, and T. C. Thang, "A Study on Quality Metrics for 360 Video Communications," *IEICE Trans. on Information and Systems*, vol. E101-D, no. 1, pp. 28–36, 2018.

[97] H. T. T. Tran, N. P. Ngoc, C. M. Bui, M. H. Pham, and T. C. Thang, "An evaluation of quality metrics for 360 videos," in *Int. Conf. on Ubiquitous and Future Networks (ICUFN)*, July 2017, pp. 7–11.

[98] D. H. Nguyen, M. Nguyen, N. P. Ngoc, and T. C. Thang, "An adaptive method for low-delay 360 VR video streaming over HTTP/2," in *IEEE Int. Conf. on Communications and Electronics (ICCE)*, July 2018, pp. 261–266.

[99] M. Nguyen, D. H. Nguyen, C. T. Pham, N. P. Ngoc, D. V. Nguyen, and T. C. Thang, "An adaptive streaming method of 360 videos over HTTP/2 protocol," in *NAFOSTED Conf. on Information and Computer Science*, Hanoi, Vietnam, Nov 2017, pp. 302–307.

[100] M. Ben Yahia, Y. Le Louedec, G. Simon, and L. Nuaymi, "HTTP/2-Based Streaming Solutions for Tiled Omnidirectional Videos," in *IEEE Int. Symp. on Multimedia (ISM)*, Dec 2018, pp. 89–96.

[101] M. Belshe, R. Peon, and M. Thomson, "Hypertext transfer protocol version 2 (http/2)," Tech. Rep., 2015.

[102] H. T. Le, T. Nguyen, N. P. Ngoc, A. T. Pham, and T. C. Thang, "HTTP/2 Push-Based Low-Delay Live Streaming Over Mobile Networks With Stream Termination," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 9, pp. 2423–2427, Sep. 2018.

[103] T. Nguyen, N. H. Dang, N. Minh, N. P. Ngoc, H. T. Le, and T. C. Thang, "An Efficient Server Push Approach for On-Demand Video Streaming Over HTTP/2," *IEICE Trans. on Communications*, vol. E101.B, no. 11, pp. 2371–2379, 2018.

[104] H. T. Le, T. Vu, N. P. Ngoc, A. T. Pham, and T. C. Thang, "Seamless mobile video streaming over HTTP/2 with gradual quality transitions," *IEICE Trans. on Communications*, vol. 100, no. 5, pp. 901–909, 2017.

[105] Z. Xu, Y. Ban, K. Zhang, L. Xie, X. Zhang, Z. Guo, S. Meng, and Y. Wang, "Tile-Based Qoe-Driven Http/2 Streaming System For 360 Video," in *IEEE Int. Conf. on Multimedia & Expo Workshops (ICMEW)*, San Diego, CA, USA, 2018, pp. 1–4.

[106] W. Lo, C. Huang, and C. Hsu, "Edge-Assisted Rendering of 360° Videos Streamed to Head-Mounted Virtual Reality," in *IEEE Int. Symp. on Multimedia (ISM)*, Dec 2018, pp. 44–51.

[107] L. Bassbouss, S. Steglich, and M. Lasak, "High quality 360° video rendering and streaming," in *NEM Summit proceedings*, 2016.

[108] Y. S. de la Fuente, G. S. Bhullar, R. Skupin, C. Hellge, and T. Schierl, "Delay Impact on MPEG OMAF's Tile-Based Viewport-Dependent 360° Video Streaming," *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 1, pp. 18–28, March 2019.

[109] H. T. T. Tran, T. H. Hoang, P. N. Minh, N. P. Ngoc, and T. C. Thang, "A perception-based quality metric for omnidirectional images," in *IEEE Int. Conf. on Consumer Electronics - Asia (ICCE-Asia)*, Bangkok, Thailand, Jun. 2019, pp. –.

[110] Texture filtering with mipmaps (direct3d 9. Accessed: 2019-05-25. [Online]. Available: https://docs.microsoft.com/en-us/windows/desktop/direct3d9/texture-filtering-with-mipmaps

[111] Japan's First Real 8K Dome Theater into Fukui City Museum of Astronomy. Accessed: 2019-05-25. [Online]. Available: http://www.goto.co.jp/english/news/20160728/

[112] Fulldome earth science simulation. Accessed: 2019-05-25. [Online]. Available: https://www.spitzinc.com/planetarium/scidome/fulldome-earth-science/

[113] Y. Bao, H. Wu, T. Zhang, A. A. Ramli, and X. Liu, "Shooting a moving target: Motion-prediction-based transmission for 360-degree videos," in *Proc. IEEE Int. Conf. on Big Data (Big Data)*, Washington, DC, USA, Dec 2016, pp. 1161–1170.

[114] T. Lohmar, T. Einarsson, P. Fröjdh, F. Gabin, and M. Kampmann, "Dynamic adaptive HTTP streaming of live content," in *Proc. IEEE Int. Symp. on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, Lucca, Italy, 2011, pp. 1–8.

[115] K. S. Kim, D. K. Kim, C. Chae, S. Choi, Y. Ko, J. Kim, Y. Lim, M. Yang, S. Kim, B. Lim, K. Lee, and K. L. Ryu, "Ultrareliable and Low-Latency Communication Techniques for Tactile Internet Services," *Proc. the IEEE*, vol. 107, no. 2, pp. 376–393, Feb. 2019.

[116] Y. Ye, E. Alshina, and J. Boyce, "JVET-E1003: Algorithm descriptions of projection format conversion and video quality metrics in 360Lib," in *Joint Video Exploration Team ITU-T SG 16 WP3 ISO/IEC JTC 1/SC 29/WG 11 5th Meet.* Geneva, Switzerland: Joint Video Exploration Team, 2017.

[117] M. Yu, H. Lakshman, and B. Girod, "A Framework to Evaluate Omnidirectional Video Coding Schemes," in *Proc. IEEE Int. Symp. on Mixed and Augmented Reality*, Fukuoka, Japan, 2015, pp. 31–36.

[118] K. Kammachi-Sreedhar, A. Aminlou, M. M. Hannuksela, and M. Gabbouj, "Viewport-adaptive encoding and streaming of 360-degree video for virtual reality applications," in *Proc. IEEE Int. Symp. Multimedia*, San Jose, US, 2016, pp. 583–586.

[119] YouTube. Improving VR videos. Accessed: 2018-05-23. [Online]. Available: https://youtube-eng.googleblog.com/2017/03/improving-vr-videos.html

[120] Y. Li, J. Xu, and Z. Chen, "Spherical domain rate-distortion optimization for 360-degree video coding," in *IEEE Int. Conf. on Multimedia and Expo (ICME)*, July 2017, pp. 709–714.

[121] Y. Sun and L. Yu, "Coding optimization based on weighted-to-spherically-uniform quality metric for 360 video," in *IEEE Visual Communications and Image Processing Conf. (VCIP)*, St. Petersburg, FL, USA, Dec 2017, pp. 1–4.

[122] B. Li, L. Song, R. Xie, and W. Zhang, "Weight-based bit allocation scheme for VR videos in HEVC," in *IEEE Visual Communications and Image Processing Conf. (VCIP)*, St. Petersburg, FL, USA, Dec 2017, pp. 1–4.

[123] Y. He, Y. Ye, P. Hanhart, and X. Xiu, "Geometry padding for motion compensated prediction in 360 video coding," in *Data Compression Conf. (DCC)*, Snowbird, UT, USA, April 2017, pp. 443–443.

[124] Y. Wang, Y. Li, D. Yang, and Z. Chen, "A fast intra prediction algorithm for 360-degree equirectangular panoramic video," in *IEEE Visual Communications and Image Processing (VCIP)*, St. Petersburg, FL, USA, Dec 2017, pp. 1–4.

[125] X. Xiu, Y. He, Y. Ye, and B. Vishwanath, “An evaluation framework for 360-degree video compression,” in *IEEE Visual Communications and Image Processing (VCIP)*, St. Petersburg, FL, USA, Dec 2017, pp. 1–4.

[126] C.-L. Fan, J. Lee, and K.-T. Chen, “Fixation Prediction for 360° Video Streaming in Head-Mounted Virtual Reality,” in *Proc. 8th ACM Multimedia System Conf.* Taipei, Taiwan: ACM SIGMM, 2017, pp. 67–72.

[127] R. Aksu, J. Chakareski, and V. Swaminathan, “Viewport-driven rate-distortion optimized scalable live 360° video network multicast,” in *Proc. IEEE ICME Int’l Workshop on Hot Topics in 3D (Hot3D)*, CA, USA, Jul. 2018, pp. 1–7.

[128] S. Zhao and D. Medhi, “SDN-Assisted Adaptive Streaming Framework for Tile-Based Immersive Content Using MPEG-DASH,” in *IEEE NFV-SDN*, Nov. 2017.

[129] C.-y. Huang, C.-h. Hsu, Y.-C. Chang, and K.-T. Chen, “GamingAnywhere: An Open Cloud Gaming System,” in *Proc. ACM Multimedia Systems Conf. (MMSys’13)*, Oslo, Norway, 2013, pp. 36–47.

[130] D. V. Nguyen, H. T. T. Tran, P. N. Nam, and T. C. Thang, “A QoS-adaptive framework for screen sharing over Internet,” in *Eighth Int. Conf. on Ubiquitous and Future Networks (ICUFN)*, Vienna, Austria, July 2016, pp. 972–974.

[131] L. Rizzo, “Dummynet: A Simple Approach to the Evaluation of Network Protocols,” in *SIGCOMM Computer Communications Review*, vol. 27, no. 1, 1997, pp. 31–41.

[132] A. Bokani, M. Hassan, S. S. Kanhere, J. Yao, and G. Zhong, “Comprehensive Mobile Bandwidth Traces from Vehicular Networks,” in *Proc. Int. Conf. on Multimedia Systems*. New York, NY, USA: ACM, 2016, pp. 44:1–44:6.

[133] Niamut, Omar A. and Thomas, Emmanuel and D’Acunto, Lucia and Concolato, Cyril and Denoual, Franck and Lim, Seong Yong, “MPEG DASH SRD: Spatial Relationship Description,” in *Proc. Int. Conf. on Multimedia Systems*, New York, NY, USA, 2016, pp. 5:1–5:8.

[134] H. T. Le, N. P. Ngoc, A. T. Pham, and T. C. Thang, “A Probabilistic Adaptation Method for HTTP Low-Delay Live Streaming over Mobile Networks,” *IEICE Trans. on Information and Systems*, vol. E100.D, no. 2, pp. 379–383, 2017.

[135] Miller, Konstantin and Al-Tamimi, Abdel-Karim and Wolisz, Adam, “Qoe-based low-delay live streaming using throughput predictions,” *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 13, no. 1, pp. 4:1–4:24, Oct. 2016.

[136] Jill Boyce and Elena Alshina and Adeel Abbas, “Jvet common test conditions and evaluation procedures for 360^o video coding,” in *Joint Video Exploration Team of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, JVET-D1030, 4th Meeting*, Chengdu, China, Oct. 2016.

[137] FFmpeg team, “Ffmpeg,” https://ffmpeg.org/, July 2016. [Online]. Available: https://ffmpeg.org/

[138] JVET, “360lib,” https://jvet.hhi.fraunhofer.de/, October 2016. [Online]. Available: https://jvet.hhi.fraunhofer.de/

[139] T. Chris, R. Patrick, H. Xavier, and C. Attila. (2014) Opentrack: head tracking software. [Online]. Available: https://github.com/opentrack/opentrack

[140] D. V. Nguyen, H. T. Le, P. N. Nam, A. T. Pham, and T. C. Thang, “Adaptation method for video streaming over HTTP/2,” *IEICE Communications Express*, vol. 5, no. 3, pp. 69–73, Mar. 2016.

[141] X. Corbillon, F. De Simone, and G. Simon, “360-degree video head movement dataset,” in *Proc. ACM MMsys*. Taipei, Taiwan: ACM, 2017, pp. 199–204.

[142] D. Ochi, Y. Kunita, K. Fujii, A. Kojima, S. Iwaki, and J. Hirose, “HMD Viewing Spherical Video Streaming System,” in *Proc. ACM Multimedia*, New York, NY, USA, 2014, pp. 763–764.

[143] X. Jiang, Y.-H. Chiang, Y. Zhao, and Y. Ji, “Plato: Learning-based Adaptive Streaming of 360-Degree Videos,” in *IEEE Conf. on Local Computer Networks (LCN)*, 2018, pp. 393–400.

[144] T. Aykut, J. Xu, and E. Steinbach, “Realtime 3d 360-degree telepresence with deep-learning-based head-motion prediction,” *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, 2019.

[145] J. Park, P. A. Chou, and J.-N. Hwang, “Volumetric Media Streaming for Augmented Reality,” in *IEEE Global Communications Conf. (GLOBECOM)*, 2018, pp. 1–6.