Adaptive Streaming of Interactive Free Viewpoint Videos to Heterogeneous Clients

Ahmed Hamza
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada

Mohamed Hefeeda
Qatar Computing Research Institute
Hamad Bin Khalifa University
Doha, Qatar

ABSTRACT

Recent advances in video capturing and rendering technologies have paved the way for new video streaming applications. Free-viewpoint video (FVV) streaming is one such application where users are able to interact with the scene by navigating to different viewpoints. Free-viewpoint videos are composed of multiple streams representing the captured scene and its geometry from different vantage points. Rendering non-captured views at the client requires transmitting multiple views with associated depth map streams, thereby increasing the network traffic requirements for such systems. Adding to the complexity of these systems is the fact that different component streams contribute differently to the quality of the final rendered view. In this paper, we present a free-viewpoint video streaming system based on HTTP adaptive streaming and the multi-view-plus-depth (MVD) representation. We propose a novel quality-aware rate adaptation method for FVV streaming based on a virtual view distortion model. This view distortion model represents the relation between the distortion of the texture and depth components of reference views and a target virtual view and enables the streaming client to find the best set of representations to request from the server. We have implemented the proposed rate adaptation method in a prototype FVV DASH-based streaming system and performed objective and subjective evaluation experiments. Our experimental results show that the proposed FVV streaming rate adaptation method improves the user's quality-of-experience and increases the visual quality of rendered virtual views by up to 4 dB for some video sequences. Moreover, users have rated the quality of videos streamed using our proposed method higher than videos streamed using other rate adaptation methods in the literature.

CCS Concepts

•Information systems → Multimedia streaming;

Keywords

Free-viewpoint video; video streaming; rate adaptation; DASH; 3D video

MMSys'16, May 10–13, 2016, Klagenfurt, Austria.
© 2016 ACM. ISBN 978-1-4503-4297-1/16/05...$15.00
DOI: http://dx.doi.org/10.1145/2910017.2910610

1. INTRODUCTION

There has been a great interest recently in visual experiences beyond what is offered by traditional 2D video systems. 2D video streaming transmits a single view of the 3D world to viewers, delivering a limited experience. With recent advances in scene capturing technologies and virtual reality hardware, e.g., Oculus Rift and Microsoft HoloLens, interactive 360-degree videos and free-viewpoint videos (FVV) are increasingly gaining popularity. Moreover, recent 3D and multi-view displays offer more realistic, visually appealing viewing experience. For example, stereoscopic 3D displays present a different view to each eye to allow viewers perceive depth, and multi-view displays show a large number of views to viewers at different spatial locations for motion parallax.

Supporting interactive free viewpoint video streaming over the best-effort Internet to heterogeneous clients is challenging because of the high system complexity. For example, when viewers navigate across viewpoints in a free viewpoint system, some of the viewpoints may not have been captured by cameras. In that case, the non-captured view, referred to as a *virtual* view, needs to be synthesized using some of the captured views, referred to as *reference* views, by applying techniques like *Depth-Image-Based Rendering* (DIBR) [9]. DIBR generates the virtual view using two reference views, where each reference view consists of image and depth streams. As illustrated in Figure 1, the virtual view synthesis can be done on the server or the client. The server-based approach imposes computation, memory, and energy overheads on cloud servers, but suffers from additional network delay when viewers switch their viewpoints, leading to long response time. The client-based approach results in short response time, but requires the server to transmit at least four video streams (two image and two depth streams), which increases the volume of transmitted traffic. One possible way to reduce the traffic amount is to exploit the inherent redundancy between the captured views by using multi-view video coding [7] to encode *all* views. However, the interactive nature of free viewpoint videos renders a single multi-view stream less appealing, as only a small subset of reference views are required at any moment and the complex dependency in the prediction structure would hinder this. Therefore, server-based synthesis is more suitable to thin clients, such as mobile devices, and client-based synthesis is more suitable to powerful clients.

In 2D video streaming, a single video stream is transmitted from the server to the client. The quality of the decoded video is directly related to the compression-induced distortion of that single stream, which in turn is inversely proportional to the bitrate of the stream. Unlike 2D video streaming, in a FVV video streaming system, multiple video streams corresponding to the 3D components of different views are sent to the client and the rendered video frames are the result of a view synthesis process from the received compo-

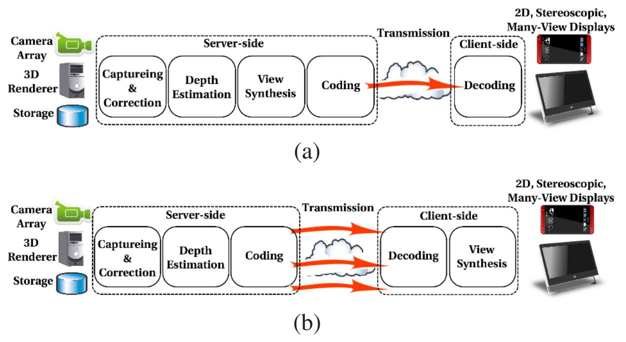


Figure 1: Free-viewpoint video streaming systems where view synthesis is performed at: (a) server and (b) client.

nents. This makes the problem of rate adaptation in these systems more complex because the quality of the rendered video stream is dependent on the qualities of the component streams used as references in the view synthesis process. Moreover, changes in the bit rates of those components do not equally contribute to the quality of the resulting video.

In this paper, we study the problem of optimizing adaptive streaming of interactive free-viewpoint videos to heterogeneous clients. More specifically, we address the problems of: (i) reducing the view-switching latency; and (ii) selecting the optimal versions for the components of reference views that maximize the quality of rendered virtual views. To this end, we present a two-step rate adaptation approach for FVV streaming systems. In the first step of the proposed approach, a number of reference views are scheduled for transmission based on the user's view navigation pattern. In the second stage, the streaming client determines the optimal bit rate allocation for the chosen component streams in order to deliver the best possible quality for rendered virtual views given the available network bandwidth. We implement the proposed approach in a real DASH-based free-viewpoint video streaming testbed. DASH (Dynamic Adaptive Streaming over HTTP) [14] is a flexible and popular standard for implementing adaptive streaming systems.

The rest of this paper is organized as follows. Section 2 summarizes the related works in the literature. Section 3 presents our proposed rate adaptation approach for FVV streaming systems. Section 4 briefly describes the design of a complete DASH-based FVV streaming system including a our prototype implementation of an FVV streaming client. Section 5 presents the experimental evaluation of the proposed rate adaptation approach using our prototype system in different network environments. Section 6 concludes the paper.

2. RELATED WORK

A number of research works in the literature attempted to address the problem of interactive multi-view and FVV streaming. These works can be classified into two classes: server-based non-adaptive approaches and client-based adaptive streaming approaches.

2.1 Server-based Approaches

An interactive multi-view video system based on the idea of transmitting residual information to aid the client in generating virtual views is presented in [19]. In the proposed system, the encoder-side performs view synthesis and computes the necessary residual information which is then stored and transmitted to the user when needed. This increases the storage and bandwidth requirements of the server and it may affect the scalability of the system because the server needs to prepare and transmit user-specific information for each incoming request. Kurutepe et al. propose a selective streaming system based on multi-view video coding and user head tracking [17]. In this system, the client sends the current viewpoint to the server over a feedback channel. The server then encodes and streams a multi-view video sequence containing a stereo pair corresponding to the user's viewpoint and two lower resolution side views to reduce the view-switching latency. An enhancement layer is simulcast coded for the stereo pair to improve the quality of the selected views. Such a system requires performing the complex and time consuming multi-view coding process on-the-fly for each client. This increases the server's processing load and does not scale well with a large number of clients requesting different views. Moreover, because the system does not perform view synthesis, the users are restricted by the viewpoints available at the server and a smooth view-switching experience can only be achieved when a large number of captured views are available at the server, thereby increasing the storage overhead.

2.2 Client-based Approaches

Xiao et al. [27] present two approaches to streaming multi-view videos over DASH. The focus of [27] is providing timely view switching without playback interruptions. Unlike our proposed system, their work only considers multi-view videos with no depth information. The main idea of these two approaches is to utilize multi-view encoders performing inter-view prediction to reduce the view switching latency. In their system, different versions of both simulcast coded and inter-view coded views are stored on the server. Moreover, in the first approach, all possible version combinations for inter-view coded streams are generated. This imposes a large storage overhead on the content server and significantly increases the cost on the content provider. In [10], Gao et al. present a multi-modal 3D video streaming system based on the DASH standard which allows users to view arbitrary sides of a captured object. Although their work is somewhat similar to ours, the authors mainly focus on supporting multi-modal data and do not provide details on how rate adaptation across the different modalities is performed.

The two works that are closely related to ours are [24] and [12]. Su et al. [24] present an HEVC multi-view streaming system using DASH. Similar to our proposed system, the streamed video is represented using a number of views and associated depth maps. Unlike our proposed system, however, the client adapts the bit rate of the video by leveraging view-scalability where the number of transmitted reference views (and possibly the distances between them) is varied based on the available network bandwidth. Because the views and their depth streams are jointly coded, the video data for all components are encoded into a single stream. This dictates a fixed distribution of the total segment bit rate between the different components. In [24] all segment components for a given representation have an equal bit rate. Therefore the system does not provide much flexibility in terms of rate adaptation and does not consider how each component stream contributes differently to the qualities of the synthesized views.

In our previous work [12], we presented a FVV streaming system that is based on *empirical* rate-distortion (R-D) models which relate bit rates of the reference views and the quality of synthesized views. Creating such empirical model requires generating a synthesized view for each combination of representations for the pairs of reference views and measuring the average distortion in each iteration. Assuming an MVD video with M captured views, this requires $(M-1)KL^4$ decode-synthesize iterations. In addition, these empirical models need to be communicated to the client at the beginning of the streaming session. Adding L^4 values for each segment index in the MVD video's MPD file incurs additional overhead and delays the start-up of playback. This is especially the case for long duration videos with a large number of segments. In Section 5, we compare our proposed system against the approaches in [24] and [12] and show that it results in near optimal quality without the large overhead associated with generating and communicating empirical models.

3. PROPOSED FVV ADAPTIVE STREAMING SYSTEM

In this section, we start by defining the rate adaptation problem in DASH-based FVV streaming systems addressed in this paper. Then, we present our two-step approach to solving it.

3.1 Problem Definition

A content server stores a number of free-viewpoint videos in which scenes are captured from multiple views. Each captured view has a corresponding depth map stream representing the depth value of each pixel in the captured frames. The texture and depth streams for each view are simulcast coded with the same encoding configuration to obtain a multi-view-plus-depth (MVD) representation of the scene. Each component stream is encoded at L different quality levels (representations). The resulting streams are each divided into a set of segments with equal playback duration τ. A single manifest file describing the component streams as well as metadata information related to the captured reference views is stored at the server.

Let $\mathbf{V}$ be a set of evenly spaced captured views, where $|\mathbf{V}| = N$. We assume that an equal number of evenly spaced virtual views, say K, are available for view navigation between each two adjacent captured views i and $i+1$. In this paper, we refer to the set of virtual views between two adjacent captured views as the *virtual view range*. The set of views that a user can navigate to is denoted $\mathbf{V}'$, where $|\mathbf{V}'| = N + K(N-1)$, and the views are separated by a distance of $d = 1/(K+1)$. We can express a viewpoint position as a multiple of the inter-view distance, i.e., a view with index j is at position $k = j \cdot d$.

The problem of virtual view quality-aware rate adaptation that we attempt to address in this paper can therefore be stated as follows:

PROBLEM 1. *Consider a free-viewpoint video where a number of reference views composed of texture and depth component streams encoded at L different representations are stored on the server. Given the current viewpoint position and the available network bandwidth between the server and client, determine which reference views should be requested and which representations for each texture and depth component should be downloaded such that the quality of the rendered virtual views at the client side is maximized.*

To solve this problem, we propose a two-step approach. In the first step, the client determines the set of reference views it needs to request from the server in order to render the current viewpoint as well as any potential viewpoints that the user may navigate to in the future. In the second step, the client's rate adaptation logic should decide on the representations for each of the segments of the scheduled views' components. In the following subsections, we discuss the two steps of the proposed approach in more detail.

3.2 Reference View Scheduling

In a FVV streaming client, the user expects to be able to navigate freely to any desired viewpoint. The view navigation pattern depends on the nature of the video and the interests of the user. It may happen that the rate at which the user is changing views causes the viewpoint to change to a position that lies outside the virtual view range of the buffered reference views before the current segment duration ends. When the user navigates to a viewpoint outside the virtual view range of the reference views currently in the buffer, the client needs to replace one of the reference views by downloading another segment for a view that bounds the new virtual view range in which the requested virtual view lies. During the download time of the new reference view, the client can either perform view synthesis using only one of the available references, resulting in sudden quality degradation, or wait for a segment from the new reference view to be available, which results in high view switching latency. A FVV streaming client therefore requires a reference view scheduling component that determines which reference views should be downloaded based on the viewer's view-switching behaviour.

To reduce the reference view switching latency, our FVV streaming client conditionally pre-fetches an additional reference view based on the current and previous viewpoint positions of the user. This is achieved by periodically recording the viewpoint position of the user and using a navigation path prediction technique to extrapolate the viewpoint position of the user based on historical measurements. We propose using a simple location estimation technique known as *dead reckoning* [23, Ch.5]. Dead reckoning calculates an object's current position by using a previously determined position, or fix, and advancing that position based on known or estimated velocities over a duration of elapsed time and course. By tracking the user's viewpoint positions, dead reckoning can enable the client to predict the future path by assuming that the user maintains the current view-switching velocity. We divide the time into discrete instants with interval Δ, where $\tau = \zeta\Delta$ and ζ is fixed value. Let $x(t)$ be the view position at time instant t. We can therefore calculate the instantaneous view-switching velocity $v(t)$ using

$$v(t) = (x(t) - x(t-\Delta))/\Delta. \tag{1}$$

Knowing the view-switching velocity also enables the client to determine whether pre-fetching an additional reference view is necessary for the next segment duration, depending on how fast the view-switching is. Based on the calculated view-switching velocity, the client can predict the view position at the beginning of the next segment as

$$x(t+\tau) = x(t) + v(t) \cdot \tau. \tag{2}$$

If the estimated position is not within the current virtual view range, the view scheduler will schedule an additional reference view that, along with one of the current reference views, bounds the estimated viewpoint position. To obtain a more accurate prediction of the viewpoint position, we apply a smoothing filter, such as the exponentially weighted moving average (EWMA), to either the predicted position or the view-switching velocity. The smoothed out view-switching velocity $v'(t)$ can be calculated as

$$v'(t) = \theta \cdot v(t) + (1-\theta) \cdot v'(t-\tau), \tag{3}$$

where $\theta \in [0, 1]$ is the smoothing factor.

Figure 2 shows an example of a set of viewpoint positions over time, the estimated view navigation path, and the corresponding view scheduling window. The reference view scheduling method can be summarized as follows:

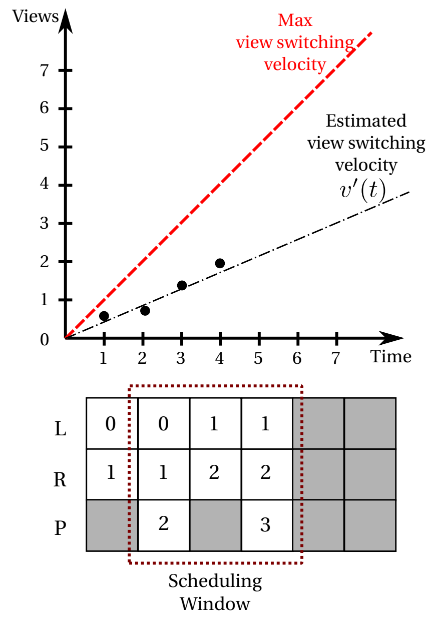

Views
Max view switching velocity
Estimated view switching velocity
$v'(t)$
Time
L
R
P
Scheduling Window

Figure 2: Segment scheduling window. Deciding on left (L) reference view, right (R) reference view, and pre-fetched (P) view.

- The streaming client maintains a view scheduling window that stores decisions about which views are requested for each segment index within the window. The length of the window is kept relatively small, e.g., three segments, to reduce the bandwidth overhead in the case of inaccurate predictions in the viewpoint position.

- During one segment duration, the view scheduler records viewpoint position changes and updates the direction and velocity of view-switching based on Eq. (1) and Eq. (3). The estimated viewpoint position for the next segment index to be downloaded is then calculated based on Eq. (2). We note that the system may enforce a maximum view-switching velocity to ensure that at least one of the buffered views would be an immediate reference at the time of rendering.

- After the download of all component segments of the current segment index has completed, a view scheduling decision is made based on the estimated viewpoint position and the view scheduler slides the window by one segment duration. A scheduling decision includes the index of the left (L) and right (R) reference views and, if necessary, an additional view to be pre-fetched (P).

Because the view scheduler schedules views for a number of segment indices in the future, it is possible that some scheduling decisions may not be valid after the scheduling window slides due to some unpredictable behaviour of the user. In such cases, the client will keep any previous decisions for component segments of that segment index for which no download has been initiated. When playback reaches that particular segment index, the streaming client attempts to utilize the available views to perform view synthesis.

3.3 Virtual View Quality-Aware Rate Adaptation

We propose a new rate adaptation algorithm for DASH-based FVV streaming systems. The algorithm creates a virtual view quality-distortion model to estimate the quality of the synthesized view based on the qualities of the reference views' components. Using a virtual view distortion model, the adaptation module can quickly calculate the expected distortion of each supported virtual view given a certain *operating point* for the immediate reference views. An operating point is a combination of representations for the components of the scheduled reference views. The rate adaptation module then chooses the representations corresponding to the operating point which minimizes the expected distortion over a set of virtual views and satisfies the available network bandwidth.

In the following, we derive a relation for the virtual view distortion based on the qualities of the reference views' components. We note that our system can also support other virtual view distortion models provided that the model gets signalled within the MPD file. For example, models such as [8] and [26] can be adapted and used in our system.

Let S_v be the virtual image synthesized by original (uncompressed) texture images and original depth maps, $\bar{S}_v$ is the virtual image synthesized by the original texture images and compressed depth maps, and $\hat{S}_v$ is the virtual image synthesized by the compressed texture images and the compressed depth maps. The distortion of a synthesized virtual view, in terms of *mean squared error* (MSE), can be expressed as

$$\begin{aligned} D_v &= E[(S_v - \hat{S}_v)^2] \\ &= E[\{(S_v - \bar{S}_v) + (\bar{S}_v - \hat{S}_v)\}^2] \\ &= E[(S_v - \bar{S}_v)^2] + E[(\bar{S}_v - \hat{S}_v)^2] \\ &\quad + 2E[(S_v - \bar{S}_v)(\bar{S}_v - \hat{S}_v)] \\ &\approx E[(S_v - \bar{S}_v)^2] + E[(\bar{S}_v - \hat{S}_v)^2], \end{aligned} \tag{4}$$

where $E(\cdot)$ represents the expectation taken over all pixels in one image.

In Eq. (4), $E[(S_v - \bar{S}_v)^2]$ represents the view synthesis distortion induced by depth map compression and original texture video, and $E[(\bar{S}_v - \hat{S}_v)^2]$ represents the view synthesis distortion induced by texture video compression and decoded depth maps. Note that the term $2E[(S_v - \bar{S}_v)(\bar{S}_v - \hat{S}_v)]$ can be neglected since the distortions induced by texture video and depth map compression are not correlated [29].

DIBR view synthesis algorithms are based on the concept of 3D image warping [20]. 3D image warping maps pixels from the original view to the correct positions in the desired view based on their corresponding depth values. To minimize the quality degradation resulting from unoccluded regions appearing in the virtual view, 3D warping based DIBR algorithms generally use two reference views, the left and right adjacent camera views, to synthesize the virtual image. This process in known as *double-warping*. The virtual view synthesis is expressed as

$$I_V(x_V, y_V) = \omega_L I_L(x_L, y_L) + \omega_R I_R(x_R, y_R), \tag{5}$$

where $I_V(x_V, y_V)$, $I_L(x_L, y_L)$, and $I_R(x_R, y_R)$ are the pixel values of matching points in the virtual view, left reference view, and right reference view, respectively, and ω_L and ω_R are distance-dependent *blending weights* satisfying $\omega_L + \omega_R = 1$. Hence, the virtual view distortion can be expressed as [28]

$$D_v = \omega_L^2 D_v^L + \omega_R^2 D_v^R, \tag{6}$$

where D_v^L and D_v^R are the virtual view distortions induced by the left and right reference views, respectively, and each can be modeled using Eq. (4).

Using power spectral density (PSD) and Gaussian modeling of the depth map [21], the term $E[(\bar{S}_v - \hat{S}_v)^2]$ in Eq. (4) can be modeled by

$$E[(\bar{S}_v - \hat{S}_v)^2] = \varrho \cdot E[\Delta P_L^2], \tag{7}$$

where ϱ is a linear parameter associated with the texture image contents of the reference view and represents the motion sensitivity factor [18], and ΔP is the warping position error of the reference view. The warping position error at point (x, y) can be formulated as

$$\Delta P(x,y) = \frac{f\delta}{255}\left(\frac{1}{Z_{\text{near}}} - \frac{1}{Z_{\text{far}}}\right) e(x,y), \tag{8}$$

where f is the focal length of the cameras, δ represents the horizontal distance between the virtual viewpoint and the left reference view and right reference view, $e(x,y)$ corresponds to the error between the original and the compressed depth map at point (x,y). Z_{near} and Z_{far} are the values for the nearest and farthest depth in the scene, respectively. Hence, from Eq. (7) and Eq. (8), the depth compression induced distortion can be represented by

$$\begin{aligned} E[(\bar{S}_v - \hat{S}_v)^2] &= \Phi\delta^2 E[(e(x,y))^2]\varrho \\ &= \Phi\delta^2 D_d \varrho, \end{aligned} \tag{9}$$

where D_d is the compression distortion for the depth component of the reference view, and Φ is a constant expressed as

$$\Phi = \left[\frac{f}{255}\left(\frac{1}{Z_{\text{near}}} - \frac{1}{Z_{\text{far}}}\right)\right]^2. \tag{10}$$

The term $E[(\bar{S}_v - \hat{S}_v)^2]$ in Eq. (4) can be considered as the compression distortion for the texture component of the reference view. Therefore, Eq. (6) can be re-written as

$$\begin{aligned} D_v &= \omega_L^2(D_t^L + \Phi\delta_L^2 D_d^L \varrho_L) + \omega_R^2(D_t^R + \Phi\delta_R^2 D_d^R \varrho_R) \\ &= \lambda D_t^L + \mu D_d^L + \nu D_t^R + \xi D_d^R + c. \end{aligned} \tag{11}$$

It is therefore sufficient to find the values of the coefficients λ, μ, ν, ξ, and the constant c in Eq. (11) for each supported virtual viewpoint and communicate them to the client. In order to obtain the values of those coefficients, we use the multiple linear least squares regression function in Matlab [3] and a small set of sample of R-D points to solve the system of linear equations given in Eq. (12). The values of these coefficient are then added to the MPD file in a `VVRDModel` element, as shown in the example in Listing 1 for one segment and one virtual view range with three virtual view positions.

$$\begin{pmatrix} D^L_{t\,1} & D^R_{t\,1} & D^L_{d\,1} & D^R_{d\,1} & 1 \\ D^L_{t\,2} & D^R_{t\,2} & D^L_{d\,2} & D^R_{d\,2} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ D^L_{t\,n} & D^R_{t\,n} & D^L_{d\,n} & D^R_{d\,n} & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} = \begin{pmatrix} D_{v1} \\ D_{v2} \\ \vdots \\ D_{vn} \end{pmatrix} \tag{12}$$

The rate adaptation logic therefore proceeds as given in Algorithm 1, where $R(p)$ is the total bit rate for operating point p, and $D(p,\alpha)$ is the corresponding distortion at virtual view position α. After deciding on a reference view schedule (Section 3.2), the client invokes the rate adaptation logic to determine how the available bandwidth will be distributed amongst the video components of the scheduled views. In the case of stationary viewing where the user does not navigate much around a certain viewpoint, the client will only schedule two reference views and the rate adaptation algorithm evaluates for each operating operating the corresponding average estimated distortion of the virtual views between the two reference views. The representations corresponding to the operating point which results in minimal distortion are then chosen by the algorithm.

For segment durations where an additional view will be pre-fetched, a few minor modifications to the algorithm are required. In the case of a scheduled pre-fetch view, an operating point p will involve six components instead of four: four components for the left and right reference views, and two for the pre-fetch view. The rate adaptation logic will first calculate the average distortion for the virtual views in the current view range (D^c_{avg}) and the average distortion for the expected view range separately (D^e_{avg}). Hence, lines 4 to 7 in Algorithm 1 will be repeated for the expected view range. Since the viewer will navigate between two virtual view ranges, the value assigned to D_{avg} in line 8 will be replaced with the weighted sum $(1-\beta)D^c_{\text{avg}} + \beta D^e_{\text{avg}}$, where β determines the likelihood of the user navigating to the expected view range. When β equals 0.5, the weighted sum becomes the average distortion over all virtual views of the two ranges.

Listing 1: Model parameters in the MPD file (attributes a_0, a_1, a_2, a_3, and a_4 represent model coefficients λ, μ, ν, ξ, and c, respectively).

```
<VVRDModel metric="psnr">
<SegmentVVRDModel segId="11">
  <VVRange id="1" l="3" r="5">
    <VVParam alpha="0.25" a0="0.24" a1="0.27"
        a2="0.08" a3="0.03" a4="11.99" />
    <VVParam alpha="0.50" a0="0.23" a1="0.22"
        a2="0.05" a3="0.05" a4="14.07" />
    <VVParam alpha="0.75" a0="0.24" a1="0.19"
        a2="0.02" a3="0.07" a4="14.06" />
  </VVRange>
</SegmentVVRDModel>
</VVRDModel>
```

Algorithm 1: FINDBESTOPERATINGPOINT

```
Input: Set P of operating points for left and right reference views qualities
Input: Bandwidth constraint $R_c$
Input: Set A of $\alpha$ values corresponding to $K$ virtual view positions
Output: Operating point $p^*$ which minimizes the average distortion over all virtual views
1  $p^* \leftarrow \phi$, $D_{\text{min}} \leftarrow \infty$
2  foreach $p \in \mathbf{P}$ do
3      if $R(p) \leq R_c$ then
4          $D_{\text{sum}} \leftarrow 0$
5          for $i \leftarrow 0$ to $K$ do
6              $\alpha \leftarrow A(i)$
7              $D_{\text{sum}} \leftarrow D_{\text{sum}} + D(p,\alpha)$
8          $D_{\text{avg}}(p) \leftarrow D_{\text{sum}}/K$
9          if $(i == 0)$ OR $(D_{avg}(p) < D_{min})$ then
10             $D_{\text{min}} \leftarrow D_{\text{avg}}(p)$
11             $p^* \leftarrow p$
12             continue
13 if $p^* == \phi$ then
14     $p^* \leftarrow$ lowest quality representations
15 return $p^*$
```

4. FVV STREAMING SYSTEM DESIGN

The architecture of our DASH-based free-viewpoint video streaming system is shown in Figure 3. In this section, we describe a complete system that implements the proposed virtual view quality-aware rate adaptation algorithm. Our system contains two main entities: content server, and FVV streaming client, which are described in the following.

4.1 Content Server

The free-viewpoint videos are captured using a camera array which capture the scene from multiple viewpoints. To generate the depth information, an additional depth camera may be provided for each viewpoint of the camera array. Alternatively, depth maps can be generated at a later time using one of the known depth estimation methods [22]. The content server stores the videos in the MVD representation format, where each captured view is composed of two separate streams: a texture stream, and an associated depth stream. We refer to these streams as the *components* of the view. The server also runs a standard HTTP Web server process that handles requests from DASH clients.

Similar to 2D-based DASH streaming systems, each component is encoded at different bit rates (quality levels) using standard 2D video codecs, such as H.264/AVC and HEVC [25]. To synchronize the different component videos, the same frame rate and group-of-pictures (GOP) size are used for all components. The resulting streams constitute different *representations* of the component at different qualities. The representations are then segmented according to the DASH standard [14] to generate segments of equal duration. To support the view synthesis process and rate adaptation logic on the client-side, a *media presentation descriptor* (MPD) file describes the different views and components of the MVD content. This file contains information information about the different representations of each component as well as the camera parameters and quality models for supported virtual views. We now describe the proposed MPD structure utilized by our FVV streaming system.

MPD Structure. In DASH, the MPD file provides the client with a description of all available components of the media content as well as per-segment and per-representation information which enable the client to make adaptation decisions at each segment download time. In our FVV streaming system, each `Period` element in the MPD file is divided into three sections: component adaptation sets, camera parameters, and virtual view quality models. We note here that each `Period` element represents a scene within the MVD content and that each scene may be captured by a different camera arrangement and, therefore, may contain a different number of captured views. Without loss of generality, we assume in the following an MVD video with a single `Period` element for simplicity. Unlike traditional single view 2D content which contains only a single video stream, MVD content contains multiple video streams (one for each component of each view). Therefore, in our proposed system, each texture or depth component of a captured view will have its own `AdaptationSet` element within the `Period` elements of the MPD file. To uniquely identify each view, we use a `Viewpoint` descriptor element within the `AdaptationSet` elements of each of the view's components. To identify the type of the component, we use the `Role` descriptor element with `@value="t"` for texture streams and `@value="d"` for depth streams.

The MPD file needs additional information about the MVD video sequence to support client-side view synthesis. This information includes the *intrinsic* and *extrinsic* parameters of the cameras capturing the scene, as well as the values of the closest and furthest depth values. We use a `CameraParameters` element, as shown in Listing 2, to signal the camera parameters for each captured view. Each captured is given a sequential identifier based on a left-to-right order of the views and is represented using a `View` child element.

Listing 2: Camera parameters element in MPD.

```
<CameraParameters>
<View id="0">
 <IntrinsicParam fx="2241.26" fy="2241.26"
     cx="701.5" cy="514.5" />
 <ExtrinsicParam rotation="1,0,0,0,1,0,0,0,1"
     translation="5,0,0" />
 <ZRange zNear="448.2512" zFar="11206.2803" />
</View>
</CameraParamters>
```

Listing 3: Texture component with two representations.

```
<AdaptationSet mimeType="video/mp4"
    codecs="avc1.640828">
  <Viewpoint schemeIdUri="urn:mpeg:dash:mvv:2014"
      value="0"/>
  <Role schemeIdUri="urn:mpeg:dash:v+d:2014"
      value="t"/>
  <Representation bandwidth="128000"
      avgPSNR="34.1" avgSSIM="0.959">
   <SegmentList duration="1">
       <Initialization
           sourceURL="oblivion_128_t0_init.mp4"/>
       <SegmentURL
           media="oblivion_128_t0_seg1.m4s"/>
      </SegmentList>
   </Representation>
</AdaptationSet>
```

Since our rate adaptation module utilizes a distortion model to estimate the quality of virtual views, it is necessary that the streaming client gains access to quality information for the component streams. A number of MPEG proposals, e.g., [15], were recently presented for signalling quality information in DASH. Quality information can be signalled either at the MPD level or the media container level. In the former case, the MPD would contain additional *metadata sets* with metadata representations having associations to corresponding media representations in the adaptation sets. Therefore each metadata segment is always associated with a media segment and both are time aligned. The association between the different metadata elements and their corresponding media elements within the MPD can be achieved by sharing the same id values, for example. Alternatively, communicating quality information to the client can be achieved using metadata tracks within the media container file format itself. This, however, requires modifications to the demuxers used by the decoder to support the syntax of the additional quality metadata tracks. We use a simpler approach in which the average quality of a component stream is provided using additional attributes in the corresponding `Representation` element. For example, each video `Representation` element in the MPD file may have *avgPSNR* and *avgSSIM* attributes holding the average values of the PSNR and SSIM quality metrics, respectively. Listing 3 provides an example of an adaptation set for the texture component of one of the captured views with one sample representation.

4.2 FVV Streaming Client

The FVV streaming client is composed of a number of modules: *MVD-DASH Manager*, *Segment Downloader*, *Segment De-*

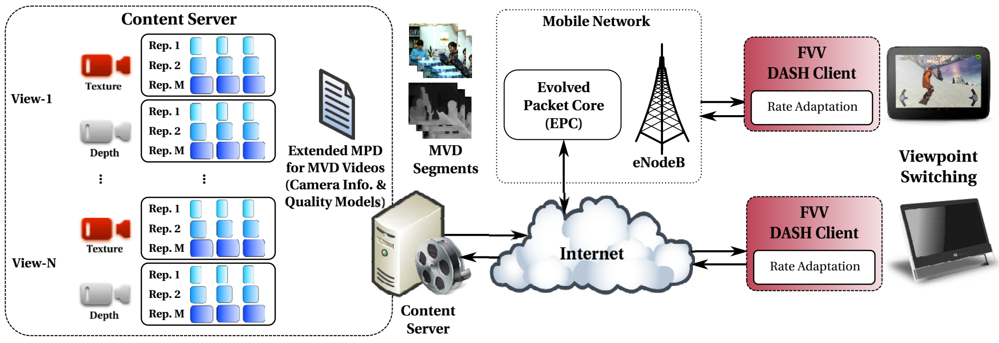


Figure 3: Overview of the proposed adaptive free-viewpoint video streaming system.

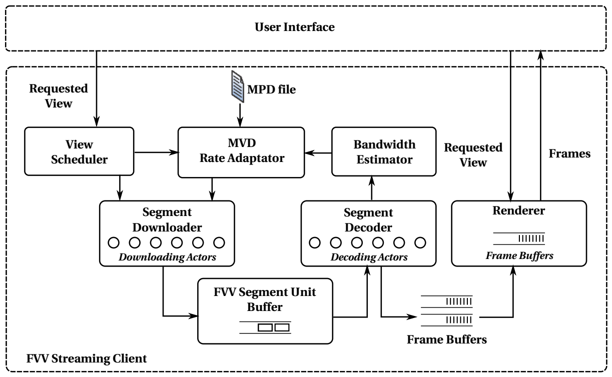
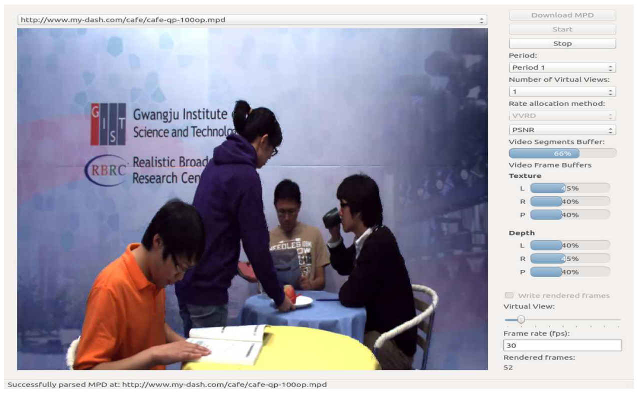


Figure 4: The components of our FVV streaming client.



http://www.my-dash.com/cafe/cafe-qp-100op.mpd

Period: Period 1
Number of Virtual Views: 1
Rate allocation method: VVRD
PSNR
Video Segments Buffer: 66%
Video Frame Buffers
Texture
L 45%
R 40%
P 40%
Depth
L 40%
R 45%
P 40%
Write rendered frames
Virtual View:
Frame rate (fps): 30
Rendered frames: 52

Successfully parsed MPD at: http://www.my-dash.com/cafe/cafe-qp-100op.mpd

Figure 5: The user interface of our FVV client prototype.

coder, *View Scheduler*, *MVD Rate Adaptor* and *Renderer*, as illustrated in Figure 4. Our streaming client maintains a buffer of *multi-component segments*. A *multi-component segment* is a logical container for the segments of the components of the reference views, possibly including a pre-fetched view. All segments within a multi-component segment have the same duration and correspond to the same segment index and time duration in the presentation timeline. The client is implemented using the *actor*-based concurrent programming model which relies on message passing between the different actors. Because actors do not share state and messages are sent asynchronously, the different modules will not compete for locks which significantly increases the performance of the streaming client. Figure 5 shows the user interface of a prototype implementation of our client. In the following, we discuss the roles of the various modules.

Renderer. The renderer generates the final frames that will be displayed to the user. Based on the user's current viewpoint, this may require performing view synthesis to generate a virtual view if the requested viewpoint is not at a captured view position. The renderer contains six frame buffers: two for the components of the left reference view, two for the components of the right reference view, and two for the components of the pre-fetched view. For uninterrupted playback, the levels of the frame buffers are maintained above a certain threshold. Whenever the level of one of the frame buffers drops below the threshold, the renderer requests a batch of frames from the controller to avoid having buffer underflows. If the user's viewpoint falls at a virtual view position, the renderer performs depth-based view synthesis to generate the target view using available reference views frames. It is important that the chosen view synthesis algorithm can run in real-time in order to keep up with the frame rate of the 3D video. We developed a DIBR implementation which exploits graphics processing units (GPUs) to speed up the view synthesis process for 1-D parallel camera arrangements, where cameras are aligned in a straight line perpendicularly to their optical axes. The rendering module uses the OpenGL graphics API [5] to perform the different stages of the view synthesis process. The frames are uploaded to the GPU's memory and shader programs perform the warping, blending, and hole-filling steps. Our implementation achieved a 30 fps frame rate for full high definition (HD) resolution on an NVIDIA GeForce 560 Ti GPU.

Segment Downloader. The segment downloader holds references to a number of download actors. When a segment download request is received, the request is forwarded to one of the available actors and that actor is responsible for fetching the contents of the segment from the content server. Each download actor maintains a persistent HTTP connection with the server to reduce delays and overhead associated with establishing a new TCP connection for each segment request. The segment downloader is also responsible for keeping track of the download start and finish times for each component segment and reports this information back to the MVD-DASH manager to estimate the available channel bandwidth.

Segment Decoder. Similar to the segment downloader, this module maintains a number of actors which are responsible for decoding individual component segments. It receives decode messages containing a downloaded DASH segment for a texture or depth component of one of the views and forwards it to one of the decoding actors. Decoding actors launch decoding threads for each of the segments and decoded frames are returned to the controller for buffering.

View Scheduler. The view scheduler maintains a scheduling window which determines the reference views to be downloaded within a time window. View scheduling decisions are based on predictions of future viewpoints, taking into consideration the current viewpoint of the user as well as historical view navigation positions. The module implements the dead reckoning-based view scheduling method described in Section 3.2.

MVD Rate Adaptor. This module is responsible for responding to variations in network conditions by allocating the available bandwidth between the component segments of the views scheduled for download. The rate adaptor takes advantage of virtual view distortion models and implements the proposed rate adaptation algorithm described earlier in Section 3.3.

MVD-DASH Manager. This module is the controller which orchestrates the interaction between the different components of the player. When a multi-component segment is to be downloaded, the manager invokes the MVD rate adaptor to decide on the set of component representations that will be requested from the server. Based on the decision returned from the adaptation module, the MVD-DASH manager sends a segment download message with the chosen representations to the segment downloader. The downloaded DASH segments for scheduled reference views belonging to the same segment index are aggregated into multi-component segments which are then placed into the segment buffer. Each multi-component segment fetched from the segment buffer is sent to the segment decoder module for decoding. Similar to the rendering module, the controller also maintains six frame buffers to hold decoded segment frames. In the case where only two reference views are being requested, i.e., without a pre-fetch view, the decoders for the pre-fetch view components generate dummy frames for those components to synchronize the number of frames across the buffers.

5. EMPIRICAL EVALUATION

5.1 Experimental Setup

The proposed FVV streaming client is implemented using C++ and *libdash* [2], an open-source library which implements the MPEG-DASH standard as defined by ISO/IEC 23009-1 [14]. The implementation includes the proposed virtual view quality-aware R-D-based rate adaptation algorithm and reference view scheduling method.

To evaluate the proposed FVV client and R-D-based rate adaptation, we used three MVD sequences from the MPEG 3DV adhoc group data sets [13] [4]: Kendo, Balloons, and Café, which have different characteristics. The resolution for the Kendo and Balloons sequences is 1024×768 and the resolution of the Café sequence is 1920×1080. The Kendo and Balloons sequences have moving cameras while the cameras in Café are fixed. We extended the length of the video sequences from 10 to 30 seconds by repeating the frame sequence. For each MVD video, we chose three cameras from the set of captured views and we allow three virtual views within each virtual view range, for a total of 6 supported virtual view positions. The video streams for the texture and depth components of each camera were then encoded using two configurations. In the first configuration, we use the variable bit rate (VBR) setting of the H.264/AVC encoder with quantization parameter values ranging from 24 to 44 with a step of 4. In the second configuration, constant bit rate (CBR) encoding was used with bit rate values ranging from 250 Kbps to 1.5 Mbps with a step of 250 Kbps. We used the GPAC framework [1] to generate one second segments for the different representations of each component.

For each segment index, we generate virtual view quality models

Table 1: Coefficient of determination and average absolute fitting error for virtual view quality models generated from 100 operating points at view position 2 of the Kendo and Balloons sequences (encoded using VBR).

Seg. Index	Kendo		Balloons	
	R^2	Avg. Error	R^2	Avg. Error
11	0.9780	0.1787	0.9721	0.1643
12	0.9765	0.1863	0.9770	0.1474
13	0.9796	0.1707	0.9736	0.1598
14	0.9756	0.1738	0.9789	0.1537
15	0.9722	0.1722	0.9798	0.1530

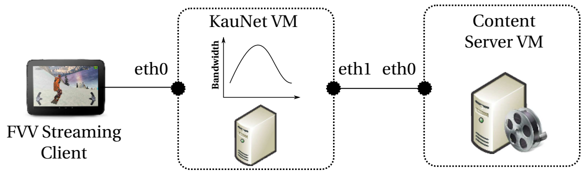


Figure 6: Evaluation testbed.

for all supported virtual view positions as described in Section 3.3. We generate two quality models for each virtual view position: one based on 100 operating point samples, and one based on 40 samples. To validate the generated quality models, we calculate the coefficient of determination (R^2) and average absolute fitting error. Table 1 shows the results for the virtual view at position 2 for 5 segments of the Kendo and Balloons video sequences. The results indicate that the derived virtual view quality models are good fits for the empirical quality values obtained for all operating points of the encoded video sequences. Similar results were obtained for the Café video sequence and for other virtual view positions and segment indices. We also developed Python scripts to parse the MPDs of the component streams and virtual view model parameters files and generate a single MPD file for the MVD video based on the structure described in Section 4.1. In the evaluation experiments, the client's multi-component segment buffer capacity was set to 3 segments and the rendering module's frame buffer capacity to 300 frames.

Our testbed setup is shown in Figure 6. The server storing and streaming the content is a virtual machine (VM) running the Apache2 Web server. The second VM running FreeBSD 7.3 and KauNet network emulator [11] is placed between the server and the client and is configured with two network interfaces. The two VMs run on the same physical machine where the streaming client is running. Two virtual networks are set up to connect the two interfaces to the server and the client, respectively. The KauNet VM is responsible for controlling the available bandwidth between the client and the server based on input bandwidth change patterns. A trigger pattern is configured to send a periodic signal to the client to synchronize the start of the streaming session with beginning of the bandwidth change pattern.

We perform two sets of experiments to evaluate the performance of our proposed rate adaptation method. In the first set of experiments, we compare the quality of our approach against the rate adaptation strategy used in [24] and the optimal rate allocation obtained using [12] using objective video quality metrics. To obtain a fair comparison, the bandwidth estimates resulting from a playback run using the equal allocation strategy [24] are recorded and used as input to the rate adaptation logic in subsequent runs. We first evaluate the client's behaviour at a fixed network bandwidth and fixed viewing point (the center virtual view position of the first virtual view range). We then assess the behaviour of our streaming client when the viewpoint is fixed while the bandwidth is varied. This enables us to assess the response of the virtual view rate-distortion allocation algorithm in isolation from the view switching prediction logic. In the second set of experiments, we conduct a subjective quality assessment study of the results. In these experiments, subjects were asked to compare the qualities of virtual view videos generated using our proposed approach and the approach presented in [24].

5.2 Objective Quality Results

5.2.1 Fixed Bandwidth

For the fixed bandwidth experiments, we set the value of the available network bandwidth to certain value and repeat the streaming session using a different rate adaptation method in each run. The bandwidth values used are 1, 2, 4, 5, and 6 Mbps. We fix the view angle at the middle viewpoint between the first two captured camera views (view 2 for Balloons and Kendo, and 2.5 for Café). Figures 7 and 8 demonstrate the the resulting average virtual view quality, in terms of peak signal-to-noise ratio (PSNR), for segments 11 to 20 of the Balloons and Café videos, respectively, using CBR-encoding. The quality of the rendered virtual view is measured against the virtual view synthesized at the same position using the original uncompressed reference streams. In the figures, **Opt.** refers to the optimal quality using [12], **Equal** refers to the equal rate allocation strategy used in [24], and **VVRD 40** and **VVRD 100** refer to our proposed approach where the numbers indicate the number of samples used to generate the virtual view quality model. It can be seen that even at low bandwidth conditions, our rate adaptation approach is able to achieve significant quality gains (up to 4 dB for Balloons and 2.8 dB for Café). We notice that the improvement achieved by our algorithm is more significant when the bandwidth is not too large (around 2 Mbps), which is the common case. For larger bandwidth values (e.g., more than 6 Mbps), there is little room for optimization and most adaptation algorithms would yield similar or very close qualities. By comparing the results for **VVRD 100** and **VVRD 40**, we can see that the quality improvements can be achieved even when a small number of operating points are used to obtain the model coefficients, which means that our algorithm does not impose significant computational overheads on the servers. The model coefficients are computed only once for each video (not with every streaming session of the same video) and stored in metafiles. Similar improvement gains were achieved using the Kendo sequence.

Our proposed virtual view quality-aware rate adaptation method also results in quality improvements when the reference views are encoded using VBR. Firgure 9 shows the average virtual view quality for the Balloons sequence when the reference views are encoded using VBR. Our method improves the quality of the rendered stream with gains of up to 2.23 dB for some segments (with an average gain of 2.03 dB) in the case of 2 Mbps network bandwidth, and up to 2.26 dB (with an average gain of 1.76 dB) in the case of 4 Mbps network bandwidth. We note that even though the reference views representations were encoded for constant quality using VBR, the combination of representations that results in the optimal virtual view quality differs from one segment to another, as can be seen in the figures.

In addition to calculating the quality gains in terms of PSNR, we also evaluated the quality gains using the structural similarity (SSIM) index quality metric for CBR and VBR encoded videos. We note that our proposed method also outperformed [24] in terms of SSIM but omit the results due to space limitations.

Table 2: Bandwidth change patterns.

Kendo	Time (sec.)	0	3	10	20	25	30
	Mbps	1.5	2.5	3	2	1	2.5
Café	Time (sec.)	0	5	10	15	25	30
	Mbps	1.5	2.5	4.5	3	2.5	3.5

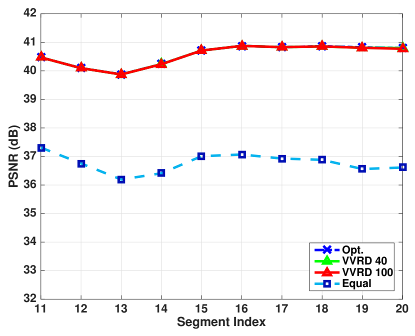
5.2.2 Variable Bandwidth

We now evaluate the streaming client's behaviour when the viewpoint is fixed while the bandwidth is varied. Because the video sequences used in the evaluation have different resolutions, and therefore different bandwidth requirements, we generate two different bandwidth change patterns, as shown in Table 2. These patterns are used by the KauNet VM to change the channel bandwidth between the client and server during the streaming session of the corresponding video sequence. We note that the results presented in this section are for VBR encoded videos. Similar results were obtained for the CBR encoding configuration. Figure 10 shows the results for segments 11 to 30 for the Kendo sequence. In Figure 10(a), the client's estimated channel bandwidth before downloading each segment is compared to the total bit rate for the operating point chosen by our rate adaptation method and that chosen based on [24]. As shown in the figure, because the approach presented in [24] is not flexible enough in terms of distributing the bit rates of the component segments due to the tight coupling between them at content generation time, it is unable to efficiently utilize all of the available bandwidth. Our virtual view quality-aware approach on the hand is able to take full potential of all the available bandwidth and improve the quality of the rendered virtual view. Using our rate adaptation algorithm, the client is able to achieve PSNR gains up to 2.13 dB, Figure 10(b), and SSIM gains up to 0.014, Figure 10(c), for view 2. Similar results were obtained for the Café sequence using the bandwidth change pattern in Table 2, where our proposed rate adaptation approach resulted in PSNR gains up to 1.09 dB and SSIM gains up to 0.022, as shown in Figure 11(b) and Figure 11(c), respectively, for view 2.5.

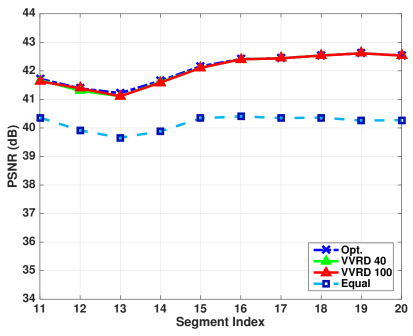
5.3 Subjective Evaluation

We have followed the recommendations given by ITU-R BT.500-13 [16] to perform subjective quality assessment experiments using the *double-stimulus continuous quality-scale* (DSCQS) method. In our subjective tests the quality of two impaired video sequences are considered in relation to each other. We evaluated a total of 12 test conditions (3 video content $\times$ 2 encoding configurations $\times$ 2 bandwidth capacities). Similar to the objective quality evaluation presented in Section 5.2, the virtual view at camera position 2 was used in the case of the Kendo and Balloons sequences, and the virtual view at camera position 2.5 was used for the Café sequence. For each test condition, the subjects where presented with two stimuli corresponding to two versions of the synthesized virtual view: one based on our proposed virtual view quality-aware rate adaptation algorithm, and one using the algorithm presented in [24]. The virtual views were generated from the reference segments chosen at 10 segment indices to obtain test stimuli with a duration of 10 seconds, following the BT.500-13 recommendations.

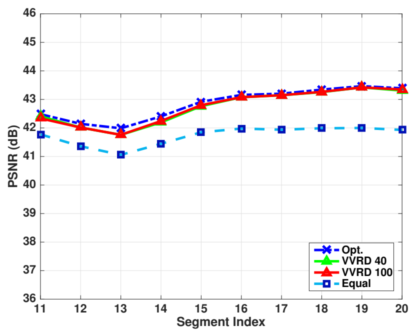
A total of 17 subjects participated in our experiments. The subjects were graduate computer science students (12 males and 5 females) at the university whose age ranged from 23 to 33 years old. All subjects were screened and given written instructions before the test session, and these instructions were also explained verbally to make sure they fully understood the experimental procedure. A



(a) 2 Mbps

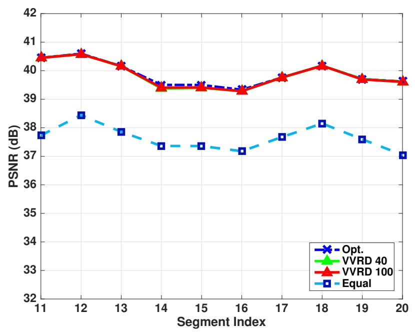


(b) 3 Mbps

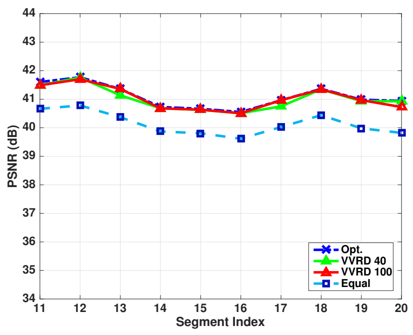


(c) 4 Mbps

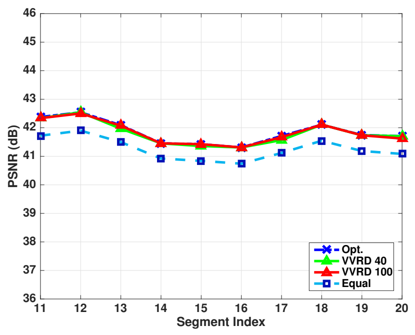
Figure 7: Average quality for the Balloons video sequence with CBR encoding and fixed network bandwidth.



(a) 2 Mbps

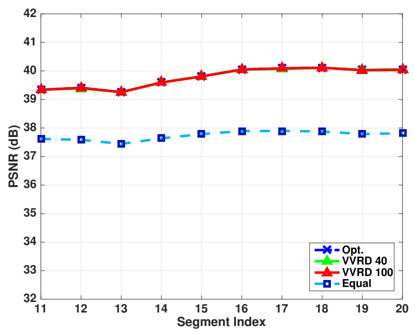


(b) 3 Mbps

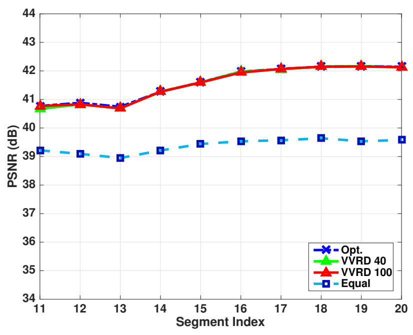


(c) 4 Mbps

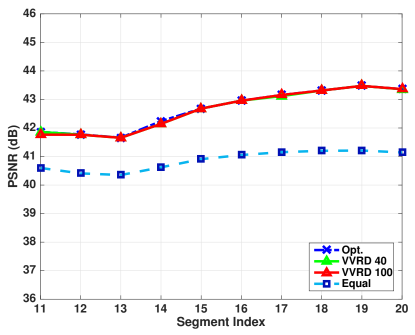
Figure 8: Average quality for the Café video sequence with CBR encoding and fixed network bandwidth.



(a) 2 Mbps



(b) 3 Mbps

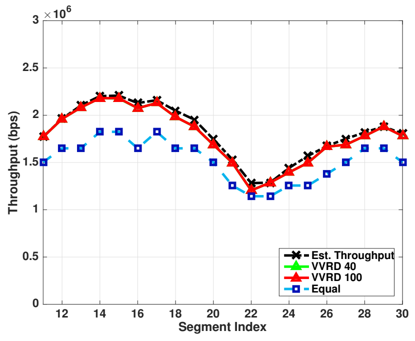


(c) 4 Mbps

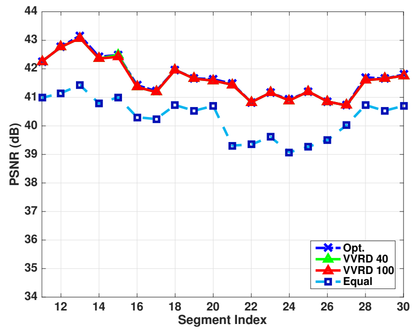
Figure 9: Average quality for the Balloons video sequence with VBR encoding and fixed network bandwidth.

comfortable seating arrangement was made for the subjects at a distance of three to four times the height of the display size. The test video sequences were shown on a 60" LG 4K Ultra HD 240Hz display (model 60UF8500). The 12 test conditions were shown to the subjects in random order. The order of the two stimuli was also randomized in each test session. For each test condition, one of the stimuli was shown for 10 seconds preceded by 3 seconds of mid-grey field indicating the coded name of the stimulus. Another mid-grey field with the coded name of the other stimuli is then shown for 3 seconds followed by the other 10 second stimulus. This presentation sequence is repeated a second time and followed by 10 seconds of mid-grey field in between different test conditions. The subjects were asked to rate the overall quality of both stimuli and mark their scores on a continuous grading scale. The marks were then mapped to integer values in the range $0 - 100$ and the *difference opinion score* (= score for stimulus based on our proposed approach − score for stimulus based on [24]) was calculated. Finally, we calculate the mean of the difference opinion score (DMOS) for each test condition. The results were then screened for outliers, where the voting scores of a test subject deviate above twice the standard deviation in more than half of the test videos. Only one test subject was determined to be an outlier and their voting scores were removed from the final results.

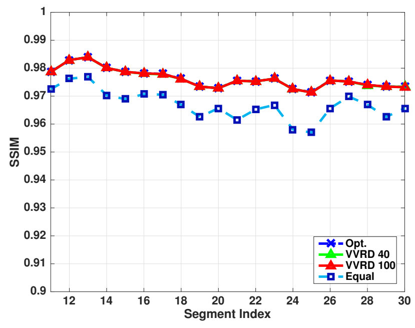
Figure 12 shows the DMOS values for the Kendo, Balloons, and Café video sequences at different available network bandwidth conditions. We evaluate two encoding configuration where the captured reference views are encoded using VBR and CBR. The results indicate that in almost all test conditions, the subjects rated



(a) Total segment bitrate

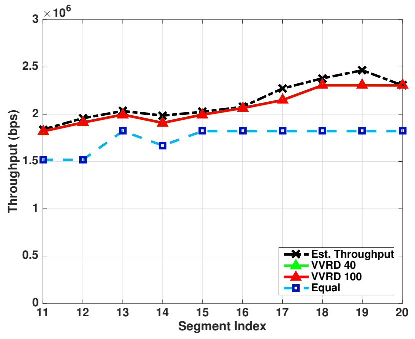


(b) PSNR

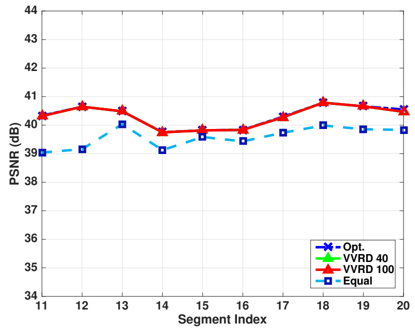


(c) SSIM

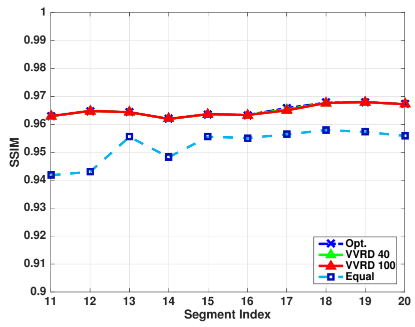
Figure 10: Results for the Kendo video sequence with variable network bandwidth.



(a) Total segment bitrate



(b) PSNR



(c) SSIM

Figure 11: Results for the Café video sequence with variable network bandwidth.

the virtual views synthesized from reference views representations chosen by our proposed rate allocation approach higher than those generated from reference views representations based on an equal bitrate allocation. This is indicated by positive DMOS values in the figure. Moreover, the quality improvement due to the proposed approach was higher in the case of CBR encoded reference views, which is the widely used encoding configuration for DASH content by most content providers. For example, for the Balloons video sequence, the subjects have rated the results of our rate adaptation method higher than those based on the method presented in [24] by 25 points on average and reported seeing a much clearer image.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a complete architecture for HTTP adaptive streaming of free-viewpoint videos. We proposed a novel two-step rate adaptation method that takes into consideration the user's interaction with the scene as well as the special characteristics of multi-view-plus-depth videos and the quality of rendered virtual views. Our rate adaptation method schedules the reference views that will be requested from the server based on the estimated viewpoint position of the user. The representations of the scheduled views are then determined based on virtual view quality models which are generated offline by the content provider from a small number of operating points. We introduced a number of extensions to the MPD file structure to include metadata necessary for supporting rate adaptation for FVV, such as camera parameters and virtual view quality models. A complete design of an FVV streaming client based on the proposed rate adaptation method was presented and a real client using implementing this design was developed using the MPEG-DASH standard and open source libraries and frameworks. Experimental results indicate that our proposed virtual view quality-aware rate adaptation method results in significant quality gains over other rate adaptation approaches (up to 4 dB for CBR streams and up to 2.26 dB for VBR streams), especially at low bandwidth conditions.

For future work, we plan to conduct more studies on user view navigation patterns and further enhance reference view scheduling. We also plan to take advantage of the recently finalized HTTP/2 protocol [6] by having all segment requests for a particular multi-component segment multiplexed over a single full-duplex persistent connection. Finally, we will consider hybrid rate adaptation approaches which incorporate both the estimated throughput as well as the client's segment buffer levels.

Acknowledgements

This work is partially supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

7. REFERENCES

[1] GPAC multimedia framework. http://gpac.wp.mines-telecom.fr/.
[2] libdash C++ library. https://github.com/bitmovin/libdash.
[3] MATLAB and Curve Fitting Toolbox Release 2015b. http://www.mathworks.com/.
[4] Nagoya University FTV test sequences. http://www.fujii.nuee.nagoya-u.ac.jp/multiview-data/mpeg/mpeg_ftv.html.
[5] OpenGL 2D/3D graphics API. https://www.khronos.org/opengl/.

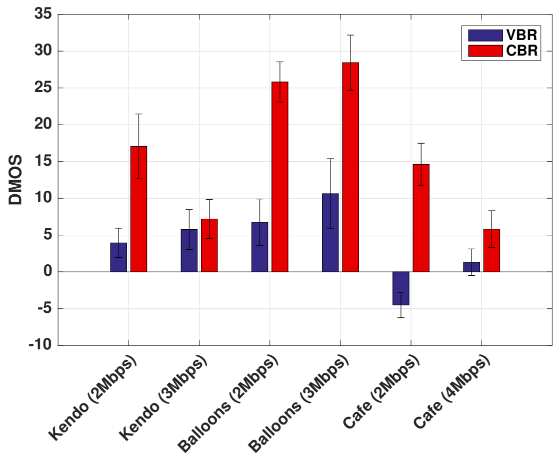


Figure 12: Difference mean opinion score (DMOS) between proposed virtual view quality-aware rate allocation and [24] for VBR and CBR encoded MVD videos at different available network bandwidth values (given between parentheses). Positive DMOS indicates that our approach is preferred over [24].

[6] M. Belshe, R. Peon, and M. Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540, RFC Editor, May 2015. http://www.rfc-editor.org/rfc/rfc7540.txt.

[7] Y. Chen, Y.-K. Wang, K. Ugur, M. Hannuksela, J. Lainema, and M. Gabbouj. The emerging MVC standard for 3D video services. *EURASIP Journal on Advances in Signal Processing*, 2009(1):786015, 2009.

[8] T.-Y. Chung, J.-Y. Sim, and C.-S. Kim. Bit allocation algorithm with novel view synthesis distortion model for multiview video plus depth coding. *IEEE Transactions on Image Processing*, 23(8):3254–3267, August 2014.

[9] C. Fehn. Depth-image-based rendering (DIBR), compression, and transmission for a new approach on 3D-TV. In *Proc. of SPIE Stereoscopic Displays and Virtual Reality Systems XI*, pages 93–104, 2004.

[10] Z. Gao, S. Chen, and K. Nahrstedt. OmniViewer: Enabling multi-modal 3D DASH. In *Proc. of the ACM International Conference on Multimedia*, pages 801–802, October 2015.

[11] J. Garcia, E. Conchon, T. Pérennou, and A. Brunstrom. KauNet: Improving reproducibility for wireless and mobile research. In *Proc. of the International Workshop on System Evaluation for Mobile Platforms*, pages 21–26, 2007.

[12] A. Hamza and M. Hefeeda. A DASH-based free-viewpoint video streaming system. In *Proc. of the ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 55–60, March 2014.

[13] ISO/IEC. Description of Exploration Experiments in 3D Video Coding. Doc. N11095, ISO/IEC JTC1/SC29/WG11 (MPEG), January 2010.

[14] ISO/IEC. Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats. ISO 23009-1:2012, 2012.

[15] ISO/IEC. Metadata Representation Carrying Quality Information Signalling for DASH. Doc. M32198, ISO/IEC JTC1/SC29/WG11 (MPEG), January 2014.

[16] ITU-R. Methodology for subjective assessment of the quality of television pictures. Recommendation ITU-R BT.500-13, 2012.

[17] E. Kurutepe, M. Civanlar, and A. Tekalp. Client-driven selective streaming of multiview video for interactive 3DTV. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(11):1558–1565, 2007.

[18] Y. Liu, Q. Huang, S. Ma, D. Zhao, and W. Gao. Joint video/depth rate allocation for 3D video coding based on view synthesis distortion model. *Signal Processing: Image Communication*, 24(8):666–681, 2009.

[19] T. Maugey and P. Frossard. Interactive multiview video system with low complexity 2D look around at decoder. *IEEE Transactions on Multimedia*, 15(5):1070–1082, 2013.

[20] L. McMillan. *An Image-Based Approach to Three-Dimensional Computer Graphics*. PhD thesis, University of North Carolina at Chapel Hill, 1997.

[21] P. Ramanathan and B. Girod. Rate-distortion analysis for light field coding and streaming. *Signal Processing: Image Communication*, 21(6):462 – 475, 2006.

[22] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1-3):7–42, 2002.

[23] S. Singhal and M. Zyda. *Networked Virtual Environments: Design and Implementation*. Addison-Wesley Professional, 1st edition, 1999.

[24] T. Su, A. Sobhani, A. Yassine, S. Shirmohammadi, and A. Javadtalab. A DASH-based HEVC multi-view video streaming system. *Journal of Real-Time Image Processing*, pages 1–14, 2015.

[25] G. Sullivan, J. Ohm, W.-J. Han, and T. Wiegand. Overview of the high efficiency video coding (HEVC) standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, 2012.

[26] V. Velisavljević, G. Cheung, and J. Chakareski. Bit allocation for multiview image compression using cubic synthesized view distortion model. In *Proc. of the IEEE International Conference on Multimedia and Expo*, pages 1–6, July 2011.

[27] J. Xiao, M. M. Hannuksela, T. Tillo, and M. Gabbouj. A paradigm for dynamic adaptive streaming over HTTP for multi-view video. In Y.-S. Ho, J. Sang, Y. M. Ro, J. Kim, and F. Wu, editors, *Advances in Multimedia Information Processing*, volume 9315 of *Lecture Notes in Computer Science*, pages 410–418. Springer International Publishing, 2015.

[28] H. Yuan, Y. Chang, J. Huo, F. Yang, and Z. Lu. Model-based joint bit allocation between texture videos and depth maps for 3-D video coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(4):485–497, April 2011.

[29] H. Yuan, J. Liu, H. Xu, Z. Li, and W. Liu. Coding distortion elimination of virtual view synthesis for 3D video system: Theoretical analyses and implementation. *IEEE Transactions on Broadcasting*, 58(4):558–568, December 2012.