UNIVERSITY OF CALIFORNIA SAN DIEGO

Enabling Wireless VR using Predictive Intelligence and Edge-Computing

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering (Computer Engineering)

by

Xueshi Hou

Committee in charge:

Professor Sujit Dey, Chair
Professor Dinesh Bharadia
Professor Truong Q. Nguyen
Professor Jurgen P. Schulze
Professor Geoffrey M. Voelker

2020

The dissertation of Xueshi Hou is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

Chair

University of California San Diego

2020

DEDICATION

To my family.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Foremost, I would like to thank my advisor Professor Sujit Dey for his invaluable guidance and support throughout my Ph.D. study. I have learned a lot from his enormous amount of knowledge, passion and responsibility in the research process.

I would like to thank every member of the MESDAT lab at UCSD for making my journey with joy and fun. I would also like to thank former senior labmate Dr. Yao Lu for his assistance through enthusiastic discussions.

I would like to thank my industrial collaborators (Jianzhong Zhang and Madhukar Budagavi) for their assistance and feedback in my research projects.

My sincere thanks go to my thesis committee members Professor Dinesh Bharadia, Professor Truong Nguyen, Professor Jurgen Schulze and Professor Geoffrey Voelker for their time, encouragement and insightful comments.

Last, but not least, I would like to express my deepest gratitude to my family and friends. Their support and love always inspire and encourage me in this journey.

The material in this thesis is based on the following publications.

Chapter 1 contains the reprint of Xueshi Hou, Yao Lu and Sujit Dey, "Wireless VR/AR with Edge/Cloud Computing," *Proc. of IEEE International Conference on Computer Communications and Networks*, 2017. The dissertation author is the primary investigator and author of this paper.

Chapter 2 contains the reprints of Xueshi Hou, Yao Lu and Sujit Dey, "A Novel Hyper-cast Approach to Enable Cloud-based Virtual Classroom," *Proc. of IEEE International Symposium on Multimedia*, 2016; and Xueshi Hou, Yao Lu and Sujit Dey, "Novel Hybrid-Cast Approach to Reduce Bandwidth and Latency for Cloud-Based Virtual Reality," *ACM Transactions on Multimedia Computing, Communications, and Applications*, 2018. The dissertation author is the primary investigator and author of each of these papers.

Chapter 3 contains the reprints of Xueshi Hou, Sujit Dey, Jianzhong Zhang and Madhukar Budagavi, "Predictive View Generation to Enable Mobile 360-degree and VR Experiences," *Proc. of the 2018 Morning Workshop on Virtual Reality and Augmented Reality Network*, 2018; and Xueshi Hou, Sujit Dey, Jianzhong Zhang and Madhukar Budagavi, "Predictive Adaptive Streaming to Enable Mobile 360-degree and VR Experiences," *IEEE Transactions on Multimedia*, 2020. The dissertation author is the primary investigator and author of each of these papers.

Chapter 4 contains the reprints of Xueshi Hou, Jianzhong Zhang, Madhukar Budagavi and Sujit Dey, "Head and Body Motion Prediction to Enable Mobile VR Experiences with Low Latency," *Proc. of the 2019 IEEE Global Communications Conference*, 2019; and Xueshi Hou and Sujit Dey, "Motion Prediction and Pre-Rendering at the Edge to Enable Ultra-Low Latency Mobile 6DoF Experiences," *IEEE Open Journal of the Communications Society*, 2020. The dissertation author is the primary investigator and author of each of these papers.

VITA

2015	B. Eng., Electronic Engineering, Tsinghua University, Beijing, China
2018	M. S., Electrical Engineering (Computer Engineering), University of California San Diego
2019	C. Phil., Electrical Engineering (Computer Engineering), University of California San Diego
2020	Ph. D., Electrical Engineering (Computer Engineering), University of California San Diego

PUBLICATIONS

X. Hou and S. Dey, "Motion Prediction and Pre-Rendering at the Edge to Enable Ultra-Low Latency Mobile 6DoF Experiences," *IEEE Open Journal of the Communications Society*, vol. 1, pp. 1674-1690, Oct. 2020.

X. Hou, S. Dey, J. Zhang and M. Budagavi, "Predictive Adaptive Streaming to Enable Mobile 360-degree and VR Experiences," *IEEE Transactions on Multimedia*, April 2020.

X. Hou, J. Zhang, M. Budagavi and S. Dey, "Head and Body Motion Prediction to Enable Mobile VR Experiences with Low Latency," in *Proc. of the 2019 IEEE Global Communications Conference*, Dec. 2019, pp. 1-7.

X. Hou, S. Dey, J. Zhang and M. Budagavi, "Predictive View Generation to Enable Mobile 360-degree and VR Experiences," in *Proc. of the 2018 Morning Workshop on Virtual Reality and Augmented Reality Network*, Aug. 2018, pp. 20–26.

X. Hou, Y. Lu and S. Dey, "Novel Hybrid-Cast Approach to Reduce Bandwidth and Latency for Cloud-Based Virtual Reality," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 14, no. 3s, pp. 1-25, June 2018.

X. Hou, Y. Lu and S. Dey, "Wireless VR/AR with Edge/Cloud Computing," in *Proc. of IEEE International Conference on Computer Communications and Networks*, Aug. 2017, pp. 1-8.

X. Hou, Y. Lu and S. Dey, "A Novel Hyper-cast Approach to Enable Cloud-based Virtual Classroom," in *Proc. of IEEE International Symposium on Multimedia*, Dec. 2016, pp. 533–536.

X. Hou, Y. Li, M. Chen, D. Wu, D. Jin and S. Chen, "Vehicular Fog Computing: A Viewpoint of Vehicles As the Infrastructures," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 6, pp. 3860–3873, June 2016.

X. Hou, Y. Li, D. Jin, D.O. Wu and S. Chen, "Modeling the Impact of Mobility on the Connectivity of Vehicular Networks in Large-Scale Urban Environment," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 4, pp. 2753–2758, April 2016.

Y. Liu, **X. Hou**, J. Chen, C. Yang, G. Su and W. Dou, "Facial Expression Recognition and Generation using Sparse Autoencoder," in *International Conference on Smart Computing*, Nov. 2014, pp. 125–130.

ABSTRACT OF THE DISSERTATION

Enabling Wireless VR using Predictive Intelligence and Edge-Computing

by

Xueshi Hou

Doctor of Philosophy in Electrical Engineering (Computer Engineering)

University of California San Diego, 2020

Professor Sujit Dey, Chair

Triggered by several head-mounted display (HMD) devices that have come to the market in recent years, such as Oculus Rift, HTC Vive, and Samsung Gear VR, significant interest has developed in virtual reality (VR) systems, experiences and applications. However, the current HMD devices are either tethered with PC/console, or rendering locally on itself (quite clunky to wear), negatively affecting user experience.

This thesis presents innovative methodologies to enable a truly portable and mobile VR experience, with lightweight VR glasses wirelessly connecting with edge/cloud computing devices that perform the rendering. There are two main challenges for this edge/cloud-based solution: (i) ultra-high bandwidth needed to transmit encoded video, and (ii) ultra-low latency

needed to avoid user's dizziness feeling.

To address the challenging requirements of bandwidth and latency, this thesis presents three methodologies. Firstly, we have investigated and developed a novel hybrid-cast approach to save bandwidth in a multi-user streaming scenario. We identify and broadcast the common pixels shared by multiple users, while unicast the residual pixels for each user. We formulate the problem of minimizing the total bitrate needed to transmit the user views using hybrid-casting and present a common view extraction approach and a smart grouping algorithm to achieve our hybrid-cast approach. Secondly, we have explored 360-degree video and three Degrees of Freedom (3DoF) VR streaming scenario, and proposed a predictive adaptive streaming approach to reduce latency needed by pre-delivery. By streaming the predicted view with high predictive probability in relatively high quality according to bandwidth conditions and transmitted in advance, we aim to address both latency and constrained wireless bandwidth challenges. Thirdly, for six Degrees of Freedom (6DoF) VR content, we have developed predictive pre-rendering approach to reduce latency needed. By predicting both head and body motions, our approach can pre-render the predicted view in advance and thus save latency in 6DoF VR scenarios.

This thesis concludes with a summary of its contributions and open directions for future research.

Chapter 1

Introduction

1.1 Background

Virtual reality (VR) systems have triggered enormous interest over the last few years in various fields including entertainment, enterprise, education, manufacturing, transportation, etc. However, several key hurdles need to be overcome for businesses and consumers to get fully on board with VR technology [1]: cheaper price and compelling content, and, most importantly, a truly mobile VR experience. Of particular interest is how to develop mobile (wireless and lightweight) head-mounted displays (HMDs), and how to enable VR experience on the mobile HMDs using bandwidth-constrained mobile networks, while satisfying the ultra-low latency requirements.

Currently, there are several categories of HMDs [2]: PC VR, standalone VR, and mobile VR. Specifically, PC VR has high visual quality with rich graphics contents as well as high frame rate, but the HMD is usually tethered with PC [3, 4]; standalone VR HMD has a built-in processor and is mobile, but may have relative low-quality graphics and low refresh rate [5, 6]; mobile VR is with a smartphone inside, leading to a heavy HMD to wear [7, 8]. Therefore, current HMDs still cannot offer us a lightweight, mobile, and high-quality VR experience. To solve this problem, we

propose an edge computing based solution. By performing the rendering on an edge computing node and streaming videos to users, we can complete the heavy computational tasks on the edge computing node and thus enable mobile VR with lightweight VR glasses. The most challenging part of this solution is ultra-high bandwidth and ultra-low latency requirements, since streaming 360-degree video causes tremendous bandwidth consumption and good VR user experiences require ultra-low latency ($<$20ms) [9, 10].

Specifically, the total end-to-end latency of edge computing based VR system includes the following parts: time to transmit sensor data from HMD to edge computing node, time to render (and encode) on the edge node, time to transmit rendered video from the edge computing node to HMD, and time to (decode and) display the view on the HMD. The encoding and decoding are optional according to the specific application design. Once the user moves his/her head or body position, high-quality VR requires this end-to-end latency as less than 20ms [9, 10] to avoid motion sickness. For the edge computing based VR system, it is extremely challenging to meet this requirement.

1.1.1 Challenges

In this section, we look at two types of applications, entertainment/collaboration applications that are characterized by virtual spaces where multiple virtual users interact, train and collaborate, and gaming applications which are typically characterized by higher levels of activity and speed. We develop cloud/edge streaming based representative virtual space (virtual classroom) and VR gaming applications, and experimentally determine the challenges to enable such cloud/edge-based VR applications.

Enabling edge/cloud-based wireless virtual spaces can be really challenging. It has been shown earlier that cloud/edge-based video rendering and streaming (for applications like cloud-based mobile gaming) requires high cloud and network bandwidth as well as low response time, satisfying which can be challenging considering dynamic variability of available wireless channel

conditions and bandwidth [11, 12]. Furthermore, use of HMD for VR experiences makes the requirements much steeper. Compared to PC monitor, HMDs are closer to human eyes with a much broader view, so it needs significantly higher bandwidth. Also, HMDs responds to the head motion, thus it needs higher framerate and lower latency to reduce dizziness.

Table 1.1: Bitrate and latency requirements for a virtual classroom application, under different types of display devices and motion scenarios.

Display Device	*Head Motion*	*Framerate & QP*	*Bitrate (1080p)*	*Acceptable Total Latency*
PC Monitor	–	45fps, QP=20	5.8Mbps	100∼200ms
Oculus	–	45fps, QP=15	10.9Mbps	28ms
Oculus	√	75fps, QP=15	28.2Mbps	22ms

Table 1.2: Bitrate and latency requirements for a racing game application, under different types of display devices and motion scenarios.

Display Device	*Head Motion*	*Framerate & QP*	*Bitrate (1080p)*	*Acceptable Total Latency*
PC Monitor	–	45fps, QP=20	16.6Mbps	<100ms
Oculus	–	45fps, QP=15	33.9Mbps	28ms
Oculus	√	75fps, QP=15	39.7Mbps	22ms

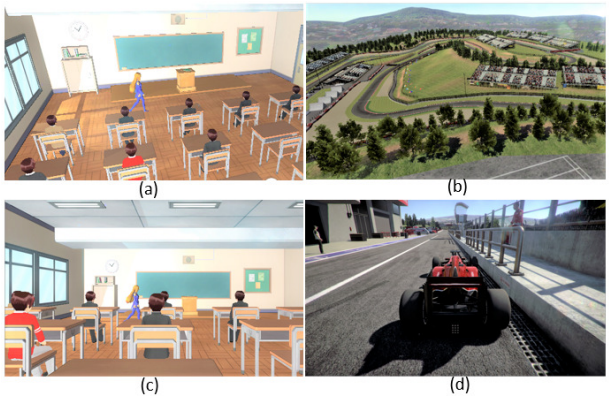


Figure 1.1: Two applications used in our experiments: (a)(c) show a virtual classroom application while (b)(d) demonstrate a racing game application.

To better understand the requirements and challenges of enabling virtual space experiences using remote rendering (at the cloud or edge), we conducted the following experiments with a virtual classroom application we developed using Unity [13], which is presented in Figure 1.1(a)(c). The virtual classroom is rendered remotely on a server, and the rendered video is

encoded using H.264/AVC encoder at 1080p resolution and GOP of 30. Subjects experience the virtual classroom on a PC monitor, as well as with Oculus Rift DK2 as VR HMD. Table 1.1 shows the video bitrate needed and acceptable round-trip response time for a good user experience of the virtual classroom application, depending on whether PC or Oculus is used for viewing, and whether the user (Oculus) has head motion. The results validate that when using HMD, very high data rate and very low latency are required for an acceptably high user experience. Note that the bitrates shown in Table 1.1 are for a single 1080p video stream (corresponding to a single user in the virtual space) – the bitrate needed will significantly increase if 4K or higher resolution is used, and the total requirements will significantly increase depending on the number of virtual users in the space.

Similarly, we also did the following experiments with a racing game application using Unity [13], which is demonstrated in Figure 1.1(b)(d). The differences between these two applications are mainly that virtual classroom can be more static than the racing game in general. The content change in virtual classroom without head motion generally lies in the movement of teacher and occasional movement of students, while for racing game the view will be continuously and significantly changed since the car is almost always moving. Table 1.2 shows the video bitrate needed and acceptable round-trip response time for a good user experience of the racing game application, with different types of display devices and motion scenarios (i.e., (i) whether PC or Oculus is used, and (ii) whether the user has head motion). The results for racing game also validate that when using HMD, high data rate and low latency are required for an acceptably high user experience. Comparing Table 1.1 and Table 1.2, we can observe that data rate needed for racing game is higher than virtual classroom since generally the former can have more content motion.

To address the dual challenge of requiring very high bitrate and very low latency for remote server based VR applications (i.e. virtual space and gaming applications), we propose novel solutions consisting of key innovations in streaming VR rendered videos and rendering

models. In the following, we describe more details of our proposed techniques and also present other possible solutions, which can satisfy the ultra-low latency requirement and address the bitrate/bandwidth challenges.

1.2 This Thesis

This thesis presents innovative optimizations and design methodologies to address bandwidth and latency challenges in edge/cloud-based VR. Figure 1.2 illustrates the scope and organization of this thesis.

To address the challenging requirements of bandwidth and latency, this thesis presents three methodologies. Firstly, we have investigated and developed a novel hybrid-cast approach to save bandwidth in a multi-user streaming scenario. We identify and broadcast the common pixels shared by multiple users, while unicast the residual pixels for each user. We formulate the problem of minimizing the total bitrate needed to transmit the user views using hybrid-casting and present a common view extraction approach and a smart grouping algorithm to achieve our hybrid-cast approach. Secondly, we have explored 360-degree video and three Degrees of Freedom (3DoF) VR streaming scenario, and proposed a predictive adaptive streaming approach to reduce latency needed by pre-delivery. By streaming the predicted view with high predictive probability in relatively high quality according to bandwidth conditions and transmitted in advance, we aim to address both latency and constrained wireless bandwidth challenges. Thirdly, for six Degrees of Freedom (6DoF) VR content, we have developed predictive pre-rendering approach to reduce latency needed. By predicting both head and body motions, our approach can pre-render the predicted view in advance and thus save latency in 6DoF VR scenarios.

The remainder of this thesis is organized as follows.

- Chapter 2 presents our work towards enabling cloud-based virtual space applications, for better computational scalability and easy access from any end device, including future lightweight wireless head-mounted displays (HMDs). In particular, we investigate virtual

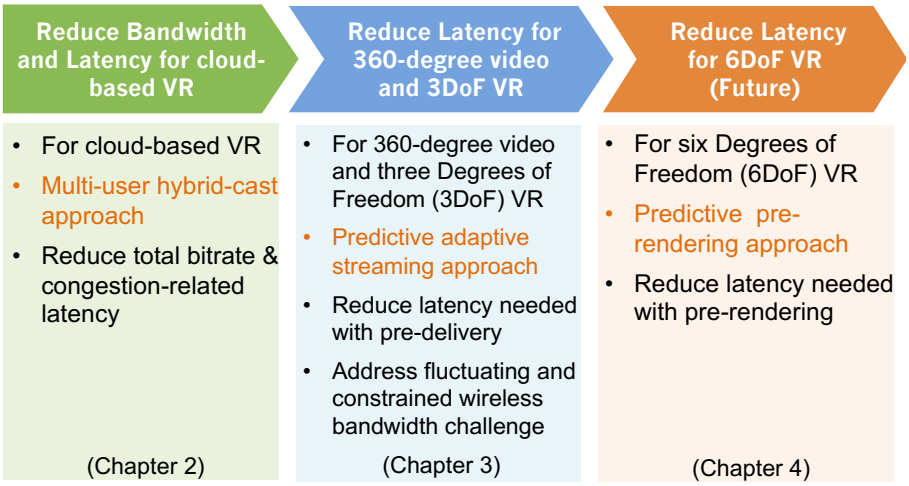


Figure 1.2: Scope and organization of this thesis.

space applications such as virtual classroom and virtual gallery, in which the scenes and activities are rendered in the cloud, with multiple views captured and streamed to each end device. A key challenge is the high bandwidth requirement to stream all the user views, leading to high operational cost and potential large delay in bandwidth-restricted wireless network. We propose a novel hybrid-cast approach to save bandwidth in a multi-user streaming scenario. We identify and broadcast the common pixels shared by multiple users, while unicast the residual pixels for each user. We formulate the problem of minimizing the total bitrate needed to transmit the user views using hybrid-casting and describe our approach. A common view extraction approach and a smart grouping algorithm are proposed and developed to achieve our hybrid-cast approach. Simulation results show that the hybrid-cast approach can significantly reduce total bitrate by up to 55% and avoid congestion-related latency, compared to traditional cloud-based approach of transmitting all the views as individual unicast streams, hence addressing the bandwidth challenges of cloud, with additional benefits in cost and delay.

- Chapter 3 presents the methodology targeted to the mobile 360-degree video streaming. As 360-degree videos and virtual reality (VR) applications become popular for consumer and

enterprise use cases, the desire to enable truly mobile experiences also increases. Delivering 360-degree videos and cloud/edge-based VR applications require ultra-high bandwidth and ultra-low latency, challenging to achieve with mobile networks. A common approach to reduce bandwidth is streaming only the field of view (FOV). However, extracting and transmitting the FOV in response to user head motion can add high latency, adversely affecting user experience. In this chapter, we propose a predictive adaptive streaming approach, where the predicted view with high predictive probability is adaptively encoded in relatively high quality according to bandwidth conditions and transmitted in advance, leading to a simultaneous reduction in bandwidth and latency. The predictive adaptive streaming method is based on a deep-learning-based viewpoint prediction model we develop, which uses past head motions to predict where a user will be looking in the 360-degree view. Using a very large dataset consisting of head motion traces from over 36,000 viewers for nineteen 360-degree/VR videos, we validate the ability of our predictive adaptive streaming method to offer high-quality view while simultaneously significantly reducing bandwidth.

- Chapter 4 presents the methodology that can be used for mobile VR application streaming. As virtual reality (VR) applications become popular, the desire to enable high-quality, lightweight, and mobile VR can potentially be achieved by performing the VR rendering and encoding computations at the edge and streaming the rendered video to the VR glasses. However, if the rendering has to be performed after the edge gets to know of the user's new head and body position, the ultra-low latency requirements of VR will not be met by the roundtrip delay. In this chapter, we introduce edge intelligence, wherein the edge can predict, pre-render and cache the VR video in advance, to be streamed to the user VR glasses as soon as needed. The edge-based predictive pre-rendering approach can address the challenging six Degrees of Freedom (6DoF) VR content. Compared to 360-degree videos and 3DoF (head motion only) VR, 6DoF VR supports both head and body motion, thus not only viewing direction but also viewing position can change. Hence, our

proposed VR edge intelligence comprises of predicting both the head and body motions of a user accurately using past head and body motion traces. In this chapter, we develop a multi-task long short-term memory (LSTM) model for body motion prediction and a multi-layer perceptron (MLP) model for head motion prediction. We implement the deep learning-based motion prediction models and validate their accuracy and effectiveness using a dataset of over 840,000 samples for head and body motion.

- Chapter 5 concludes the thesis and gives future directions in edge/cloud-based VR solutions.

1.3 Acknowledgements

Chapter 1 contains the reprint of Xueshi Hou, Yao Lu and Sujit Dey, "Wireless VR/AR with Edge/Cloud Computing," *Proc. of IEEE International Conference on Computer Communications and Networks*, 2017. The dissertation author is the primary investigator and author of this paper.

Chapter 2

Novel Hybrid-Cast Approach to Reduce Bandwidth and Latency for Cloud-Based Virtual Space

This chapter presents our work towards enabling cloud-based virtual space applications, for better computational scalability and easy access from any end device, including future lightweight wireless head-mounted displays (HMDs). In particular, we investigate virtual space applications such as virtual classroom and virtual gallery, in which the scenes and activities are rendered in the cloud, with multiple views captured and streamed to each end device. A key challenge is the high bandwidth requirement to stream all the user views, leading to high operational cost and potential large delay in bandwidth-restricted wireless network. We propose a novel hybrid-cast approach to save bandwidth in a multi-user streaming scenario. We identify and broadcast the common pixels shared by multiple users, while unicast the residual pixels for each user. We formulate the problem of minimizing the total bitrate needed to transmit the user views using hybrid-casting and describe our approach. A common view extraction approach and a smart grouping algorithm are proposed and developed to achieve our hybrid-cast approach. Simulation

results show that the hybrid-cast approach can significantly reduce total bitrate by up to 55% and avoid congestion-related latency, compared to traditional cloud-based approach of transmitting all the views as individual unicast streams, hence addressing the bandwidth challenges of cloud, with additional benefits in cost and delay.

2.1 Introduction

In recent years, Virtual Reality (VR) has become increasingly popular and triggered great interest worldwide. A series of new head-mounted displays (HMDs), with the advancement of display and system on chip (SoC) technology, are unlocking new applications in various fields including gaming, education, enterprise, entertainment, manufacturing, media and transportation. However, due to the high computational requirement of VR applications, current HMDs are either tethered to a PC (like Oculus Rift [14] and HTC Vive [4]), or attach to a smartphone (like Samsung Gear VR [15]), thus decreasing their mobility. In order to enable a truly portable and mobile VR experience, cloud-based approach is of great interest, where views are rendered and encoded in the cloud, and then transmitted to the end user as video streams over the wireless network. In this case, with mobile cloud computing techniques [16, 17], the HMDs only need to decode the video stream, with the possibility of a dramatically simplified and lightweight HMD design, not tethered to PCs or smartphones.

In this chapter, we explore the possibility of enabling an emerging category of virtual reality applications, virtual spaces, for better computational scalability and easy access from any end device. Specifically, we consider two virtual space applications – i.e., virtual classroom and virtual gallery, where a teacher/guide and students/visitors from different geographic locations can participate and communicate in the same classroom/gallery session, rendered as avatars in the same virtual space. A key challenge in enabling such cloud-based virtual space applications is the high bandwidth requirement to stream multiple views to each of the user, especially in the

case of wireless VR applications, where all the users may share a limited bandwidth. Hence, we seek to minimize the total bitrate needed to transmit video streams of all users' view without compromising video quality. Note that though we develop and demonstrate our approach on two specific virtual space applications, virtual classroom and gallery, our algorithms and approach can be applied to other virtual space applications, such as virtual conference, stadium, campus, etc.

As opposed to general cloud-based streaming applications, where each user's view is unique, in a virtual space application, users are usually close-by with views overlapping. For example, in Figure 2.1, view A and view B are captured from two virtual cameras (i.e. visitors' views) in a virtual museum application, with large portion of shared pixels. By taking advantage of this observation, in this chapter, we propose a novel hybrid-cast approach in which not every pixel of each user's view rendered on the cloud needs to be encoded and streamed from the cloud to each user separately. Instead, we broadcast only one copy of common pixels (common view), and unicast the rest of the pixels (residual pixels or residual views) to each individual user.

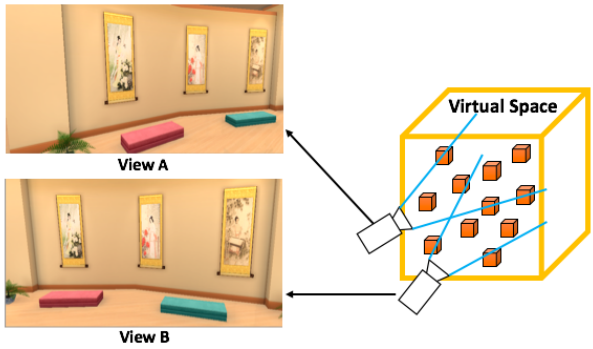


Figure 2.1: Two views in the same virtual space application (museum).

Another challenge lies in the selection of common view among multiple users. For better bandwidth reduction, users should be grouped with large portion of shared pixels so that common view can be maximized and residual view can be minimized. However, an optimal algorithm for grouping is computational expensive, and impractical for real-time use, due to the frame-to-frame view change of each user. In this work, we propose a fast, and effective smart grouping algorithm with novel metrics to evaluate the quality of grouping. The main contributions of our work can be

summarized as follows.

- We propose a novel and efficient hybrid-cast approach to reduce the total bandwidth to stream all the user views in cloud-based virtual space applications, by broadcasting a single common view and unicasting only the residual views of individual users, without loss of video quality. Thus, our approach is able to enhance the user experience (i.e. higher video quality and less latency) by alleviating network congestion. We show that we can reduce up to 55% total bitrate needed for a virtual classroom and a virtual gallery application, with latency improvement in a bandwidth-limited wireless network.

- We develop a common view extraction approach to calculate common view and residual view between any two user views. To the best of our knowledge, we are the first to perform common view extraction by exploring the pipeline of 3D virtual space rendering.

- We propose an automated, and smart grouping algorithm to assign multiple users to different groups in order to minimize the total bitrate needed to transmit all user views.

- We propose a new metric to enable fast and accurate calculation of common view between two views. Compared to the optimal algorithm, we achieve similar solution quality with hundreds of times speedup.

Note that we have published a conference paper [18] about this work, where we report on the hybrid-cast approach and some preliminary results. In this chapter, we extend our approach by proposing a smart real-time grouping algorithm, improving our proposed metrics and conducting experiments on various virtual space applications.

The remainder of the chapter is organized as follows. In Section 2.2, we review related work. In Section 2.3, we introduce virtual space applications and describe the architecture of our proposed hybrid-cast approach. In Section 2.4, we firstly describe the methodology for common view extraction, problem statement, optimization metrics, and grouping algorithm. Section 2.5 presents our experimental set up and analyze the results. Section 2.6 concludes our work.

2.2 Related Work

For cloud-based 3D virtual space applications, the main difference compared to a standard local rendering pipeline [19] is the use and transmission of the entire screen as a video stream [20]. Therefore, reducing total bitrate of the video streams is the key to avoid congestion-related latency and improve user experience, as stated in Section 2.1. In this section, we review the previous works addressing the bandwidth challenge, in (i) codec development, (ii) joint optimization for computer-generated view streaming, and (iii) other techniques.

Upgrading video codec can be the most direct way to reduce bitrate consumption. Video coding standards have evolved primarily through development by ITU-T and ISO/IEC standardization, from H.261 [21] and H.263 [22] by ITU-T, MPEG-1 [23] and MPEG-4 Visual [24] by ISO/IEC, to jointly produced H.262/MPEG-2 [25], H.264/MPEG-4 Advanced Video Coding (AVC) [26] and H.265/MPEG-H High Efficiency Video Coding (HEVC) [27] standards. In recent years, VP9 [28] and AV1 [29] also appear as strong functional video coding formats due to their open and royalty free features. However, even if the bit rate can be reduced by using new video encoding standards, with the increasing resolution of the devices, the bit rate requirements of the video streams are expected to keep increasing, and particularly in the case of multi-stream VR applications.

In terms of streaming computer-generated views, joint optimization for both video codec and graphics engine has proved to be more effective [11, 30, 31]. Wang *et al.* [11] propose a rendering adaptation technique to dynamically adjust the richness and complexity of graphic rendering depending on the network and cloud computing constraints. Liu *et al.* [30] derive a content-aware adaptive rendering algorithm to adjust rendering factors depending on current network conditions, so as to obtain an ”optimal tradeoff” between rendering and encoding quality. Lu *et al.* [31] propose a joint asymmetric graphics rendering and video encoding approach for automatic selection of texture details or view distance settings for the left view and right view.

However, all these approaches compromise video quality to achieve smaller bitrate and therefore inappropriate to be used for HMDs due to proximity to the eyes.

Alternatively, Cai *et al.* [32] propose a peer-to-peer cooperative video sharing solution in a multi-player scenario to substantially reduce the total bitrate from cloud server to game clients. However, their approach is only applicable to scenarios such as 2D third-person games, where game players may share the same bird-view. Hence, their method cannot be applied to 3D virtual spaces.

In summary, codec-only optimization is limited by not exploring any additional benefits in computer-generated view streaming. Joint optimization usually tradeoff quality for bandwidth, which is inappropriate to be applied for HMDs since users are more sensible to the video quality in VR virtual space applications. Other works explore peer-to-peer streaming, with the limitation of fixed views of all users. Our work is distinguished from all previous works in that (i) we propose a hybrid-casting approach to explore the benefits of computer-generated views in virtual space VR applications to reduce the total bitrate needed, while not sacrificing video quality; (ii) we develop a common view extraction algorithm by exploring the 3D rendering pipeline; (iii) we propose an automated and smart grouping algorithm, with fast and accurate real-time evaluation with empirically validated experimental results.

2.3 Overview

In this section, we first present virtual classroom and gallery applications. Next, we describe the architecture of our proposed hybrid-cast approach.

2.3.1 Virtual Space Applications

We have developed a prototype implementation of the virtual classroom application using Oculus [14] and Unity [13]. Compared to the Massive Open Online Course (MOOC) [33], where

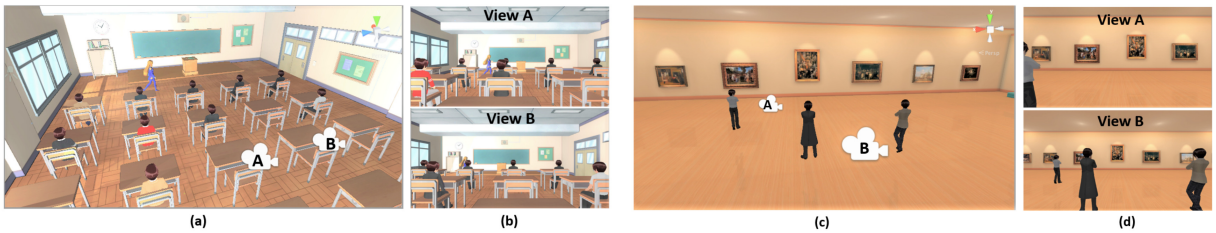


Figure 2.2: (a) Two users' positions in a virtual classroom; (b) Corresponding views of two users; (c) Two users' positions in a virtual gallery; (d) Corresponding views of two users.

students can only watch video without interaction with the teacher and each other, the virtual classroom offers students a more immersive and interactive experience. Figure 2.2(a) illustrates the virtual classroom, where each student has a unique view of the classroom. Figure 2.2(b) shows two views from two students as an example. In our implementation, we place cameras to represent the students' views and more views can be easily obtained by placing more cameras. In such a virtual classroom application, neighboring students may share a common view (the set of pixels corresponding to same coordinates in the object world between two views), as shown in Figure 2.2(b). As is stated in Section 2.1, with our proposed approach, we can identify and broadcast the common pixels shared by multiple users, while unicast the residual pixels for each user. In this way, our proposed approach can address the bandwidth challenges of cloud, with benefits in cost and delay.

Similarly, we have developed a virtual gallery application using Unity and conducted the related experiments on it. Figure 2.2(c) illustrates the virtual gallery, where each visitor has a unique view of the gallery. Figure 2.2(d) shows two views from two visitors as an example. As is shown in Figure 2.2(d), two visitors also share a common view.

2.3.2 Hybrid-Cast Approach

Figure 2.3 illustrates the architecture of the hybrid-cast approach. On the cloud, multiple views are rendered in response to commands from multiple users, and then captured, encoded and transmitted as video streams. To enable our hybrid-cast approach, we cluster the users into

different groups based on their views. Within each group, only one user's view is assigned to be the *primary view*, and all the other views are *secondary views*. For transmission, we broadcast the entire *primary view* with full pixel information to all group members, but only unicast the residual pixels (non-existing in the *primary view*) for each *secondary view*. Figure 2.4 gives an example. Figure 2.4(a) shows a *primary view* A and Figure 2.4(b) presents a *secondary view* B. *Secondary view* B consists of two parts, *common view* and *residual view*. The *common view* is part of the view that shares the common pixels between A and B. Figure 2.4(d) shows the *common view* of B shared with A and Figure 2.4(c) presents the corresponding shared view from view A's perspective. For *secondary view* B, *residual view* is defined as the pixels non-existing in *primary view* A. Figure 2.4(e) shows the *residual view*, which can be used to recover *secondary view* B (Figure 2.4(f)) by combining with *common view* (Figure 2.4(d)). We explain the common view extraction and our grouping algorithm in Section 2.4.1, 2.4.3 and 2.4.4. For instance, in a virtual classroom like Figure 2.2(a), user A is a student in a back row within the class, sharing a subset of the view from the students in front of him. With our proposed grouping algorithm, user A will be clustered into a group, where another student within this group is assigned as the user with *primary view* (since only one user is assigned to have *primary view* in each group). The *primary view* may include the blackboard, teacher, walls, etc., while the *residual view* of user A may consist of the back of other students in front of him or her, etc. After doing synthesis of *primary view* and *residual view* on the client side, user A can obtain his own *secondary view*.

The bandwidth savings achieved by our hybrid-cast approach may depend on the location of the users. Figure 2.3 shows the data flow from the cloud server (*cloud*), through core network and gateways (*backhaul*), to base stations and end users (*cellular*), consuming different types of bandwidth – *cloud*, *backhaul* and *cellular bandwidth* – at different stages of the network. The bandwidth savings achieved by our approach (as reported in Section 2.5.3) are fully translated to the cloud bandwidth saving achieved, irrespective of the location of users, since transmitting a single *primary view* will suffice from the cloud servers. However, the backhaul and cellular

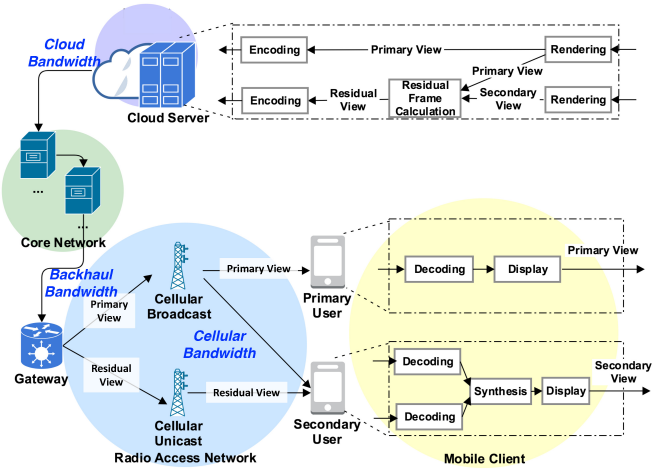


Figure 2.3: Hybrid-Cast approach.

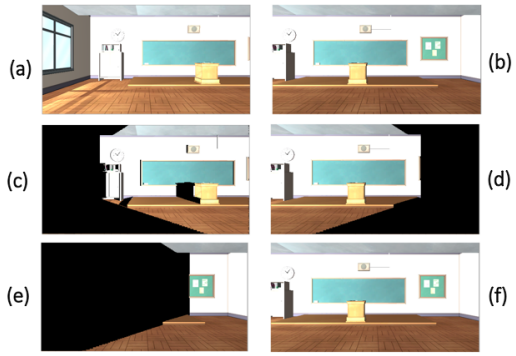


Figure 2.4: Illustration of common view extraction: (a) *primary view* A; (b) *secondary view* B; (c) common pixels from view A; (d) common pixels from view B; (e) *residual view* in B; (f) generated view B from *common view* and *residual view*.

bandwidth savings will depend on the location of users. For users associated with the same base station, clearly the bandwidth savings translate to both the cellular and backhaul, besides cloud bandwidth savings, as a single *primary view* can be broadcast to all base stations through the backhaul, and broadcast to all primary users through the access network (*cellular*). For users associated with different base stations which share the same gateway, although cellular bandwidth cannot be saved through cellular broadcast, backhaul bandwidth saving can be achieved since a single *primary view* can be transmitted (multi-cast) through the backhaul to the base stations for these users. However, for users associated with different base stations which do not share

Table 2.1: Three types of bandwidth savings.

Bandwidth Savings	***Feasibility***
Cloud	Always
Backhaul	For users associated with base stations connected by same gateway
Cellular	For users associated with same base station

Table 2.2: Several cases for applications.

Cases	***Description***	***Bandwidth Savings***
1	*Students in campus join a virtual gallery application* (associated with same base station)	Cloud, backhaul, cellular
2	*Users in the same urban region attend a virtual classroom (associated with base stations connected by same gateway)*	Cloud and backhaul
3	*Users in distant places participate in a virtual space* (e.g. belonging to different networks)	Only cloud

same gateway, while the savings do not translate to the backhaul and access bandwidth, cloud bandwidth savings can still be achieved as explained earlier. We summarize the three types of bandwidth savings achieved by our approach under different user locations in Table 2.1.

We provide an estimation of bandwidth savings for several cases in Table 2.2. For case 1, when students in campus join a virtual gallery application, we can obtain *cloud, backhaul* and *cellular* bandwidth savings. In case 2, for users in the same urban region attending a virtual classroom, *cloud* and *backhaul* bandwidth savings can be gained with our proposed approach. Since the gateway connected to core network can support from 100 base stations for medium size urban network (5x5 km coverage) to over 1000 base stations (15x15 km coverage) [34, 35], the virtual application users within this coverage area will share the same gateway, and hence will result in *backhaul* bandwidth savings, though not *cellular* bandwidth savings as the latter will depend on the distribution of the users across the base stations. For case 3 where users may belong to different networks, only *cloud* bandwidth saving can be achieved.

Note that to maximize gains in bandwidth saving, our proposed method requires multicast protocols to be employed on the entire end-to-end paths (from the cloud center, through the core network and all the way to users). Since multicast protocols are not widely employed in

today's networks, our approach is developed based on the assumption that multicast protocols will be well deployed and utilized in the future networks. Next, we briefly discuss the existing data transmission mechanisms that can be used for access links (cellular) and backhaul links. As for wireless access link, broadcast and unicast can be realized by using existing LTE Evolved Multimedia Broadcast Multicast Services (eMBMS) [36] standards, and broadcast/multicast point to multipoint (PTM) [37] being developed for future 5G access networks. In terms of transmissions in cloud and backhaul links, advanced IP multicast [38] protocols can be used, including Pragmatic General Multicast (PGM) [39], Multicast File Transfer Protocol (MFTP) [40], Real-time Transport Protocol (RTP) [41], and Resource Reservation Protocol (RSVP) [42]. Data routing, forwarding and associated transmission protocols should be further studied in future work.

Our approach achieves best performance in reducing bandwidth when users are associated with the same base station, and only *cloud* bandwidth saving (no *backhaul* and *cellular*) can be guaranteed when users are distributed in distant places (since users do not share the same cellular gateway). Moreover, the limitations of our approach mainly come from (a) the difficulty of large-scale deploying multicast protocols in current networks; (b) our approach only supports an indoor 3D environment (where views aimed toward the same wall) and does not support a fully tridimensional environment such as outdoor environments. We propose the following to address these limitations. For limitation (a), apart from expecting multicast protocols widely deployed in the future, we can first apply our proposed approach in scenarios where users are associated with same base station or base stations connected by the same gateway, and take the conventional method (i.e. unicast the view directly) for transmitting views for users associated with other distant base stations. For example, the community in a campus or company is such a good scenario. In terms of limitation (b), in our future work, we will further develop corresponding metrics to apply our approach in fully tridimensional environments such as outdoor environments.

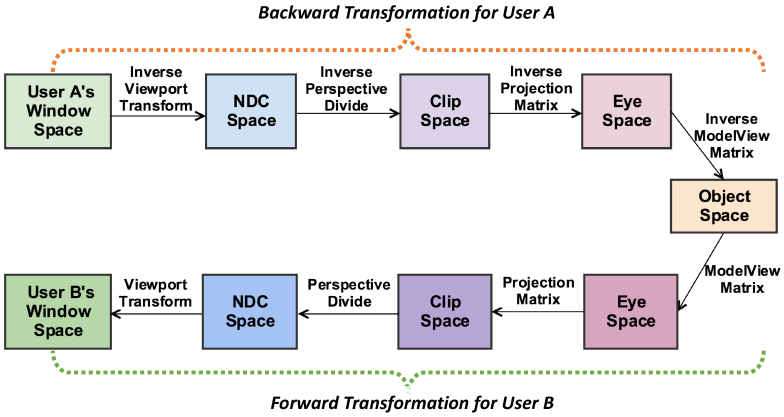


Figure 2.5: Architecture of common view extraction.

2.4 Our Approach

In this section, we first describe the methodology for common view extraction and give the problem statement. Next, we describe our optimization metrics and provide empirical validation. We then present our grouping algorithm.

2.4.1 Common View Extraction

Figure 2.5 illustrates the common view extraction flow between two users. In the flow, we assume that user A is assigned the *primary view* and user B is assigned the *secondary view*. To extract the *common view*, we perform a series of transformations from window space of user A to the object space, and further to the window space of user B. A *common view* is the set of pixels within the window space of A that falls into the window space of B after the above transformation. To better illustrate our approach, we define two series of transformations: (i) *forward transformation*, and (ii) *backward transformation*. For a pixel with coordinate $(x,y,z)_{object}$ in the object space, the *forward transformation* calculates the corresponding coordinate $(x,y,z)_{window}$ in the window space; and vice versa for the *backward transformation*.

Specifically, the OpenGL [43] pipeline performs the *forward transformation* from object space to window space, through four steps [44, 45]. The first step is to transform from the object

space to the eye space using the ModelView matrix $M_{ModelView}$; then we transform from the eye space to the clip space use projection matrix $M_{Projection}$; next we transform from the clip space to the Normalized Device Coordinate (NDC) [46] space using perspective dividing M_{Divide}; last, we transform from the NDC space to the window space by performing viewport transformation $M_{Viewport}$. (The details of OpenGL pipeline are omitted due to space constraints.) Overall, for a pixel with coordinate (x, y, z) in the object space, the procedure of transformation to the window space can be described as the following:

$$(x, y, z)_{window} = M_{Viewport} \cdot M_{Divide} \cdot M_{Projection} \cdot M_{ModelView} \cdot (x, y, z)_{object}$$

To check the common pixels between two window spaces, as described in Figure 2.5, we propose to use the inverse transformation matrices for *backward transformation* (i.e. calculating corresponding coordinate from the window space to the object space), as described in the following equation:

$$(x, y, z)_{object} = M^{-1}_{ModelView} \cdot M^{-1}_{Projection} \cdot M^{-1}_{Divide} \cdot M^{-1}_{Viewport} \cdot (x, y, z)_{window}$$

Therefore, by utilizing both the *backward* and the *forward transformations* in serial for only those pixels in window space A, we can tell if the pixel falls in window space B. Thus, we can extract the *common view*. Furthermore, we can also recover the *secondary view* by using the *primary view* and *residual view* based on the same theory. This procedure is named as synthesis in the Figure 2.3.

Table 2.3: Notations used in our formulation.

Notation	Meaning
V	view
V_{com}	common view
V_{res}	residual view
n	number of users
$P[\cdot]$	number of pixels
$R[\cdot]$	pixel ratio
P_{frame}	number of pixels in one frame
P_{total}	total number of pixels transmitted
R_{total}	total pixel ratio transmitted
p	primary view
q	secondary view
S_p	set of primary views
S_q	set of secondary views
B	binary indicator matrix
$b_{p,q}$	element of binary indicator matrix B
D	distance matrix
$D(p,q)$	element of distance matrix D

2.4.2 Problem Statement

We introduce the problem statement in this subsection. The basic notation used in our formulation is provided in Table 2.3.

Given: All views in the virtual classroom; Dimensions of the virtual classroom, including the width and length.

Find: An optimal strategy to minimize the total number of pixels transmitted P_{total} for all views.

$$\begin{aligned}
\min P_{total} &\Leftrightarrow \min\{\sum_{q\in S_q} P[V_{res}(q)] + \sum_{p\in S_p} P[V(p)]\} \\
&\Leftrightarrow \min\{\sum_{q\in S_q} R[V_{res}(q)] + \sum_{p\in S_p} R[V(p)]\} \\
&\Leftrightarrow \min\{\sum_{q\in S_q} R[V_{res}(q)] + |S_p|\} \\
&\Leftrightarrow \min\{\sum_{p\in S_p}\sum_{q\in S_q} b_{p,q}\cdot D(p,q) + |S_p|\} \qquad (2.1)
\end{aligned}$$

where

$$b_{p,q} = \begin{cases} 1, & \text{if } p,q \text{ are in the same group} \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

$$D(p,q) = R[V_{res}(q)] = 1 - R[V_{com}(q)] \tag{2.3}$$

Equation 2.1 demonstrates our objective to minimize the sum of total number of pixels transmitted P_{total} for all views. S_p and S_q represent the set of *primary views* and *secondary views*, respectively. $P[V_{res}(q)]$ and $P[V_{com}(q)]$ represent the number of residual and common pixels within *secondary view q*, respectively. Note that for each group, there are one user with *primary view* and rest of users with *secondary views*. For each *secondary view*, only one *primary view* is corresponding to calculate $P[V_{com}(q)]$ and $P[V_{res}(q)]$. $P[V(p)]$ denotes the number of pixels within *primary view p*.

Moreover, we define the residual pixel ratio of a *secondary view q* as

$$R[V_{res}(q)] = \frac{P[V_{res}(q)]}{P_{frame}}$$

and the common pixel ratio of a *secondary view q* as

$$R[V_{com}(q)] = \frac{P[V_{com}(q)]}{P_{frame}}$$

where P_{frame} is the number of pixels per frame. Since the number of pixels in *primary view p* is equal to P_{frame}, the pixel ratio $R[V(p)] = 1$. $|S_p|$ indicates the number of *primary views*.

In Equations 2.2 and 2.3, the elements of binary indicator matrix B and distance matrix D are defined as $b_{p,q}$ and $D(p,q)$ respectively. Specifically, the value of $D(p,q)$ equals to the

Table 2.4: Notations used to build metrics.

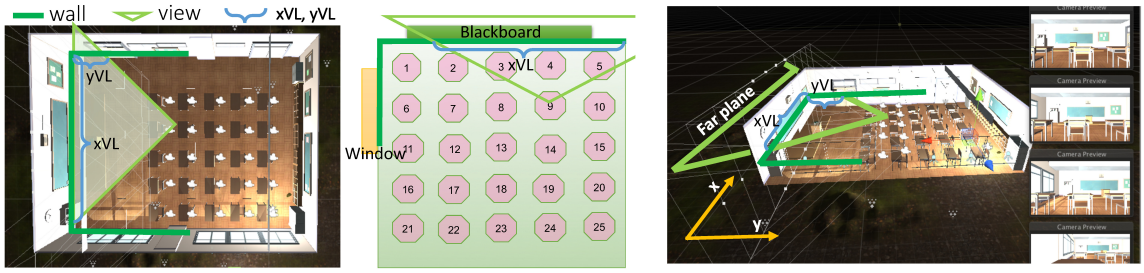
Notation	Meaning
xVL	length of front wall
yVL	length of side wall
$VLength$	metric to represent the range of a view in x-axis
x_i, y_i	location for user i in x-axis and y-axis
Δx	distance between a user and left side wall in x-axis
Δy	distance between a user and front wall in y-axis
cVL	metric to evaluate the common ratio between two views
$cnVL$	normalized form of cVL
$xfactor$	length of front and side wall within both views
$yfactor$	squared ratio of distance to the front wall for both views
RmW	room width
k, b	parameters in linear regression

residual pixel ratio of a *secondary view q* when the *primary view p* is selected. The larger the $D(p,q)$ is, the more the difference between view p and view q is.

2.4.3 Metrics and Evaluation

In our approach, in order to assign multiple users to different groups and minimize the sum of bitrate across multiple users, we want to develop a strategy to group the views which have a lot of common parts together. To make the grouping technique fast, we need to avoid the time-consuming process of conducting common view extraction between all pairs of different views. Instead, we develop and use an easy to calculate metric to represent how much is common between two views; we term this metric as *common normalized VLength (cnVL)*, which we define next. Though we consider the virtual classroom application to develop the metric below, the same definition will be applicable to other virtual spaces like virtual gallery.

The basic notation used in our proposed metrics is provided in Table 2.4. To be specific, we define *VLength* as the sum of *xVL* and *yVL*, as shown in Equation 2.4. In this definition, *xVL* and *yVL* denote the length of the front wall and side wall within the current view, respectively. Figure 2.6(a) illustrates from the top of the classroom an example of *xVL* and *yVL* with the view



(a) Looking from the top.

(b) Looking from the side.

Figure 2.6: The model of the virtual classroom and camera view when looking from the top and side. Lengths of *xVL* and *yVL* are shown. (a) presents the boundary of the classroom, seat positions as well as a demonstration of users' view. (b) exhibits views (on the right) as illustrations for student views of 4 different positions on the second row.

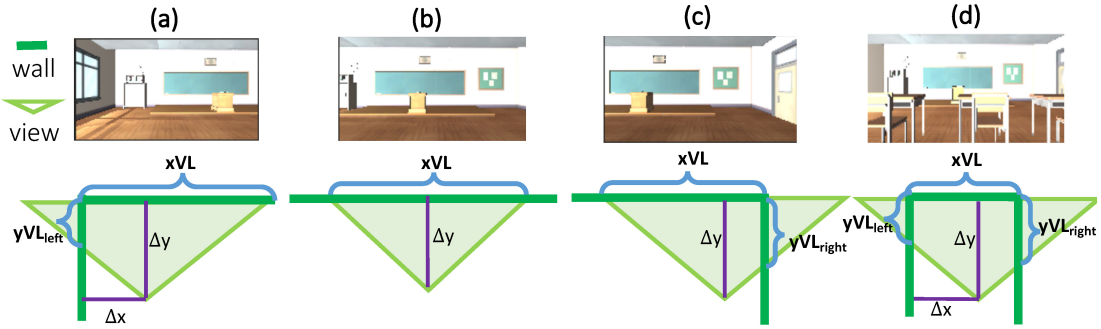


Figure 2.7: (a)-(d) show four types of relative positions between classroom boundary and view boundary. And *RmW* denotes the width of the classroom in x-axis.

of seat #9. Figure 2.6(b) illustrates from the side of the classroom another example of *xVL* and *yVL* of one student view, which is shown as the first view on the right. Figure 2.6(b) also presents four actual student views from four different positions in the second row of the classroom. We can also see the position of far plane, which exceeds the wall of virtual classroom. *VLength* also indicates the range of view in x-axis and is defined in Equation 2.4.

$$VLength = xVL + yVL \tag{2.4}$$

To calculate *VLength*, we summarize four difference types of relative positions between classroom boundary and view boundary, as shown in Table 2.5. Figure 2.7 also shows these

Table 2.5: Four different cases.

Case	a	b	c	d
Left wall visible	✓			✓
Front wall visible	✓	✓	✓	✓
Right wall visible			✓	✓

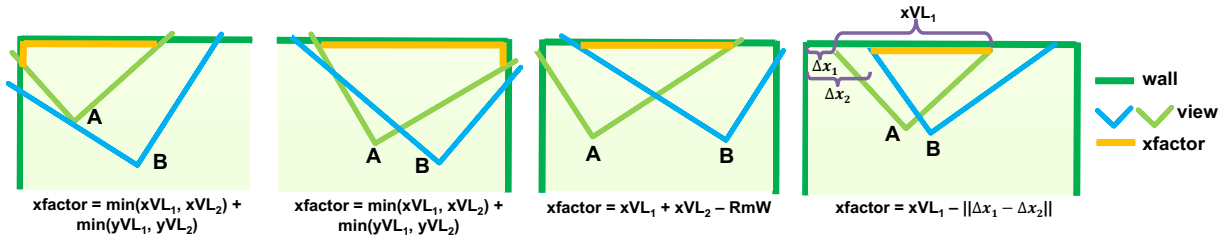


Figure 2.8: Four cases of two user views, where the yellow line denotes $xfactor$.

four types of relative positions as cases (a)–(d). Especially, we note that for case (d), VLength is obtained separately according to yVL from the left wall (yVL_{left}) and right wall (yVL_{right}), respectively, as shown in Equation 2.5.

$$VLength = xVL + yVL_{left} + yVL_{right} \tag{2.5}$$

We define the *Common VLength* (cVL) as a metric to evaluate the common ratio between two views p and q. Equation 2.6 describes cVL, with *xfactor* and *yfactor* defined in Equation 2.7 and 2.8 respectively. *xfactor* is defined as the length of the front and side wall within both views. Specifically, we assign two users with subscripts 1 and 2 to distinguish them. Subscript 1 represents the user on the left while subscript 2 denotes the user on the right. Condition 1 indicates that two views are of case (a) and (c), respectively; condition 2 indicates that both views are of case (b); and condition 3 indicates all other case combinations. *yfactor* is defined as the squared ratio of the distance to the front wall for both views.

$$cVL(p,q) = xfactor \cdot yfactor \tag{2.6}$$

where

$$xfactor = \begin{cases} xVL_1 + xVL_2 - RmW & \text{if condition 1;} \\ xVL_1 - |\Delta x_1 - \Delta x_2| & \text{if condition 2;} \\ \min(xVL_1, xVL_2) + & \\ \min((yVL_{left})_1, (yVL_{left})_2) + & \\ \min((yVL_{right})_1, (yVL_{right})_2) & \\ \qquad\qquad \text{otherwise (condition 3);} & \end{cases} \tag{2.7}$$

$$yfactor = \begin{cases} (\Delta y_1/\Delta y_2)^2, & \text{if } \Delta y_1 < \Delta y_2 \\ (\Delta y_2/\Delta y_1)^2, & \text{if } \Delta y_1 \geq \Delta y_2 \end{cases} \tag{2.8}$$

Figure 2.8 shows four cases of two user views (for users A and B), where the yellow line denotes *xfactor*. The proposed metric *cVL* is used to represent common pixel ratio between two views, and our approach can handle arbitrary viewing direction. For instance, when two users have small or even no common view, the corresponding *xfactor* and calculated *cVL* are small or even zero, representing that the common pixel ratio is small. As described in the next section, when a user A's viewing direction may be very different than user B's, user A may be clustered into a different group than user B due to their small common pixel ratio.

To evaluate our proposed metric, we compare *cVL* (Equation 2.6) to the common pixel ratio (defined in Section 2.4.2) for every pair of views in a virtual classroom with a seat pattern of 2x5 (10 views, 100 pairs of views). Each view has a resolution as 1080p. Figure 2.9(a) illustrates the correlation between common *VLength (cVL)* and the common pixel ratio values for the 100 pairs of views, with an overall correlation of 0.8828. To increase the correlation with common

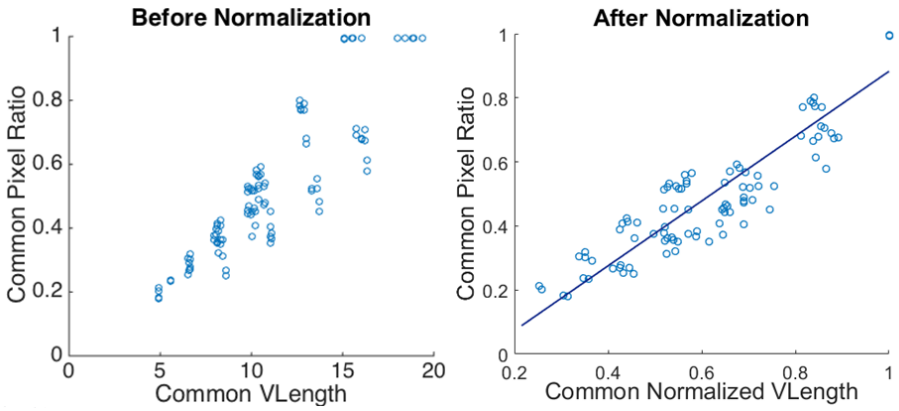


Figure 2.9: Validation of model parameters, showing the relationship between the common pixel ratio and *cVL* as well as *cnVL*.

pixel ratio, we define *common normalized VLength (cnVL)* as follows:

$$cnVL(p,q) = \frac{cVL(p,q)}{cVL(p,p)} \tag{2.9}$$

Figure 2.9(b) illustrates a higher correlation, 0.9207, between *cnVL* (Equation 2.9) and the common pixel ratio. We also conduct a linear regression and obtain the results as follows:

$$R[V_{com}(q)] \approx k \cdot cnVL(p,q) + b \tag{2.10}$$

where $k = 1.012$ and $b = -0.1291$.

For instance, Figure 2.10 is a distribution of $cnVL$ in a virtual classroom with a 5x5 seats pattern (shown in Figure 2.6(a)); the 25 students (views) are indexed index from 1 to 25, p refers to primary view and q is the secondary view. In Figure 2.10, we can see that the distribution of $cnVL(p,q)$ reflects the distribution of common ratio between two views p and q. Specifically, if two views are captured in closer positions, the $cnVL$ tends to be larger; otherwise, $cnVL$ will decrease. Within these 25 views, $cnVL$ attains 1 when views p and q are selected as the same view, while the minimum is attained as 0.1 when *primary view* and *secondary view* are assigned as view #25 and view #1 respectively.

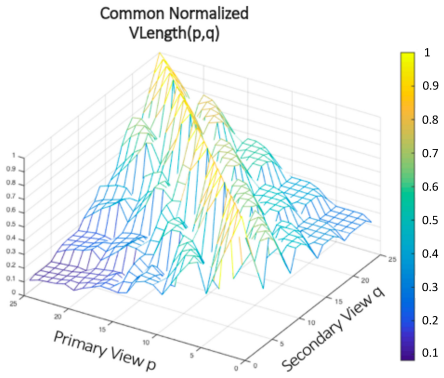


Figure 2.10: The *cnVL(p,q)* between every two views in a 5x5 seat pattern, where the p and q represent the primary view index and secondary view index respectively.

The calculation of $cnVL$ values is much simpler and faster compared to obtaining the actual *Common Pixel Ratio* between every two views using graphic rendering. Therefore, due to the high correlation between $cnVL$ and common pixel ratio, we can approximate the common pixel ratio with the metric $cnVL$, which greatly saves runtime. Subsequently, we update the calculation of $D(p,q)$ in our problem formulation (Equation 2.1) as follows:

$$\min P_{total} \Leftrightarrow \min\{\sum_{p\in S_p}\sum_{q\in S_q} b_{p,q}\cdot D(p,q) + |S_p|\} \tag{2.11}$$

where

$$\begin{aligned} D(p,q) &= R[V_{res}(q)] \\ &= 1 - R[V_{com}(q)] \\ &\approx 1 - (k\cdot cnVL(p,q) + b) \end{aligned} \tag{2.12}$$

In this way, we can use the newly defined metric $cnVL$ to estimate the common pixel ratio $R[V_{com}]$ between any pair of views. We can also denote the residual pixel ratio as $D(p,q)$ between the *primary view* p and the *secondary view* q, and calculate D based on metric $cnVL$ as Equation 2.12.

2.4.4 Grouping Strategy

In a virtual space with a large number of users, if we have all the users in the same group (i.e. one primary view and all others have secondary views), then the number of residual pixels needed may become very large, with some residual views almost equaling the size of the primary view. On the other hand, if we divide users into too many groups (e.g. each one as a unique group), then the size of residual views may greatly decrease (e.g. even to 0) but the number of common pixels may become very large (e.g. even equal to the total pixel number for all views), thus leading to overall high bitrate needed. Hence the challenge is to identify the most appropriate partitioning (groups) of the users of the virtual space so overall bitrate needed to transmit their views is minimized (Equation 2.11). For example, for better bitrate reduction, users should be grouped with large portion of shared pixels so that common view can be maximized and residual views can be minimized.

Since finding the optimal groups to minimize the overall bitrate (Equation 2.11) is NP-Hard, we first propose a heuristic algorithm $VS-GRP$ (presented in Algorithm 1), which we show in Section 2.5.4 to have linear complexity in terms of number of users. So that we can evaluate the performance of $VS-GRP$, we subsequently present an optimal but exponential time complexity algorithm $VS-OPT$ (shown in Algorithm 2), and compare their performances in Section 2.5.

We next describe the steps of Algorithm $VS-GRP$. In Lines 2-11, we make an implement to take the minimum value of evl^* (i.e. the value of P_{total} in Equation 2.11), traversing all potential optimized grouping strategies for K groups for I_w iterations. In Lines 4-5, we use the procedures $Kmeans$ and $Evaluate$ to obtain optimized groups and calculate their value of evl^*. The $Kmeans$ procedure consists of three parts: $Initialize$, $Optimization$ and $Update$.

We use the $Initialize(k, i_w)$ procedure to initialize k cluster centers for k groups (i,e, each group has a cluster center). In iteration number $i_w = 1$, we will give a priority index to each user. The numbering method is giving the user with smaller y a smaller priority index, and if two users

Algorithm 1 VS-GRP Algorithm

Inputs: Number of users n, locations (x_i, y_i) for each user i and distance matrix D.

Output: Binary indicator matrix B, such that bitrate needed for all user views is minimized (Equation 2.11).

$evl \leftarrow +\infty$
for $k = 1 : K$ **do**
  for $i_w = 1 : I_w$ **do**
    $B^* \leftarrow Kmeans(k, i_w)$
    $evl^* \leftarrow Evaluate(B^*)$
    if $evl > evl^*$ **then**
      $B \leftarrow B^*$
      $evl \leftarrow evl^*$
    end if
  end for
end for
return B

Procedure $Kmeans(k, i_w)$:

$S_p, S_q \leftarrow Initialize(k, i_w)$
$S_p^* \leftarrow [\,]$
while $S_p^* \neq S_p$ **do**
  $B^* \leftarrow zeros(n, n)$
  for $q \in S_q$ **do**
    $p \leftarrow \arg\min_p \{D(p,q), p \in S_p\}$
    $b_{p,q}^* \leftarrow 1$
  end for
  $S_p^* \leftarrow S_p$
  $S_p, S_q \leftarrow Update(B^*)$
end while
return B^*

Procedure $Evaluate(B^*)$:

$evl^* \leftarrow 0$
for $p \in S_p$ **do**
  for $q \in S_q$ **do**
    $evl^* \leftarrow evl^* + (b_{p,q}^* \cdot D(p,q) + k)$
  end for
end for
return evl^*

Algorithm 2 VS-OPT Algorithm

Inputs: Number of users n, locations (x_i, y_i) for each user i and distance matrix D.

Output: Binary indicator matrix B, such that bitrate needed for all user views is minimized (Equation 2.11).

$evl \leftarrow +\infty$
$v \leftarrow [1:n]$
for $k = 1:n$ **do**
  $S_{s_p} \leftarrow combntns(v,k)$
  $n_{comb} \leftarrow size(S_{s_p}, 1)$
  for $i = 1:n_{comb}$ **do**
    $B^* \leftarrow zeros(n,n)$
    $S_p \leftarrow S_{s_p}(i,:)$
    $S_q \leftarrow setdiff(v, S_p)$
    for $j = 1:(n-k)$ **do**
      $q \leftarrow S_q(j)$
      $p \leftarrow \arg\min_p \{D(p,q), p \in S_p\}$
      $b^*_{p,q} \leftarrow 1$
    end for
    $evl^* \leftarrow Evaluate(B^*)$
    if $evl > evl^*$ **then**
      $B \leftarrow B^*$
      $evl \leftarrow evl^*$
    end if
  end for
end for
return B

with equal y, we give the user with smaller x a smaller priority index. In this way, each user will be numbered with a priority index, from 1 to n. In our algorithm, we initialize k cluster centers as $\lfloor \frac{n}{k} \rfloor, \lfloor \frac{2n}{k} \rfloor, \ldots, n$. After that, we will carry out the Optimization and Update procedures. Otherwise, if iteration number $i_w \neq 1$, we initialize k cluster centers randomly by picking k different users from n users.

Then in Lines 3-11 of *Kmeans* procedure, we implement the optimization by updating the users with *primary* and *secondary views* continuously until the set of primary views S_p and the set of secondary views S_q do not change any more. The number of the iterations in Lines 3-11 is denoted as I_u, which will be discussed further in Section 2.5.4. In $Optimization$ procedure (Lines 5-7 of *Kmeans* procedure), we will find out the which view p in the set of primary views S_p has the minimum $D(p,q)$ and then select it as the *primary view* within this group. In $Update$ procedure, we will update the S_p, S_q by assigning the nearest one to the average location of all group members as the user with *primary view*.

In Lines 2-6 of $Evaluate$ procedure, we calculate the value of evl^* (i.e. the value of P_{total} in Equation 2.11) by adding up each item in Equation 2.11. With all these procedures, we can finally obtain the optimized grouping strategy with our algorithm.

As stated earlier, since algorithm $VS-GRP$ is heuristic, so that we can evaluate its performance, we also present an optimal algorithm $VS-OPT$, which considers forming all possible groups by using all possible assignments of primary views and secondary views in an exhaustive manner. Then by comparing the value of P_{total} in Equation 2.11 for every possible assignment and finding out the groups for which is minimum, we can obtain the optimal groups. Specifically, we present this algorithm with Matlab implementation. The function $combntns(set, n_0)$ returns a matrix whose rows are the various combinations that can be taken of the elements of the vector *set* of length n_0 while the function $setdiff(A,B)$ returns the data in A that is not in B, with no repetitions. In this way, we can do a traversal of all possible combinations of users assigned with primary view or secondary view. For every possible combination, we will calculate its value of

evl^*. Finally, we will obtain the users assignment with the minimum evl^*, which is the optimal strategy for grouping.

Next, we briefly analyze the time complexity of the optimal exhaustive algorithm $VS-OPT$ as well as our proposed grouping algorithm $VS-GRP$. Let n be the number of users in the virtual space, and k be the desired number of groups. For the optimal exhaustive algorithm $VS-OPT$, there are C_n^k possibilities of choosing k primary views (for k groups). After deciding k users with primary views, we have $(n-k)$ users to be assigned with secondary views. For each of the unassigned users, there are k possible choices (each view can be assigned to either of the k groups), leading to a multiplicative factor of k^{n-k} for each C_n^k. Therefore, the time complexity of the optimal exhaustive algorithm $VS-OPT$ is $O(C_n^k k^{n-k})$.

In contrast, our proposed grouping algorithm has a computation complexity of $O(nKI_u)$, where n is the number of users, I_u is the number of iterations in *Kmeans* procedure and K is the maximum number of groups we traverse. Next, we empirically estimate upper bound for K and I_u to validate our complexity analysis in Section 2.5.4.

2.5 Experimental Results

In this section, we describe our experimental setup and results. We use an Intel Core i7 Quad-Core processor with 32GB RAM and implement our approach in MATLAB. We demonstrate the performance of our grouping algorithm ($VS-GRP$) with our proposed evaluation metrics side by side to an optimal, exhaustive grouping algorithm ($VS-OPT$). We demonstrate experiments by using (i) the virtual classroom application with a regular students' seat (view) pattern; and (ii) the virtual gallery application with the location of visitors (views) randomly distributed. To further demonstrate the robustness, we also show experiments for a virtual classroom, with vacant seats (views). Then we present a complexity analysis, and a congestion-related latency study in this section. Note that we select virtual classroom and virtual gallery applications

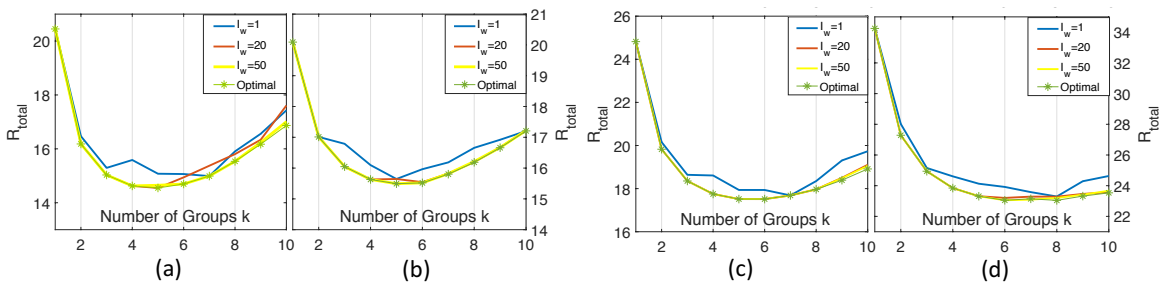


Figure 2.11: Total number of transmitted pixels divided by number of pixels in one frame, versus number of groups k for four different seats pattern virtual classrooms. Sub-graphs present seat patterns of (a) 7x5, (b) 5x7, (c) 7x6, (d) 7x7 respectively.

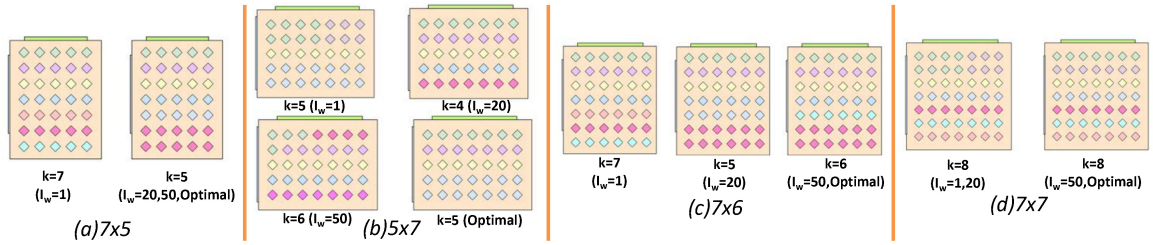


Figure 2.12: Grouping results of $I_w = 1, 20, 50$ and *Optimal* cases for four different seats pattern virtual classrooms. Sub-graphs (a)-(d) present seat patterns of 7x5, 5x7, 7x6, 7x7 respectively.

to validate the effectiveness of our approach, since they have different characteristics: the users in the classroom have a regular distribution pattern while the users in the gallery are more randomly distributed in the virtual space. For the virtual classroom, we explore two scenarios, without and with vacant seats, since vacant seats can affect the user distribution pattern and hence the grouping results.

2.5.1 Virtual Classroom

Without Vacant Seats

To evaluate the effectiveness of our approach, we perform our experiments using the virtual classroom application, with different seating patterns, and all seats occupied.

We first consider a virtual classroom with a seating configuration having the same horizontal and vertical seat spacings ($Spacing_V = Spacing_H = 2$). Figure 2.11 shows the total number

Table 2.6: Experimental Results for four different seats pattern virtual classrooms.

Space	***Alg.***	***Run Time***	k	R_{total}	***Pixel Saving***
7x5	$I_w = 1$	0.0035s	7	14.99	57.2%
	$I_w = 20$	0.0707s	5	14.55	58.4%
	$I_w = 50$	0.1660s	5	14.55	58.4%
	Optimal	$> 1h$	5	14.55	58.4%
5x7	$I_w = 1$	0.0036s	5	15.65	55.3%
	$I_w = 20$	0.0683s	4	15.63	55.3%
	$I_w = 50$	0.1715s	6	15.56	55.5%
	Optimal	$> 1h$	5	15.49	55.7%
7x6	$I_w = 1$	0.0044s	7	17.68	57.9%
	$I_w = 20$	0.0774s	5	17.55	58.2%
	$I_w = 50$	0.1917s	6	17.51	58.3%
	Optimal	$> 1h$	6	17.51	58.3%
7x7	$I_w = 1$	0.0046s	8	23.30	58.4%
	$I_w = 20$	0.0950s	8	23.30	58.4%
	$I_w = 50$	0.2349s	8	23.20	58.6%
	Optimal	$> 1h$	8	23.20	58.6%

of pixels (normalized to frame size) to be transmitted with different number of groups. R_{total} is the ratio of total number of pixels to be transmitted over total number of pixels in one frame. For example, in a 20-user scenario, $R_{total} = 14.99$ means that we only need $14.99 \times P_{frame}$ (e.g. 1920x1080 pixels) instead of $20 \times P_{frame}$. We use $I_w = 1, 20, 50$ for our proposed algorithm $VS-GRP$, and compare to the optimal algorithm $VS-OPT$ ($Optimal$), using four different seat patterns – 7x5, 5x7, 7x6 and 7x7. For a given number of groups k, we can see that there is a tradeoff between solution quality (total number of pixels to be transmitted) and I_w. Larger I_w benefits the grouping result (closer to optimal R_{total} as well as P_{total}) but also consumes more runtime, as shown in Table 2.6.

From Figure 2.11, we can also find a preference for k as well. Too few groups may result in many distinct secondary views, thus there is not much common view. Too many groups may result in too many primary views, leaving insufficient secondary views. Figure 2.12 shows examples of some grouping results for the four seat patterns. Each diamond denotes a seat and each color represents a unique group. We can see that as I_w increases, our grouping algorithm

performs closer to the optimal solution. For instance, Figure 2.12(a) shows that by using $I_w = 1$ we obtain a grouping strategy (dividing users into 7 groups) while by employing $I_w = 20, 50$ or optimal algorithm $VS-OPT$ we receive the same grouping result (5 groups). Similarly, we present the grouping results for other 3 seating patterns in Figure 2.11(b)-(d) respectively.

Table 2.6 summaries the quality of results in terms of runtime, R_{total}, and pixel savings for the four seat patterns. Our proposed approach consumes up to 0.24s for grouping, compared to one hour of the optimal exhaustive grouping algorithm, while giving at most 0.2% degradation of bitrate saving when $I_w = 50$. For a more real-time-critical scenario, we only need 5ms for grouping when $I_w = 1$, with a maximum degradation of 1.2% in pixel savings. Overall, our proposed approach can achieve more than half of the pixel savings for all four seat patterns within milliseconds. To be specific, as seen from Table 2.6, our proposed algorithm $VS-GRP$ (such as $I_w = 1$) with hybrid-cast approach reduces the pixels needed by 57.2%, 55.3%, 57.9% and 58.4% compared to the conventional approach of transmitting all the 7x5, 5x7, 7x6, 7x7 views as individual unicast streams. Also, our proposed algorithm $VS-GRP$ is able to reduce the pixels needed only by a marginal 1.2%, 0.4%, 0.4% and 0.2% less than the optimal algorithm $VS-OPT$, while ensuring that it can be run in real-time (run-time of 0.0040 seconds on average when $I_w = 1$) compared to more than an hour of run-time for the optimal algorithm. The results when $I_w = 20, 50$ are really similar to optimal results but the run time will be a little more than the run time when $I_w = 1$.

Since the above grouping results present a tendency for horizontal grouping (Figure 2.13), we then perform experiments with a virtual classroom with different seating configuration, where the spacing between vertical seats is less than spacing between horizontal seats ($Spacing_V = 1$, $Spacing_H = 2$). Figure 2.13(a)(b) show grouping results for two seat patterns (6x7 and 7x7) respectively. Each diamond denotes a seat and each color represents a unique group. We can observe that for the new seat configuration, our proposed grouping algorithm produces more groups in vertical direction, and in general, groups consisting of both horizontal and vertical

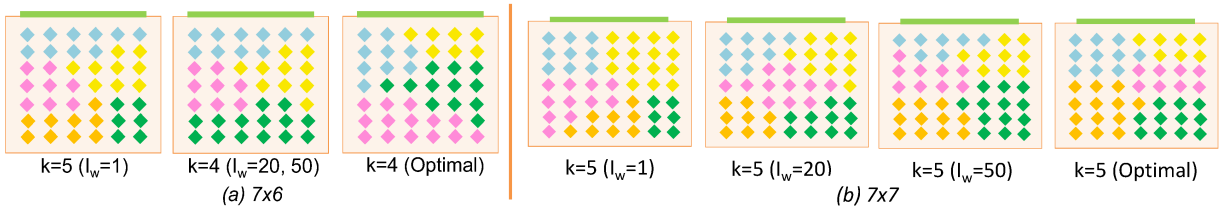


Figure 2.13: Grouping results of $I_w = 1, 20, 50$ and *Optimal* cases for two different seats pattern virtual classrooms. The sub-graphs (a)(b) present seat pattern of 7x6 and 7x7 respectively.

neighbors.

Besides showing the real-time performance of our proposed grouping algorithm, we analyze the end-to-end latency consumed for our cloud-based virtual space approach in Section 2.5.7. Our analysis shows server-side latency of 9.5-19.5 ms and client-side latency of about 5.5 ms, per rendered frame.

With Vacant Seats

To demonstrate the robustness of bitrate savings, we consider a more irregular seat pattern in the virtual classroom. In our experiment, we use a 7x5 seat pattern, and each seat has a vacant probability of 20%. (This scenario can match to a virtual class where each student may choose to drop the class with a fixed probability.) We randomly generate 1000 different vacancy patterns. Figure 2.14 shows the probability density function and cumulative distribution function for the 1000 vacancy patterns. For each vacancy pattern, we apply our proposed grouping algorithm $VS-GRP$. Figure 2.15 shows the results in terms of the number of groups and pixel savings obtained. The empirical results demonstrate that our algorithm performs well with vacant seats. We have achieved similar pixel saving ratio compared to a virtual classroom without vacancy.

Interestingly, as shown in Figure 2.15, the average group number grows approximately linearly with the increase in the number of occupied seats. Also, we can achieve more pixel saving with more occupied seats, while the exact location of those vacancies does not impact much to the pixel saving. Overall, we still achieve more than half of pixel savings for all configurations,

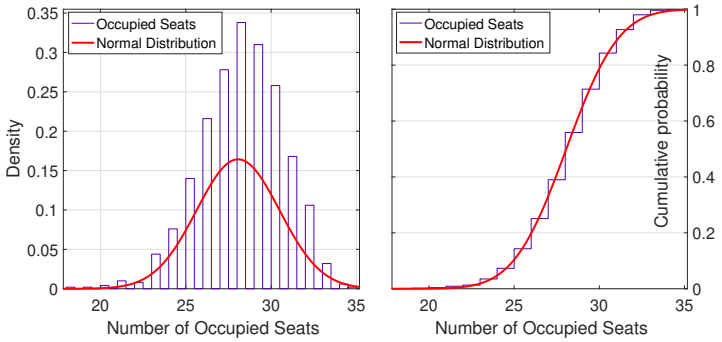


Figure 2.14: The probability density function (PDF) figure (*left*) and cumulative distribution function (CDF) figure (*right*) of occupied seats covering all 1000 samples. The red line shows the normal distribution fitting.

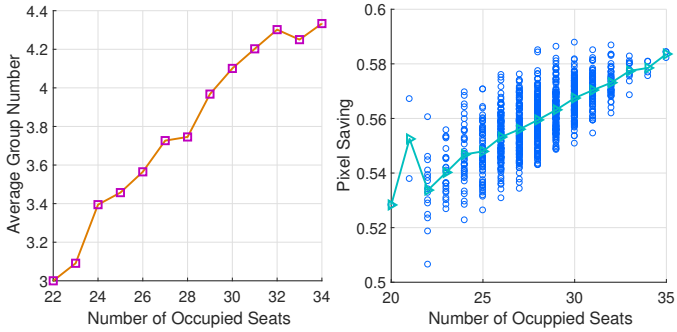


Figure 2.15: The left sub-graph shows the average number of groups versus number of occupied seats while the right sub-graph demonstrates the pixel saving versus the number of occupied sets. Specifically, in the latter, the green line represents the average pixel saving.

indicating the robustness of our algorithm. Specifically, in the right sub-figure of Figure 2.15, the x-axis is the number of occupied seats and y-axis is the pixel saving. Every blue point presents the number of occupied seats and the pixel saving achieved by using our proposed approach. The green line and the triangle point within it represents the average pixel saving corresponding to the number of occupied seats. For example, the point (20, 0.53) indicates we can achieve approximately 53% percent pixel saving to transmit all the 20 views (for 20 occupied seats) with our proposed approach. It is demonstrated that the pixel saving will increase linearly as the number of occupied seats increases. We can observe that the pixel saving will not be affected by the position of vacant seats, but rather the number of vacant (as well as occupied) seats.

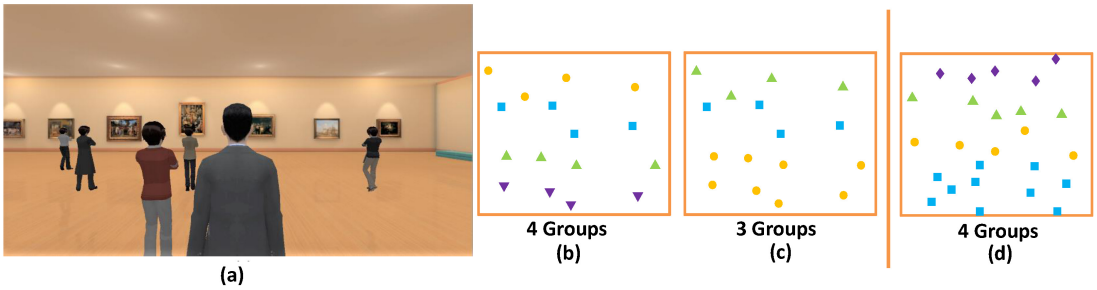


Figure 2.16: A virtual gallery scene where the visitors are randomly distributed is demonstrated in (a); Grouping results for 16 users is presented in (b)(c) while results for 25 users in shown in (d).

2.5.2 Virtual Gallery

Next, we show results of applying our approach to the virtual gallery application. In the virtual gallery shown in Figure 2.16(a), visitors (views) are randomly distributed. We conduct two sets of experiments, with 16 and 25 users, respectively. Two different grouping results for 16 users are presented in Figures 2.16(b)(c), while one grouping result for 25 users is shown in Figure 2.16(d). Table 2.7 summaries the experimental results in terms of the parameters, pixel savings obtained, and run time, compared to the optimal algorithm $VS-OPT$. We can see that for 16 users ($Scene_1$), our proposed algorithm $VS-GRP$ ($I_w = 1$) can obtain pixel saving of 55.0%, with only 1.9% degradation compared to the optimal result, while consumes only 4.3ms. The corresponding assignment for $I_w = 1$ and $k = 4$ is shown in Figure 2.16(b). The grouping assignments for $I_w = 20$ and $Optimal$ are the same, and are demonstrated in Figure 2.16(c). For 25 users ($Scene_2$), the grouping assignments for $I_w = 1, 20$ and $Optimal$ are the same, shown in Figure 2.16(d). The above results again demonstrate the effectiveness of our proposed grouping algorithm in a virtual space application with randomly distributed views. Overall, the pixel savings are similar compared to a virtual classroom application, while suitable for real-time-critical applications.

Table 2.7: Experimental results for multiple users in gallery scene.

Scene	***Alg.***	k	***Pixel Saving***	***Run Time***
$Scene_1$	$I_w = 1$	4	55.0%	0.0043s
	$I_w = 20$	3	56.9%	0.0454s
	Optimal	3	56.9%	$> 1h$
$Scene_2$	$I_w = 1$	4	56.8%	0.0056s
	$I_w = 20$	4	56.8%	0.0533s
	Optimal	4	56.8%	$> 1h$

2.5.3 High Correlation between Pixel Ratio and Bitrate Saving

In this experiment, we validate the correlation between pixel ratio and bitrate savings. In our hybrid-cast approach, the total bitrate needed for the video streams is different from the raw pixel savings in that video streams are encoded in fixed frame size, and then transmitted. Note that in our case, this is equivalent to exploring whether there is high correlation between pixel ratio saving and bitrate saving. The reason is that in our experiments, we obtain pixel ratio and bitrate saving for two approaches (proposed approach and conventional approach) with the same setting of video resolution and framerate. The pixel ratio saving equals to pixel saving divided by number of pixel in one frame (e.g. 1920x1080 pixels) while the bitrate saving equals to the saving in number of bits times the value of framerate. Thus, high correlation (between pixel ratio saving and bitrate saving) and high correlation (between pixel saving and number of bits saving) are two equivalent statements in our discussion.

To demonstrate the above, we perform live experiments for the two virtual space applications. For the virtual classroom application, we assume that there are two students in the same row, sitting close to each other, and a teacher is walking fast at the front of the classroom. The view centers of the students keep moving and are always towards the teacher, i.e. focusing continuously with movement of the teacher. We record 750 frames (30s assuming 25fps). Then we encode the video using H.264 after applying our approach. In our implementation, we only need to broadcast primary views and unicast residual views (instead of secondary views). Residual views are encoded full frame size, with common pixels replaced by black pixels. (Thus, we don't

Table 2.8: High correlation between saving in bitrates and saving in pixel ratio.

Space	***Items***	***Conv. Approach***	***Prop. Approach***	***Saving***
VC	*Total Bitrate*	43.7Mbps	26.7Mbps	39.0%
	Avg. Pixel Ratio	2.00	1.18	41.0%
VG	*Total Bitrate*	27.6Mbps	23.3Mbps	15.7%
	Avg. Pixel Ratio	2.00	1.63	18.5%

explore any further pixel savings by downscaling.) We also perform the live experiment using the virtual gallery application. In our setup, parameters are set the same except that two visitors stand further away from each other, and both watch a slowly moving tour guide.

Table 2.8 summarizes the experimental results for the above two scenarios, in terms of the average pixel ratio of the rendered frames, the total bitrate needed for the resulting encoded video frames, and the savings achieved by using the proposed approach for both the average pixel ratio and the total video bitrate. For the virtual classroom application, the saving in average pixel ratio is 41.0% while the saving in the total bitrate is 39.0%. For the virtual gallery application, our proposed approach produces 18.5% saving in average pixel ratio while the saving in total video bitrate is 15.7%. The results show that there is a high correlation between pixel ratio saving and bitrate saving for both the virtual space applications. And as shown in Section 2.5.6, the bitrate saving using our approach also leads to substantial saving of cloud cost.

2.5.4 Empirical Results Validating Complexity Analysis

As is mentioned in Section 2.4.4, our proposed grouping algorithm has a computation complexity of $O(nKI_u)$, where n is the number of users, I_u is the number of iterations in *Kmeans* procedure and K is the maximum number of groups we traverse. Next, we empirically estimate upper bound for K and I_u, and show that the complexity of our proposed algorithm can be practically simplified as $O(n)$. We perform the experiments using the virtual classroom application, with seat patterns setting from 4x4, to 10x10. For a given seat pattern, we generate 100 different seat pattern configurations and implement our grouping algorithm. We also calculate the average

of parameters (i.e. k, I_u, runtime) for these seat pattern configurations.

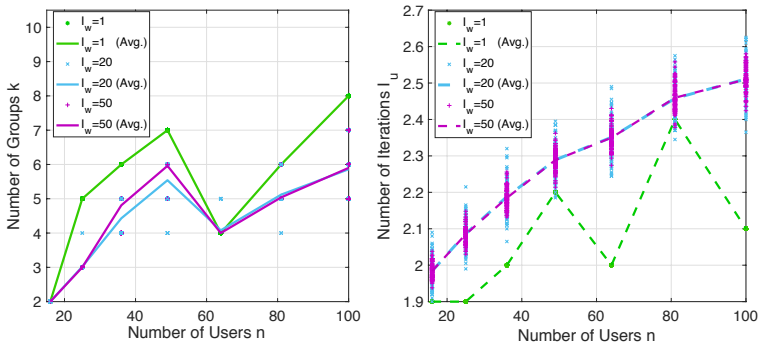


Figure 2.17: (a) Number of Groups versus the number of users. The line represents the average number of groups given the number of users; (b) Number of Iterations as I_u versus the number of users. The dashed line demonstrates the average I_u considering the number of users.

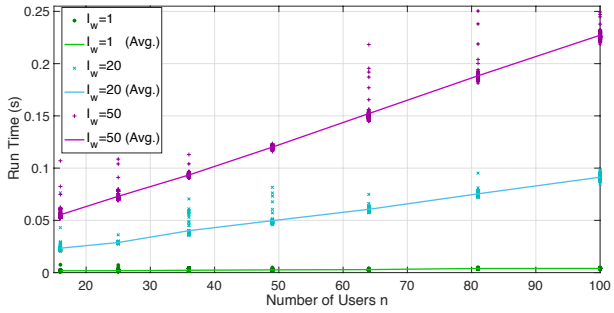


Figure 2.18: The run time versus the number of users when $I_w = 1, 20, 50$ respectively. The line represents the average run time given the number of users.

Specifically, Figure 2.17(a) demonstrates the optimal number of groups from our algorithm versus total number of users. The three lines represent the optimal, average number of groups across all seat patterns. We observe that the number of groups show similar trends and are generally smaller than $\sqrt{n}$, across all seat patterns. In our implementation, we empirically bound the maximum number of groups $K = 10$.

Figure 2.17(b) shows the number of iterations I_u versus the number of users. The dashed line demonstrates the average I_u considering the number of users. We can see an increase in I_u with the increase in n when the $I_w = 20$ and 50, and similarly for $I_w = 1$ with some fluctuations. The results indicate a steady, if not increasing I_u.

In addition, we explore the relation between the runtime and the total number of users.

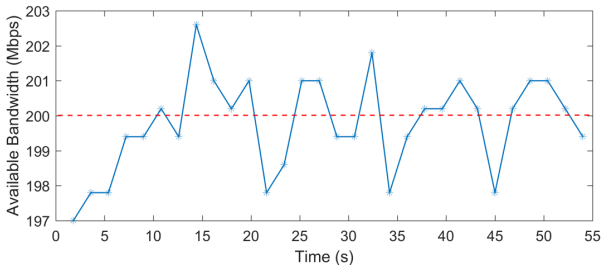


Figure 2.19: The available bandwidth of the network in our experiment.

Figure 2.18 shows the runtime vs. the number of users when $I_w = 1, 20, 50$. The lines represent the average runtime across all seat patterns. The results show that the average runtime grows linearly with the total number of user, i.e., $O(nKI_u)$ can be simplified as $O(n)$ in that: (i) K is bounded to be 10, and (ii) I_u is practically a small constant. Overall, our proposed algorithm $VS-GRP$ can be executed in real-time with linear time complexity, with similar solution quality to the optimal algorithm, while consumes negligible runtime. As presented in Section 2.5.8, we also perform experiments on the virtual gallery application to validate our complexity analysis.

2.5.5 Congestion-related Latency

In this experiment, we show the congestion-related latency improvement by utilizing our hybrid-cast approach. For the server side, we deploy our model in an Ubuntu 16.04 TLS system hosted on the Amazon Web Service (AWS) [47] server, equipped with a 2.6GHz Intel Xeon processor, 16GB RAM and a NVIDIA GRID GPU. For the client side, we simulate a 10-user scenario by deploying to 10 nodes, each with the same, above mentioned machine configuration. We assume that there is only one group (i.e. one user with a primary view and nine users with secondary views). We use DummyNet [48] to emulate the wireless network, specifically network bandwidth profiles experienced by the virtual space data transmitted from the AWS server.

In order to measure the latency from AWS cloud server to the user's client, we performed experiments to record the round-trip delay (RTT) needed with different network bandwidth profiles. Figure 2.19 shows a fluctuating bandwidth profile (for 55s), with an average available

Table 2.9: Congestion-related latency in bandwidth-limited network.

Approaches	***Settings***	***Latency***
Conv. Approach	10 users (bitrate: 23Mbps*10)	Min≈600ms Max>1s
Prop. Approach	10 users (bitrate: 11Mbps*9+23Mbps)	Min<1ms Max≈5ms

bandwidth of approximately 200Mbps. We emulate two cases: using conventional method (all user views unicast) and proposed method (one group with 1 primary view broadcast and 9 residual views unicast). We record a 55s video of one user view (23Mbps) and a 55s video of its corresponding residual view (11Mbps) separately. We assume that within the 55s period, a user will receive approximately 11Mbps video as residual view with our proposed hybrid-cast approach while the user needs to receive 23Mbps with the conventional method. Table 2.9 reports the latency measured under these two different settings. In the setting using conventional method, since the realistic bitrate needed is larger than available bandwidth, the latency varies from a low of around 600ms to a high of larger than 1s. However, with our proposed hybrid-cast approach, the bandwidth needed is significantly decreased and thus the latency achieved is much smaller (i.e. $<$5ms).

Note the above analysis assumes all users are associated with the same base station (or at least same cellular gateway) and cloud server. Hence the bandwidth savings using our approach can help in reducing the congestion-related latency. However, as have been described in Section 2.3.2, if the users are not associated either with the same base station or the same cellular network gateway, latency reduction may not be achievable using the proposed approach. With the above assumption, the results demonstrate the significant advantage our proposed hybrid-cast approach may have to alleviate congestion-related latency in fluctuating and bandwidth limited wireless networks.

2.5.6 Cloud Cost Savings

Based on the experiments described in Section 2.5.5, we perform two experiments: (i) estimating the savings obtained by our proposed approach in terms of cloud bandwidth and the consequent cloud cost for different number of days, and (ii) calculating total monthly cost for different number of users. Specifically, we estimate the cloud cost charged using conventional approach and proposed approach when the service provider of the virtual space application uses AWS. We experimentally choose to have 10% users with primary views (23Mbps) and 90% users with residual views (11Mbps). We calculate corresponding cloud cost using the AWS pricing model in Table 2.10 [49].

Figure 2.20(a) shows the accumulative data transfer and total cost for 10 days (with assumption of 100 users in the virtual space). We can observe that the total cost (denoted by orange lines) increases sublinearly when the accumulative data transfer increases linearly due to segmented pricing in AWS pricing model. The total cost savings are the difference between two orange lines. We can see that the accumulative data transfer reaches around 240TB and 125TB respectively by employing conventional and proposed approaches, for 100 users in 10 days. The corresponding total cost saving is around $5000 for 100 users in 10 days using our proposed approach.

Figure 2.20(b) presents the total monthly cost versus different number of users. The blue bar and green bar denote the total monthly cost using conventional approach and our proposed approach respectively. We can observe that total monthly cost grows continuously with the increase in the number of users. For 1000 users, the total monthly costs are up to $367,800 and $170,000 respectively using conventional and proposed approaches, which can translate to a substantial cloud cost saving of $2.04M annually for the virtual space service provider. Note the above analysis assumes existence of multicast protocols in the cloud network connecting cloud servers to core network gateways. While such multicast protocols are being researched and developed, the service provider savings discussed in Section 2.5.6 will need to wait deployment

of the such protocols in the future.

Table 2.10: Pricing model per month on Amazon Web Service.

Data Transfer	*Price*
First 1GB	$0 per GB
Up to 10TB	$0.09 per GB
Next 40TB	$0.085 per GB
Next 100TB	$0.07 per GB
Next 350TB	$0.05 per GB

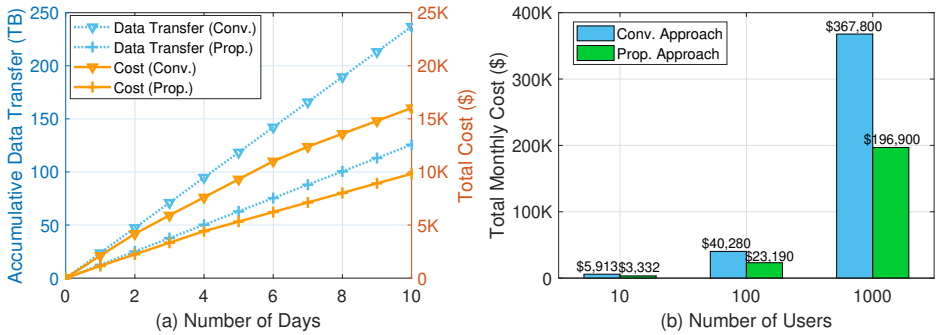


Figure 2.20: (a) Accumulative data transfer (left y-axis) and total cost (right y-axis) versus the number of days, for 100 users in the virtual space; (b) Total monthly cost versus different number of users in the virtual space.

2.5.7 Latency on the server and client sides

In this section, we show the latency for tasks on the server side and client side. In terms of latency for the rendering, encoding and decoding tasks, we give the estimated values according to their current advanced implementations [50–52]. Then we measure the latency for residual view calculation and synthesis by ourselves.

Table 2.11 and Table 2.12 present the latency for the various tasks performed on the server side and client side respectively, besides the latency of our proposed grouping algorithm discussed earlier. Specifically, on the server side, the tasks performed are real-time rendering, residual view calculation and encoding, in sequence. Meanwhile, the server will cluster users into different groups at short intervals (e.g. every 100ms) with our proposed grouping algorithm. On the client

Table 2.11: Latency for procedures on the server side.

Procedures	***Latency***
Rendering	4-9ms
Residual View Calculation	$\approx$2.5ms
Encoding	3-8ms

Table 2.12: Latency for procedures on the client side.

Procedures	***Latency***
Decoding	$\approx 3ms$
Synthesis	$\approx 2.5ms$

side, decoding task will be performed for primary users while decoding and synthesis (of primary and secondary views) tasks will be performed for secondary users.

Since the fundamental limitation of dumping the frame information in real-time from Unity [13], we demonstrate the ability of real-time residual view calculation using a separate program. Our program is written in C++ using 64-thread on a Xeon 2-CPU server. We calculate the residual view calculation for 1080p frame 100 times and report the average latency in Table 2.11 and Table 2.12. We can observe the average latency as 2.5ms. When using GPU parallel implementation (i.e. computation is executed for pixels in a frame in parallel instead of sequentially), we can expect smaller latency for these two tasks.

2.5.8 Complexity Analysis for Virtual Gallery

To analyze the complexity in the various scenarios, we also perform experiments on the virtual gallery application. We empirically estimate upper bound for maximum number of groups traversed K and the number of iterations in K-means procedure I_u, and validate that the complexity of our proposed algorithm can be practically simplified as $O(n)$. We conduct experiments with number of users (i.e. 16, 25, 36,... 100), randomly located in the virtual gallery space as explained before. For a given number of users, we generate 100 different user topologies and implement our grouping algorithm on every user topology. We also calculate the average of parameters (i.e.

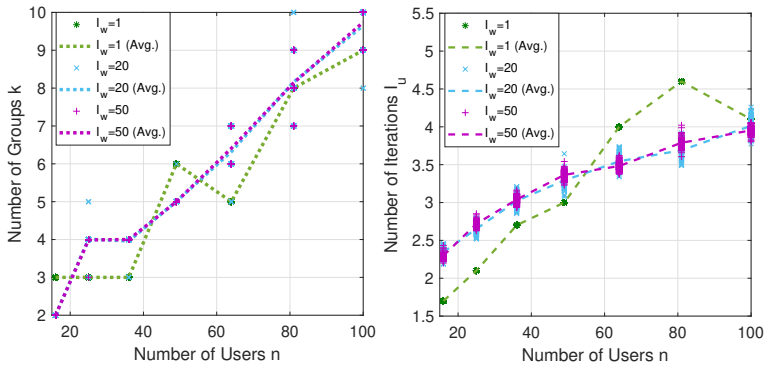


Figure 2.21: (a) Number of Groups versus the number of users in virtual gallery. The dashed line represents the average number of groups given the number of users; (b) Number of Iterations as I_u versus the number of users. The dashed line demonstrates the average I_u considering the number of users.

k, I_u, $runtime$) for these different user topologies.

Figure 2.21(a) shows the optimal number of groups k selected by our algorithm versus total number of users. The dashed lines represent, for different values of I_w used, the average number of groups across all user topologies, considering the number of users. We observe that the number of groups demonstrate similar trends and are generally smaller than $\sqrt{n}$, across all user topologies. In our implementation, we empirically bound the maximum number of groups $K = 10$. Figure 2.21(b) presents the number of iterations needed by our algorithm, I_u, versus the number of users. The dashed lines demonstrate the average I_u considering the number of users. We can see a slow increase in I_u with increase in number of users n. The values of number of iteration I_u are also identical when $I_w = 20$ and 50, and similarly for $I_w = 1$ with some fluctuations.

Figure 2.22 presents the runtime versus the number of users in virtual gallery applications. Compared with the empirical results in virtual classroom, we can see that the average runtime grows linearly with the total number of users. The difference is that the runtime for gallery application scenarios is slightly larger than for virtual classroom due to more randomness in user locations (topologies) in the former. The number of iterations I_u for virtual gallery cases is also slightly larger than for virtual classroom scenarios, as shown in Figure 2.17(b) and Figure 2.21(b).

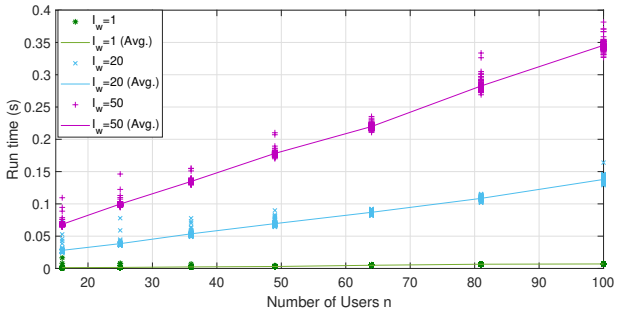


Figure 2.22: The run time versus the number of users when $I_w = 1, 20, 50$ respectively in virtual gallery. The line represents the average run time given the number of users.

2.6 Conclusion

In this chapter, we propose a multi-user hybrid-cast approach to significantly reduce the total bitrate needed to stream high-quality videos to multiple users in a virtual space application. Instead of unicasting the video of each user view, we introduce the novel approach which allows unicasting much lower-bandwidth residual views, together with one or more common view(s). Then we propose an efficient way of identifying common and residual views. To minimize the total bitrate, we develop a smart real-time algorithm for grouping the users of the virtual space, using a novel grouping metric. Our experimental results demonstrate the effectiveness of our proposed grouping algorithm both in terms of optimal performance and speed. Furthermore, the results show that the total bitrate needed to transmit multiple user views can be significantly reduced by up to 55%, and thus provide better user experience (less delay) under constrained network.

Our future research interests include: (i) integrating our hybrid-cast approach with a real network (e.g. Wi-Fi or cellular); (ii) studying data routing, forwarding and related protocols for data transmission in hybrid-cast approach; (iii) considering more complex virtual spaces, including irregular shapes and topologies; (iv) analyzing the case of having multiple primary views in the same group and the corresponding performance benefit.

2.7 Acknowledgments

Chapter 2 contains the reprints of Xueshi Hou, Yao Lu and Sujit Dey, "A Novel Hyper-cast Approach to Enable Cloud-based Virtual Classroom," *Proc. of IEEE International Symposium on Multimedia*, 2016; and Xueshi Hou, Yao Lu and Sujit Dey, "Novel Hybrid-Cast Approach to Reduce Bandwidth and Latency for Cloud-Based Virtual Reality," *ACM Transactions on Multimedia Computing, Communications, and Applications*, 2018. The dissertation author is the primary investigator and author of each of these papers.

Chapter 3

Predictive Adaptive Streaming to Enable Mobile 360-degree and VR Experiences

This chapter presents the methodology targeted to the mobile 360-degree video streaming. As 360-degree videos and virtual reality (VR) applications become popular for consumer and enterprise use cases, the desire to enable truly mobile experiences also increases. Delivering 360-degree videos and cloud/edge-based VR applications require ultra-high bandwidth and ultra-low latency [9], challenging to achieve with mobile networks. A common approach to reduce bandwidth is streaming only the field of view (FOV). However, extracting and transmitting the FOV in response to user head motion can add high latency, adversely affecting user experience. In this chapter, we propose a predictive adaptive streaming approach, where the predicted view with high predictive probability is adaptively encoded in relatively high quality according to bandwidth conditions and transmitted in advance, leading to a simultaneous reduction in bandwidth and latency. The predictive adaptive streaming method is based on a deep-learning-based viewpoint prediction model we develop, which uses past head motions to predict where a user will be looking in the 360-degree view. Using a very large dataset consisting of head motion traces from over 36,000 viewers for nineteen 360-degree/VR videos, we validate the ability of our predictive

adaptive streaming method to offer high-quality view while simultaneously significantly reducing bandwidth.

3.1 Introduction

Recently, 360-degree videos and virtual reality (VR) applications have attracted significant interest in various fields, including entertainment, education, manufacturing, transportation, healthcare, and other consumer-facing services. These applications exhibit enormous potential as the next generation of multimedia content to be adopted by enterprises and consumers via providing richer, more engaging and more immersive experiences. According to market research [53], VR and augmented reality (AR) ecosystem is predicted to be an $80 billion market by 2025, roughly the size of the desktop PC market today. However, several key hurdles need to be overcome for businesses and consumers to get fully on board with VR technology [1], such as cheaper price and compelling content, and most importantly a truly mobile VR experience, in line with the expectation and adoption of mobile experiences in almost all consumer and enterprise verticals today. Of particular interest is how to develop mobile (wireless and lightweight) head-mounted displays (HMDs), and how to enable VR experience on the mobile HMDs using bandwidth constrained mobile networks, while satisfying the ultra-low latency requirements.

Current widely used HMDs approximately include three types [54]: PC VR, console VR, mobile VR. Specifically, PC VR is tethered with PC [4, 55]; console VR is tethered with a game console [56]; mobile VR is untethered with PC/console but with a smartphone inside [8,57]. Since all the above HMDs perform rendering locally either on a smartphone tethered with the HMD, or on a computer/console tethered to the HMD, today's user experience lacks portability (when using a heavy HMD tethered to a smartphone) or mobility (when tethered to computer/console). To enable lighter mobile VR experience, we propose a cloud/edge-based solution. By performing the rendering on cloud/edge servers and streaming videos to users, we can complete the computation-

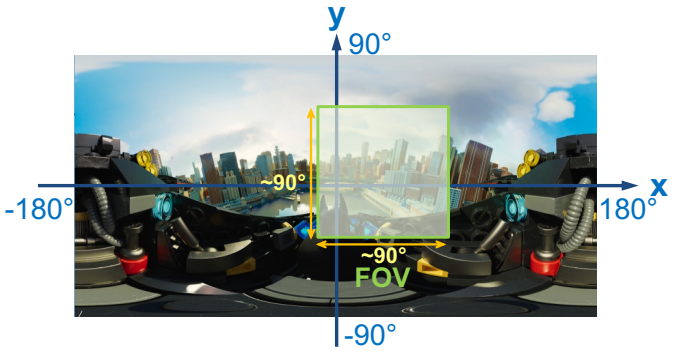


Figure 3.1: FOV in a 360-degree view.

intensive tasks on the cloud/edge server and thus enable mobile VR with lightweight VR glasses. The most challenging part of this solution is the ultra-high bandwidth and ultra-low latency requirements, since streaming 360-degree video causes tremendous bandwidth consumption and good user experiences require ultra-low latency (<20ms) [9, 10]. Various techniques have been developed for video content delivery such as adaptive streaming algorithms [58, 59], mobile edge caching placement algorithms [60] and hybrid multicast-unicast schemes [61, 62], but these approaches are designed for ordinary videos, and thus have not considered the scenario of 360-degree video streaming.

Motivated by this challenge, in this chapter, we propose a novel approach to enable mobile VR with prediction for head motions. Our basic idea comes from the following observations: the field of view (FOV) is 90°×90° for popular HMDs while the 360-degree view is 360°×180° in size (as is shown in Fig. 3.1). A common approach to reduce bandwidth is streaming only the FOV. However, extracting and transmitting the FOV in response to user head motion can add high latency, adversely affecting user experience. This motivates us to predict head motions. With prediction for head motion, our approach can address both bandwidth and latency challenges.

If we can predict head motion of users in the near future, we can achieve predictive rendering (in case of VR) and encoding on the edge device, and then stream the predicted view (i.e., a 360-degree view with more bits for the FOV tiles and less for non-FOV tiles) to the HMD in advance. Thus, latency needed will be significantly reduced since the view is delivered

and pre-buffered on the HMD. Moreover, in order to address the challenge of dynamically varying network bandwidth conditions, we use the viewpoint prediction to guide the bitrate adaptation of the stream (i.e., allocate more bits for the FOV tiles and less for non-FOV tiles to compose the predicted view). Thus, good user experience can be achieved under the dynamically varying network bandwidth conditions. Note that since *viewpoint* is defined as the center of FOV, prediction for head motions is equivalent to viewpoint prediction in this case. The main contributions of this chapter can be summarized as follows:

- We propose a new approach to enable truly mobile VR using wireless HMDs, where the rendering is performed on edge devices, and the ultra-low latency and high bandwidth requirements are addressed through a novel predictive adaptive streaming approach involving viewpoint prediction.

- We develop a viewpoint prediction method using deep learning to predict where a user will be looking into in the 360-degree view based on their past behavior. Using a very large dataset of real head motion traces from VR applications, we show the feasibility of our long short-term memory (LSTM) model with high accuracy.

- To address fluctuating and constrained wireless bandwidth available, we propose a novel predictive adaptive streaming algorithm. Given the available bandwidth constraint, it selects proper video encoding settings for each tile based on the viewpoint prediction such that user experience is maximized, i.e., PSNR in user FOV is maximized, where user FOV is defined as the actual user field of view in size of $90^\circ \times 90^\circ$.

- While adaptive streaming of 360-degree and VR videos has been proposed before, to the best of our knowledge, this is the first predictive adaptive streaming method proposed in the literature. Using a large-scale real head motion trace dataset, we demonstrate significant bandwidth savings while ensuring very high PSNR in the user FOV. We also demonstrate significant quality, bandwidth and network capacity benefits compared to streaming without

bitrate adaptation, and a recent adaptive streaming method which does not utilize prediction like our method does.

Note that a preliminary version of our work was published recently at a workshop [63], where we report on the predictive LSTM model and some preliminary results. In this article, we extend our approach by proposing a smart real-time predictive adaptive streaming algorithm, improving our proposed view generation strategy and conducting experiments on various 360-degree/VR videos.

The remainder of the chapter is organized as follows. In Section 3.2, we review related work. Section 3.3 introduces the system overview and problem definition. Section 3.4 describes our dataset and its characteristics. Section 3.5 and Section 3.6 describe our proposed predictive LSTM model and predictive adaptive streaming algorithms. We present our experimental results in Section 3.7 and conclude our work in Section 3.8.

3.2 Related Work

In this section, we review current work in the following topics related to our research.

FOV-guided streaming: Current FOV-guided 360-degree video streaming studies mainly consist of two types to address bandwidth challenge: *tiling* and *versioning* [64]. As for *tiling*, 360-degree video is spatially divided into *tiles* and only tiles within FOV are streamed at high quality while remaining tiles are streamed at lower qualities or not delivered at all [65–67]. In terms of *versioning*, the 360-degree video is encoded into multiple versions which have a different high-quality region, and viewers receive the appropriate version based on their own viewing direction [68]. The above methods are based on knowing the actual viewpoint of the user as it happens. Hence, while they can reduce bandwidth requirement of streaming 360-degree video, they cannot reduce the latency as rendering and encoding still need to be done in real-time after user FOV is determined. In contrast, our method aims to predict the user viewpoint and deliver the

predicted FOV in advance, thus eliminating the need for rendering (in case of VR) or extracting FOV (in case of 360-degree videos) and transmitting from servers over mobile networks after the user has changed viewpoint, and hence addressing the ultra-low latency requirement besides significantly reducing bandwidth.

Streaming with novel schemes: Some studies [69, 70] recognized ultra-low latency and ultra-high bandwidth challenges in the transmission of 360-degree videos and VR applications. [69] proposed a multipath cooperative routing scheme with software-defined networking architecture to reduce the delay and energy consumption of VR wireless transmissions in 5G small cell networks. [70] also studied a new link scheduling and adaptation scheme to reduce system latency and energy consumption in VR video streaming. By contrast, we are addressing the latency and bandwidth challenges with our proposed adaptive streaming approach, which can be applied on any existing wireless network without any special modification or provisioning of the network required by [69, 70].

Sequence prediction: Viewpoint prediction and related mobility prediction (since viewpoint prediction is equivalent to prediction for viewpoint mobility) both belong to the problem of *sequence prediction*, which is defined as predicting the next value(s) given a historical sequence [71]. We roughly summarize the approaches for sequence prediction as two types: traditional machine learning and deep learning methods. On one hand, traditional machine learning approaches such as randomized decision trees and forest [72, 73] have proven fast and effective performance for many sequence prediction tasks [74, 75]. Bootstrap-aggregated decision trees (BT) [72] is one of the most efficient methods among them. On the other hand, deep learning methods such as recurrent neural networks (RNN) and their variants including LSTM networks [76] and gated recurrent units (GRU) [77] have proven to be successful for sequence prediction tasks [78, 79]. Apart from RNN and their variants, there are also some studies [80, 81] using deep neural networks including deep belief networks (DBN) [82] and stacked sparse autoencoders (SAE) [83] to achieve sequence prediction. Among these deep learning methods, LSTM

recurrent neural networks show a good potential to capture the transition regularities of human movements since they have memory to learn the temporal dependence between observations (i.e., training data) [78, 79]. Inspired by this advantage, we design an LSTM model which can learn general head motion pattern and predict the future viewpoint position based on the past traces. Our prediction model shows promising results on a large-scale real head motion trace dataset.

Head motion prediction: Some studies [84–86] explore the feasibility of head motion prediction. Most of them used relatively simple models with euler angles or angular velocity as input without a tile-based perspective. Our proposed multi-layer LSTM model benefits from the design of our tile-based representation and the large-scale dataset, and thus performs better. Some studies [87, 88] also investigate more complicated prediction models to benefit 360-degree video experience. [87] achieves gaze prediction using the saliency maps and gaze trajectories (collected by an extra eye tracker), while [88] studies fixation prediction employing the saliency maps, motion maps, and head motion. However, the gaze prediction technique [87] cannot be implemented directly since most of current HMDs cannot track gaze, and the prediction models in [87, 88] are more time-consuming (i.e., 47ms and 50ms respectively) than our proposed prediction model (i.e., $<$2ms) because it needs more processing time of extracting image saliency maps and motion maps from videos. Our proposed prediction method achieves high accuracy in real time by using only head motion information, and thus are more efficient and concise for our current 360-degree video streaming scenarios to address the ultra-low latency challenge. Furthermore, these current studies [84–88] do not further consider the possibility of doing adaptive streaming using head motion prediction.

Adaptive streaming: Several techniques have been proposed for adaptive streaming for 360-degree videos [68, 89, 90]. [89] proposed to stream the 360-degree video based on average navigation likelihood, while [90] considered optimization based on expected quality distortion, spatial quality variance and temporal quality variance to do the 360-degree video streaming. [68] proposed to stream one of multiple representations of the same 360-degree video, where each

representation has a different quality emphasized region in the 360-degree view, such that bitrate fits the available throughput and a full quality region matches user's viewing. However, the above techniques [68, 89, 90] do not consider the problem of adaptive streaming in advance using prediction of user head motion to minimize latency.

To the best of our knowledge, we are the first to consider the problem of streaming predictively in advance of the actual user view so as to ensure ultra-low latency requirement of 360-degree video/VR, and using viewpoint prediction to guide the bitrate adaptation of the stream so that the highest user experience can be achieved under the dynamically varying network bandwidth conditions.

3.3 System Overview

In this section, we present an overview of our system. Note that our predictive adaptive streaming approach works for both 360-degree videos and cloud/edge-based VR applications, since it refers to (i) adaptively selecting encoded tiles (in case of 360-degree videos), and (ii) rendering the view and adaptively encoding tiles (in case of cloud/edge-based VR) depending on the predicted probability of each tile to belong to the user's actual FOV and bandwidth conditions. User's head motion as well as other controlling commands will be sent to the edge device, which performs viewpoint prediction and predictive rendering. The edge device can be either a Mobile Edge Computing node (MEC) in the mobile radio access or core network (Fig. 3.2(a)), or a Local Edge Computing node (LEC) located in the user premises or even his/her mobile device (Fig. 3.2(b)). Note that each of the above choices has tradeoffs. Use of MEC will allow for greater mobility of the VR user as compared to LEC, unless LEC is the user's mobile device, in which case the additional (computing) challenge of having to do predictive view generation in the mobile device will need to be addressed. On the other hand, use of MEC will add to more transmission delay of the rendered video than the use of LEC. Use of cloud servers can

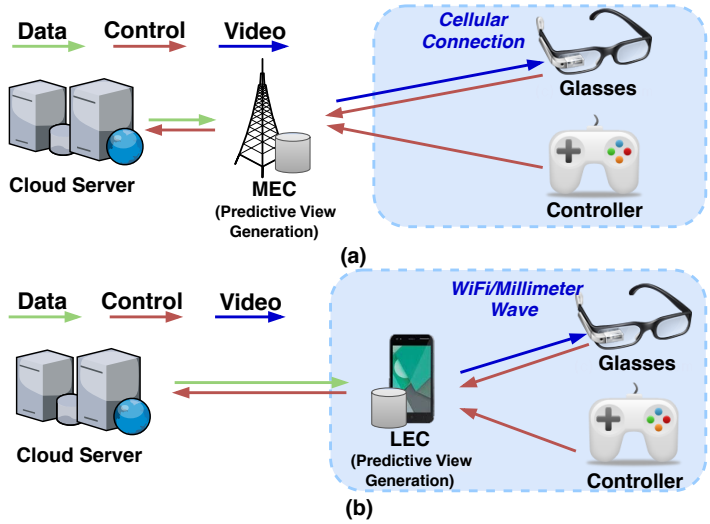


Figure 3.2: System overview.

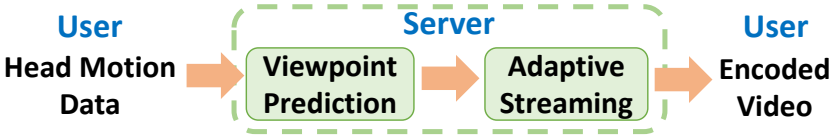


Figure 3.3: Proposed predictive adaptive streaming procedure.

also be considered to perform predictive view generation; this will allow complete mobility of VR users but will be more challenging in decreasing latency than the use of either MEC or LEC. This chapter will not specifically address the above tradeoffs and select either MEC or LEC. Instead, the predictive adaptive streaming technique we propose will apply to either of the edge device options. Note that the primary novelty of our approach in addressing the ultra-low latency requirement is in accurately predicting the user's view in advance and pre-delivering the predictive view so additional rendering (in case of VR) or extracting FOV (in case of 360-degree videos) and transmission time is avoided; our approach does not make any special use of edge devices, except that use of edge devices is recommended as opposed to cloud computing devices so as to reduce additional round-trip transmission latency between the computing device and the HMD.

Based on past few seconds of head motion and control data received from the user and using the viewpoint prediction model developed, the edge device will perform predictive adaptive

Table 3.1: VR Dataset statistics.

Categories	**#Video**	**Video Instances**
Movie Trailer	6	Kong VR, Batman Movie
Documentary	6	Fashion Show, Life on Mars
Scenery	4	Whale Encounter, Floating Markets
Entertainment	3	Roller Coaster, Bungee Jump

streaming algorithm, and stream the *predicted view* (i.e., a 360-degree view with more bits for the FOV tiles and less for non-FOV tiles) to the user HMD in advance. Later, the predicted view will be displayed on HMD and latency needed will be significantly reduced since the view is delivered and pre-buffered on the HMD before it is needed. The key to achieving efficient predictive adaptive streaming is to first solve the problem stated below.

Problem Statement: A viewpoint can occur in up to K different tiles in each time point (e.g., every 200ms). We decompose the whole predictive adaptive streaming method into two subtasks: *viewpoint prediction* and *adaptive streaming*, shown in Fig. 3.3. In *viewpoint prediction*, given previous and current viewpoint locations, our goal is to predict one or multiple tiles that the viewpoint will be in for the next time point. In *adaptive streaming*, according to the prediction results obtained from *viewpoint prediction*, we maximize user experience by smartly selecting proper video encoding (quantization step) settings of the video for each tile under dynamically varying network bandwidth conditions and do the streaming. After that the predicted view will be delivered to users.

3.4 Dataset and Its Characteristics

In this section, we first describe the dataset we use, and then show characteristics of the dataset using certain metrics we define.

To investigate viewpoint prediction in 360-degree videos, we conduct our study on a real head motion trace dataset that was collected by Samsung Electronics Company. The trace consists

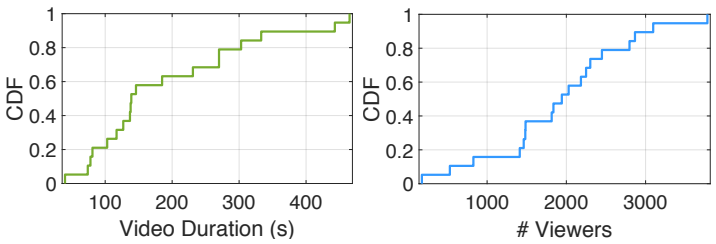


Figure 3.4: Statistics of dataset.

of head motion data from over 36,000 viewers during the week of November 2 – November 8, 2017, for 19 VR videos. Specifically, the frequency of head pose data was every 200ms on each HMD. The information reported includes the content ID, session timestamp, content timestamp, user ID and euler angles of HMD. The session timestamp and content timestamp refer to the time counted since application launches and the location in the video being played respectively, in milliseconds. Basic statistics of our head motion trace data are shown in Table 3.1. This dataset contains head pose data for 19 online VR videos, which are available on the Samsung VR website [91] and watched by a large number of viewers worldwide using their own HMD. We aggregate these videos by categories, i.e., movie trailer, documentary, scenery and entertainment. In Fig. 3.4, we plot the cumulative distribution function (CDF) of video duration and the number of viewers for each video. We can observe that over 80% of videos have more than 100s for duration and around 85% of videos have more than 1000 viewers. The large diversity and number of VR videos in the dataset, and the large number of viewers for each video, makes the dataset very suitable for developing and validating our viewpoint prediction method.

To depict key characteristics of the head motion and viewpoint changes in the dataset quantitatively, we offer the following definitions.

Definition 1— Head Motion Vector: Consider a viewer watching a video in certain time-points t_1 and t_2, where $t_1 < t_2$. We have corresponding head poses, which are denoted by $(x(t_1), y(t_1))$ and $(x(t_2), y(t_2))$ respectively. Then the head motion vector $(\triangle x, \triangle y)$ can be represented as $(x(t_2) - x(t_1), y(t_2) - y(t_1))$.

Definition 2— Head Motion Speed: The head motion speed v is defined as the distance

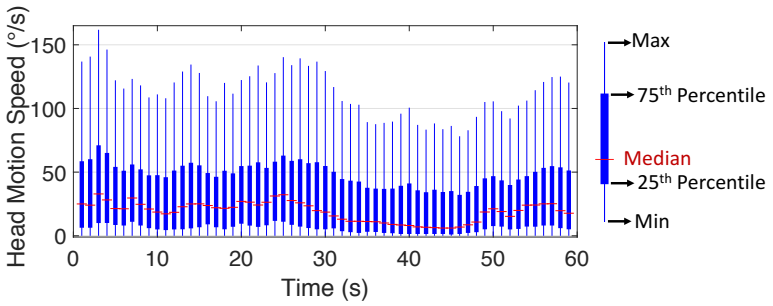


Figure 3.5: Head motion speed versus time in *Kong VR*.

the head moved divided by time.

$$v = \frac{\sqrt{\triangle x^2 + \triangle y^2}}{t_2 - t_1} \tag{3.1}$$

For *Kong VR* video in our dataset, we draw a boxplot in Fig. 3.5 to analyze head motion speed versus time. Fig. 3.5 shows head motion speed distribution for over 1500 viewers during 60s with this boxplot. Every dark blue strip represents the head motion speed distribution with an x-axis width of 1 (i.e., a width 1s in video time), whereas the height of a blue strip in the y-axis indicates the interquartile range of the head motion speed, reflecting the variability of the head motion speed. Additionally, each light blue line represents the corresponding maximum and minimum values and red symbols indicate the median head motion speed. From this boxplot, we observe that the distribution exhibits different properties when time changes. For instance, at the time point of 3s, the median head motion speed is as high as 35°/s, while 25 percent of viewers have a head motion speed larger than 75°/s and 75 percent of viewers have a head motion speed larger than 10°/s approximately. At another time point as 45s, median head motion is around 10°/s, while 25 percent of viewers have a head motion speed larger than 47°/s. The whole boxplot presents the challenging situation of predicting head motion since viewers may change viewing direction fast as well as frequently. Moreover, we can see interquartile range of head motion speed during 30-40s is around 5°/s-40°/s while during 50-60s interquartile range of head motion speed is 10°/s-50°/s approximately. Thus, we take the sequence of 30-40s as an example of *medium motion sequence* and the sequence of 50-60s as an instance of *high motion sequence*. As results presented in Section 3.7 show, viewpoint prediction and FOV generation for *high motion*

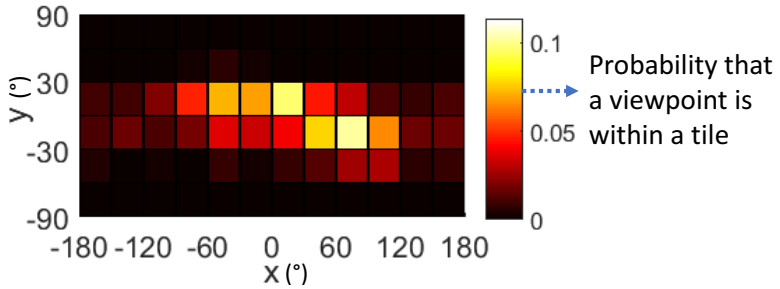
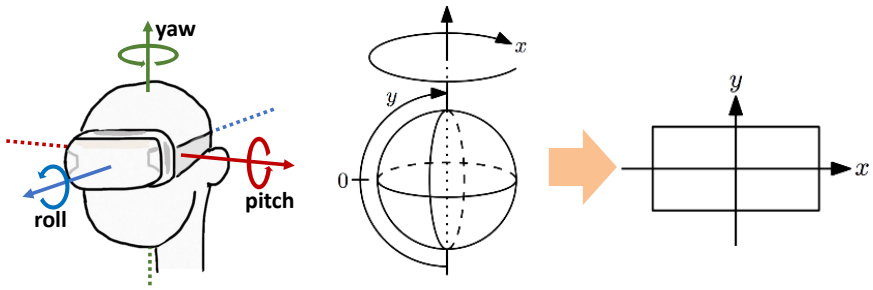


Figure 3.6: Example of attention map.

sequences are relatively more challenging than for *medium motion sequences*, resulting in either less FOV prediction accuracy, or larger FOV and hence less bandwidth savings.

Definition 3— Attention Map: For n viewers, content timestamps cts_1, cts_2 ($cts_1 < cts_2$) denote the video clip the viewers are watching. *Attention map* is defined as a series of probability that a viewpoint is within a tile for n viewers during time-period from cts_1 to cts_2. When we have K tiles in one 360-degree view, we have K elements (i.e., probabilities) in the attention map and the total sum of these probabilities is 1. When there are more tiles with relatively high probabilities, viewpoint prediction will be more challenging since different users may have multiple points of interest and require various FOVs.

Fig. 3.6 shows an example of attention map, demonstrating users' attention distribution (for over 1500 viewers) during 1s within the high motion sequence in *Kong VR* video [92] mentioned above. The value in legend represents the probability that a viewpoint is within a tile for n viewers during the given time-period. According to the legend, we can observe that the yellow tiles attract most attention and viewers are more likely to look at these areas. The yellow and red colors indicate that the probability that a viewpoint is within the corresponding tile is around 0.1 and 0.05 respectively for all n viewers during the given time-period, meaning this tile is of high interest for users. The attention map in Fig. 3.6 points to the feasibility of performing viewpoint prediction, since there are always areas attracting more attention than remaining areas within a 360-degree view. On the other hand, the attention map in Fig. 3.6 shows multiple tiles (as high as 11 tiles) have relatively high probabilities (0.05-0.1), indicating the difficulty of predicting



(a) Euler Coordinates (b) Projection

Figure 3.7: The viewpoint representation, projected into coordinates in equirectangular map.

viewpoint accurately. By visualizing a series of consecutive attention maps in a given sequence, we can observe the changes of viewpoint (as well as user attention) continuously. With proposed metrics such as head motion speed and attention map, we can characterize the viewpoint as well as user attention from both temporal and spatial perspectives.

3.5 Viewpoint Prediction

In this section, we describe our methodology of *viewpoint prediction*. Given previous and current viewpoint locations, our goal is to predict one or multiple tiles that the viewpoint will be in for the next time point. In our dataset, head motion files include user information, timestamp (time in video content), euler angles (pitch, yaw, roll), etc. Euler angles are shown in Fig. 3.7(a)) and timestamps appear each 200ms. We transform euler angles into the variables x, y in the equirectangular map [93] for 360-degree view, which is presented in Fig. 3.7(b). Variables x and y are within $(-180, 180]$ and $[-90, 90]$ degrees respectively.

We use tile-based format for viewpoint feature representation. With each grid size as $30° \times 30°$, the 360-degree view can be divided into 72 tiles. We select 2s as the prediction time window (i.e., predict viewpoint according to viewpoint traces in past 2s), since it achieves better performance than 3s, 4s and 5s based on our experiments. Note our selection of 2s is in line with the observation made by [68]. For training the model, we design a one-hot encoding representation [94, 95] for viewpoint as a 72×10 matrix V. Each element of V is 0 or 1. The

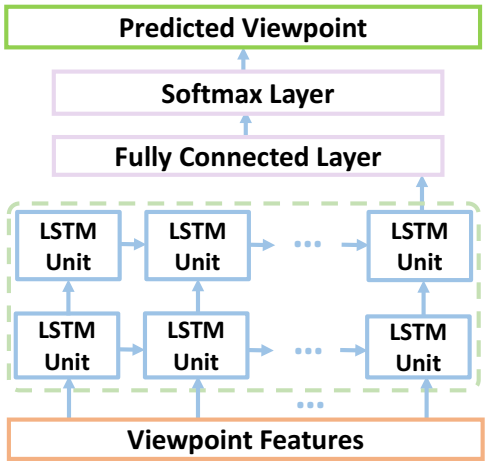


Figure 3.8: LSTM model used for viewpoint prediction.

dimensions of V correspond to the 72 tiles in a 360-degree view for possible viewpoint positions, and 10 timestamps corresponding to 2s. Thus, the element $v_{i,j}$ of matrix V equals to 1 when the viewpoint is within the *i-th* tile at the *j-th* timestamp, and equals to 0 when viewpoint is not within the corresponding tile. Another simple representation for viewpoint is a 1×10 vector, where each element equals to i when viewpoint is in the *i-th* tile. With the two representations above, we can obtain viewpoint features from previous and current viewpoint locations.

Inspired by the good performance of LSTM to capture transition regularities of human movements since they have memory to learn the temporal dependence between observations [78, 79], we design a multi-layer LSTM model which can learn general head motion patterns and predict the future viewpoint position based on the past traces. Fig. 3.8 shows the LSTM model we designed and used in our training, where first and second LSTM layers both consist of 128 LSTM units, and the fully connected layer contains 72 nodes. Our LSTM model predicts the next tile within which the viewpoint will be, given the previous sequence of viewpoint tiles. The outputs are the predicted probabilities over the 72 possible tiles. The proposed model learns parameters by minimizing cross-entropy and we train with mini-batches of size 30. Note that the settings including 128 LSTM units, 72 nodes and 30 as mini-batch size are selected during experiments and proved to be good by empirical results.

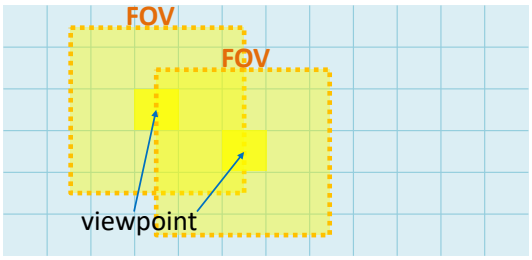


Figure 3.9: Example of generating predicted FOV.

We can use the viewpoint prediction probabilities of the tiles to generate an FOV, such that the probability that the actual user view in the next time point will be within the predicted FOV is maximized. In that case we will be able to ”predictively stream” the generated FOV in advance of the user's actual user view in the next time point, instead of the entire 360-degree/VR video, thus ensuring no additional latency at the next time point, while at the same time minimizing bandwidth consumption of the FOV transmitted (minimizing pixels/bitrate of FOV). We define FOV prediction accuracy as the probability that actual user view will be within the predicted FOV (generated from one or multiple tiles).

In our preliminary work [63], we select m tiles with highest probabilities predicted by the LSTM model, compose the predicted FOV as the combination of FOVs for each selected tile, and transmit the predicted FOV with high quality while leaving the rest of tiles blank. Fig. 3.9 shows an example of FOV generation when we select the top two highest probability tiles (i.e., $m = 2$) provided by the LSTM model, where yellow area illustrates the predicted FOV consisting of 26 tiles (i.e., the combination of FOVs for two selected tiles). In our current method, we build $120° \times 120°$ FOV around the center of the selected tile. By doing this, we can guarantee that when the viewpoint is within the predicted tile, the actual FOV is larger than $90° \times 90°$ in size (i.e., $90° \times 90°$ when the viewpoint is at the corner of predicted tile and $120° \times 120°$ when the viewpoint is in the center of predicted tile). We can use choice of m to achieve the desired trade-off between FOV prediction accuracy and bandwidth consumed in transmitting the predicted FOV.

As we show in Section 3.7.1, a very high FOV prediction accuracy can be obtained using

the above method, while also saving significant bandwidth compared to transmitting the entire 360-degree video. However, the above predictive streaming approach may not work, as the wireless network bandwidth available may not be always sufficient to transmit the predicted FOV with high accuracy (that is, considering high enough number of "m" tiles), thereby necessitating the predictive adaptive streaming approach we propose and describe in Section 3.6.

3.6 Predictive Adaptive Streaming Algorithm

While predictive streaming of FOV generated using viewpoint prediction in advance of the user actually looking at the FOV can address the ultra-low latency requirement of immersive 360-degree or VR experience, as mentioned earlier we also need adaptive streaming to address the challenge of fluctuating network bandwidth conditions. Hence in this section we propose a novel predictive adaptive streaming method. We investigate how viewpoint prediction can also be used to develop an effective adaptive streaming technique for 360-degree and VR videos, such that user experience can be maximized under dynamically varying network bandwidth conditions. Note that our predictive adaptive streaming algorithm will be executed every time slot (e.g., 200ms) using the network bandwidth at the beginning of the time slot. This way, our approach can address dynamically changing network bandwidth conditions.

We define FOV probability as probability of a given tile to belong to the user's actual FOV, where the actual user FOV refers to the actual user field of view in size of $90° \times 90°$. Our proposed algorithm aims to maximize user experience (i.e., maximize the quality of tiles with high FOV probability) by smartly selecting proper video encoding (quantization step) settings of the video for each tile. We can find an optimal solution for maximizing user experience (minimizing impairment I) given a bandwidth limit $BW(t)$ for time slot t. Finally, we give an analysis of the algorithm complexity. The notations used in our approach are described in Table 3.2.

Table 3.2: Notations used.

Notation	*Meaning*
$BW(t)$	network bandwidth limit for time slot t
$p_{fov}(k)$	predicted FOV probability for tile k
K	total number of tiles
qp	quantization parameter (QP)
L	number of levels in quantization parameter
q	quantization step
$I(VE)$	impairment caused by video encoding (VE)
q_k	optimal quantization step for each tile k
$MSE(q)$	mean square error (MSE) between encoded tile video data and corresponding raw video data for quantization step q
$R(q)$	bitrate for quantization step q
a, b, θ, γ	parameters in bitrate and MSE models
p_v	viewpoint probability
i, j, k	index parameters for tiles
q_{min}	minimum boundary of quantization step setting
R_{max}	bitrate when encoding with setting of q_{min}
MSE_{min}	MSE when encoding with setting of q_{min}

3.6.1 Problem Formulation

We formulate the problem as an optimization problem as follows. Since minimizing impairment I caused by video encoding (VE) equals to maximizing user experience, we set our optimization target as minimizing I, where I is defined as $I(VE) = \sum_{k=1}^{K} p_{fov}(k) MSE(q_k)$, $MSE(q)$ represents mean square error between the encoded tile video data and the corresponding raw video data for quantization step q, and q_k denotes the optimal quantization step for each tile k. And qp_1 as well as qp_L are the minimum and maximum boundaries of quantization step (QP) settings used for an application. This optimization problem aims to minimize impairment caused by video encoding under constraint of the bandwidth limit.

Given:

1) Network bandwidth limit $BW(t)$ for time slot t

2) Predicted FOV probability $p_{fov}(k)$ for each tile k, and total number of tiles K

3) Quantization parameter set $\{qp_1, qp_2, \ldots, qp_L\}$

4) Model parameters including a, b, θ, γ

Find:

The optimal quantization step q_k for each tile k to minimize impairment

$$I^{OPT} = \min_{q_k} I(VE) = \min_{q_k} \sum_{k=1}^{K} p_{fov}(k) MSE(q_k) \tag{3.2}$$

s.t.

$$\sum_{k=1}^{K} R(q_k) \leq BW(t) \tag{3.3}$$

$$q_k = (2^{(1/6)})^{QP_l - 4}, \quad k \in \{1, 2, \ldots, K\} \tag{3.4}$$

$$QP_l \in \{qp_1, qp_2, \ldots, qp_L\}, \quad l \in \{1, 2, \ldots, L\} \tag{3.5}$$

$$MSE(q_k) = a \cdot q_k + b \tag{3.6}$$

$$R(q_k) = \frac{\theta}{q_k^{\gamma}} \tag{3.7}$$

3.6.2 Viewpoint Probability and FOV Probability

From our predictive LSTM model described in Section 3.6, we can obtain the viewpoint probability for each tile. In this subsection, we study how to calculate the *FOV probability* based on *viewpoint probability*. Note that *FOV probability* p_{fov} is defined as the probability of a given tile to belong to the user's actual FOV, while *viewpoint probability* p_v is defined as whether the user viewpoint is within a given tile. Equation 3.8 describes the relationship between p_{fov} and p_v. We use Fig. 3.10, a frame in an example 360-degree video, as an instance to illustrate how to

Figure 3.10: Illustration for the calculation of FOV probability p_{fov} from viewpoint probability p_v.

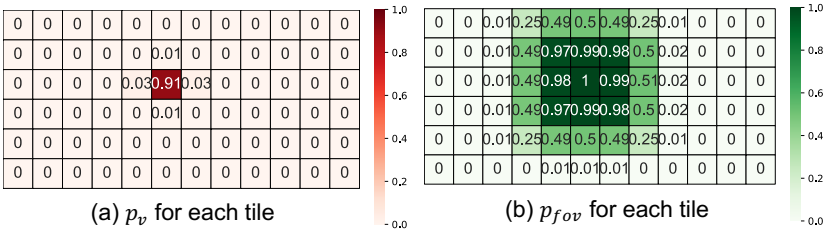


Figure 3.11: Instances (a) left, p_v for each tile (b) right, p_{fov} for each tile.

calculate p_{fov} from p_v.

$$p_{fov} = \sum_i p_v(i) + 0.5 * \sum_j p_v(j) + 0.25 * \sum_h p_v(h) \tag{3.8}$$

where p_{fov} and p_v refer to the *FOV probability* and *viewpoint probability* corresponding to a tile, and i, j, k are index parameters for tiles surrounding this given tile. For example, consider the red dashed tile (tile #1) in Fig. 3.10; in this case, tile i belongs to tiles #1-#9, tile j belongs to tiles #10-#21, and tile h belongs to tiles #22-#25. This equation describes that the probability of the red dashed tile (tile #1) will belong to the actual user FOV (including both totally- and partially-within cases) equals to the probability of user viewpoint is within the total colored area (including purple, orange and yellow area), since we assume that the user FOV is 3×3 tiles corresponding the size of 90°×90°. Specifically, the former refers to the left side of the equation, while on the right side, the first, second and third terms represents the probability of the viewpoint is within purple, orange and yellow area respectively. Note that the second and third terms assume that the viewpoint has the average probability distribution within the same tile. Fig. 3.11 presents an instance for the p_v for each tile and the corresponding p_{fov} for each tile using Equation 3.8.

Table 3.3: Experiment Setting.

Settings	***Experimental Values***
QP	15, 20, 25, 30, 33, 37, 40
Corresponding q	3.5, 6.5, 11, 20, 28, 44, 64
Resolution	4K (3840x2160) for two views, 3840x1080 for each view
GOP and Framerate	GOP: 12, framerate: 60fps

3.6.3 Bitrate and MSE Models

In this subsection, we study and model the relationship between video encoding setting used for the 360-degree/VR video (quantization step q) and the resulting bitrate (R) as well as mean square error (MSE), terms used in the problem formulation in Section 3.6.1.

1) Bitrate Model and Validation

Next, we introduce how we perform experiments to validate the model of relationship between the bitrate and quantization step q. Several techniques have been proposed to model the bitrate of the encoded video as a function of the video encoding parameters. [96] proposed models of bitrate R using quantization step q and video frame rate t. Since in this chapter, we do not consider the influence of different frame rate to bitrate, we fix the frame rate and thus we can simplify the model in [96] as follows.

$$R(q) = \frac{\theta}{q^{\gamma}} = R_{max}(\frac{q}{q_{min}})^{-\gamma} \tag{3.9}$$

In order to derive and validate this bitrate model, we encoded videos for 72 tiles respectively with different quantization parameter (QP) settings from Table 3.3. Using the H.265/HEVC standard definition that $q = (2^{(1/6)})^{QP-4}$ [96], the corresponding q values are 3.5, 6.5, 11, 20, 28, 44, and 64. For each video, we encode it by using x265 encoding library and record the bitrate under each q value. We set R_{max} to be the bitrate when encoding with q_{min} and calculate normalized bitrate $R(q)/R_{max}$.

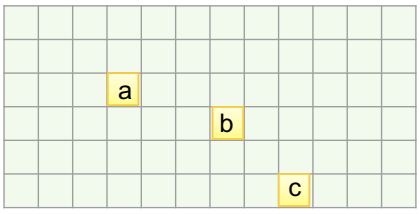


Figure 3.12: Tiles selected for model validation of bitrate and MSE.

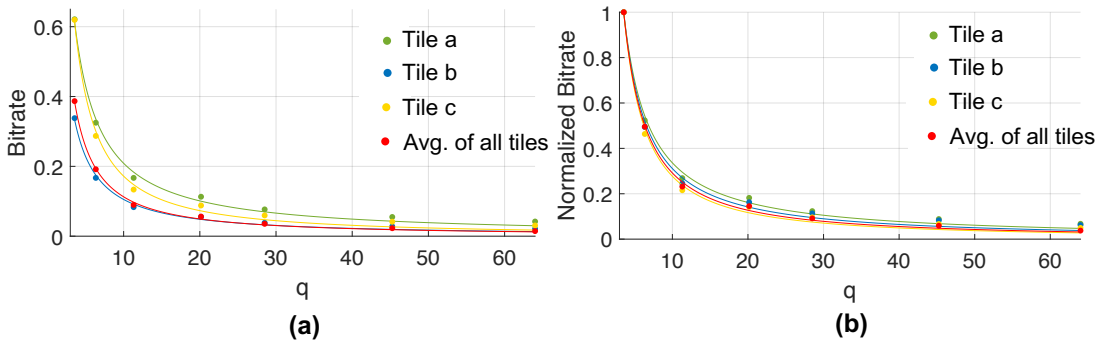


Figure 3.13: Validation of model equation for bitrate (a) left, bitrate versus q (b) right, normalized bitrate versus q.

We randomly pick several tiles shown in Fig. 3.12 as tiles a, b and c, and present results in Fig. 3.13(a)(b), where x-axis of the figures is q and y-axis are the bitrate and normalized bitrate in (a) and (b) respectively. The results of videos for each tile are represented by a specific color. Bitrates are shown as circles for the different tiles of videos and the average value of all 72 tiles, we also plot a line for each video tile to represent the model equation. The parameter γ is obtained by minimizing mean square error between the model predicted and measured bitrates for each video of tiles. Table 3.4 shows the corresponding parameter γ for the four lines in Fig. 3.13, and we can also calculate the parameter θ in Equation 3.9 accordingly. From Fig. 3.13, we can conclude that Equation 3.9 can model the bitrate of 360-degree video tiles using H.265/HEVC standard with high accuracy. Moreover, we do the validation of bitrate estimation for the tile. Fig. 3.15(a) shows the bitrate estimation results comparing the estimated bitrate for tile versus actual bitrate for tile using another 18 3s-tile video clips encoded with different QPs (i.e., QP = 15, 20, 30). The correlation is 0.9979 indicating the high accuracy of the proposed model (i.e., power law relationship between q and bitrate).

Table 3.4: Parameters for relationship between q and bitrate.

Tile	γ
Tile a	1.056
Tile b	1.131
Tile c	1.244
Avg. of all tiles	1.195

2) Model Validation for MSE

We investigate the relationship between the mean square error (MSE) and quantization step q, where the MSE refers to mean square error between the encoded tile video data and the corresponding raw video data for the tile. MSE reflects the average deviation of encoded tile pixels from their raw data counterparts. We adopt the linear model [97] between q and MSE, and investigate modeling how quantization step q influence encoding distortion MSE as follows.

$$MSE(q) = a \cdot q + b = MSE_{min}(a_1 \cdot \frac{q}{q_{min}} + b_1) \tag{3.10}$$

In order to validate the relationship between MSE and quantization step q for videos of different tiles, we follow the same encoding settings in Section 3.6.3 1), then calculate the MSE and pick three tiles in Fig. 3.12 for illustration. In Fig. 3.14, we use markers to show different data points and plot a line for each video tile to represent the fitted model equation. We set MSE_{min} to be the MSE when encoding with q_{min} and calculate normalized MSE as $MSE(q)/MSE_{min}$. The parameters a_1 and b_1 are also obtained by minimizing the mean square error between the model predicted and measured MSE for each video of tiles. Table 3.5 shows the parameters obtained in our experiments, and we can also calculate the parameters a and b in Equation 3.10 accordingly. From Fig. 3.14, we can see that Equation 3.10 can model the MSE of 360-degree video tiles using H.265/HEVC standard with high accuracy. In particular, for higher quantization step q, larger distortion (i.e., higher MSE) can be observed for tiles with more dynamic content (e.g., Tile c), while relatively smaller distortions are obtained for relatively static content (e.g., Tile b). Furthermore, we do the validation of MSE estimation for the tile. Fig. 3.15(b) shows the

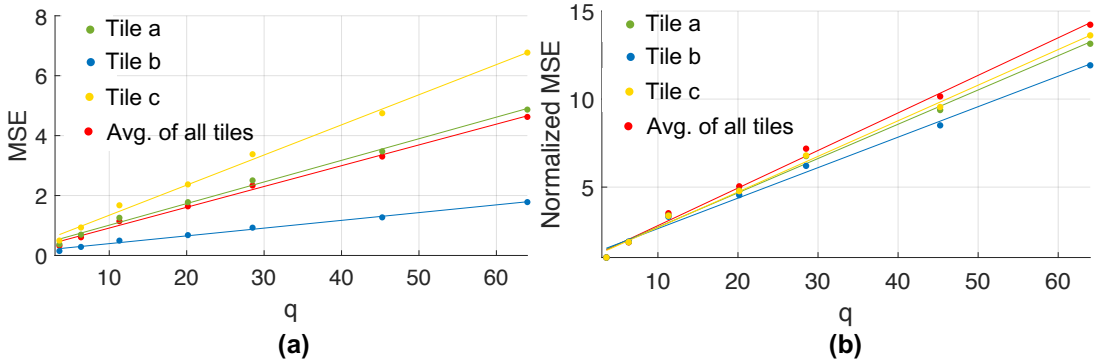


Figure 3.14: Validation of model equation for MSE (a) left, MSE versus q (b) right, normalized MSE versus q.

Table 3.5: Parameters for relationship between q and MSE.

Tile	a_1	b_1
Tile a	0.6939	0.7825
Tile b	0.6169	0.9127
Tile c	0.7207	0.6791
Avg. of all tiles	0.7603	0.6806

MSE estimation results comparing the estimated MSE for tile versus actual MSE for tile using another 18 3s-tile video clips encoded with different QPs (i.e., QP = 15, 20, 30). The correlation is 0.9923 indicating the high accuracy of the proposed model (i.e., linear relationship between q and MSE). Note that in our experiments, we employ the parameters obtained for average of all tiles to calculate the bitrate as well as MSE.

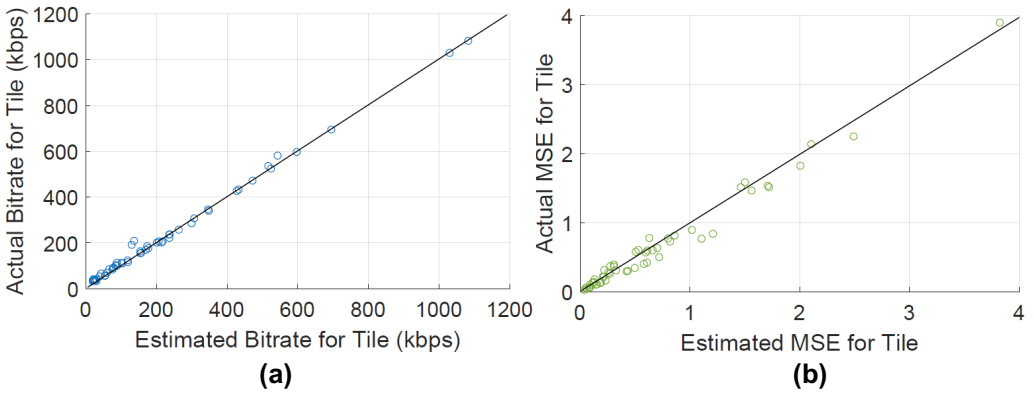


Figure 3.15: Validation of bitrate and MSE estimation for tile (a) left, results for bitrate (b) right, results for MSE.

3.6.4 Algorithm Description

We first describe the key ideas and insights of how we analyze the problem and develop the algorithm. Then we discuss the detailed steps of the algorithm.

First, notice that the problem we are to solve contains K discrete variables (i.e., one variable for each tile). If quantization step q has L levels, then we have L^K combinations for all q_k variables ($k \in \{1,2,\ldots,K\}$). In our experiment, we choose $L = 3$ resulting in 3^{72} combinations in total when the total number of tiles as $K = 72$. A brute-force algorithm based on exhaustive enumeration of the exponential number of combinations will be prohibitively expensive and cannot be applicable in real-time to perform adaptive streaming responding to real-time changes in network bandwidth. Hence, we propose a heuristic greedy algorithm to select the q level for each tile, such that impairment I is minimized (Eq. 2) subject to the bandwidth and other constraints (Eqs. 3-7).

- VR-PAS Algorithm

In order to solve the above problem, we first propose a heuristic greedy algorithm, VR Predictive Adaptive Streaming (*VR-PAS*) algorithm, which runs periodically (in this chapter we set the period to be 200ms). At the beginning of each time period, we will obtain the inputs of this algorithm: 1) network bandwidth limit $BW(t)$ for time slot t; 2) predicted FOV probability $p_{fov}(k)$ for each tile k, total number of tiles K, quantization parameter set $\{qp_1, qp_2, \ldots, qp_L\}$, and model parameters (a, b, θ, γ). We use $p_{fov}(k)$ and model parameters with Equation 3.7 to estimate the bitrate consumption in the next time period. The output of the algorithm will be the optimal quantization step q_k for each tile k.

This problem can be formulated as a variant of discrete *knapsack* problem [98], in which the bitrate consumed by each tile k can be interpreted as its *weight* (i.e., $w(k) = R(q_k)$); and the video encoding impairment caused by each tile k can be regarded as its *price* (i.e., $v(k) = p_{fov}(k)MSE(q_k)$). The problem can be restated as, for total K groups (K tiles), with each group having L objects (L quantization steps), select the optimal object (quantization step) for

Algorithm 3 VR-PAS Algorithm

Inputs:

1) Network bandwidth limit $BW(t)$ for time slot t
2) Predicted FOV probability $p_{fov}(k)$ for each tile k, and total number of tiles K
3) Quantization parameter set $\{qp_1, qp_2, \ldots, qp_L\}$
4) Model parameters including a, b, θ, γ

Output: The optimal quantization step q_k for each tile k to minimize impairment.

1: Initilization: for each tile k, set q_k to be q_{max} (i.e., $l(k) \leftarrow 1$, low quality), calculate the current bandwidth needed BW_{cur} and impairment I_{cur}
2: **while** $(BW_{cur} < BW(t))\&\&!(\text{all } q == q_{min})$ **do**
3: $BW_{per} \leftarrow BW_{cur}$; $I_{per} \leftarrow I_{cur}$
4: **for** $k = 1 : K$ **do**
5: $l(k) \leftarrow l(k)+1$; $q_k \leftarrow (2^{(1/6)})^{qp_{l(k)}-4}$
6: calculate BW_{cur} and I_{cur}
7: $\Delta BW_k \leftarrow BW_{cur} - BW_{per}$
8: $\Delta I_k \leftarrow I_{cur} - I_{per}$
9: $l(k) \leftarrow l(k)-1$; $q_k \leftarrow (2^{(1/6)})^{qp_{l(k)}-4}$
10: **end for**
11: Find k which has the maximum value of $\Delta I_k / \Delta BW_k$
12: Set the change of q_k of tile k as
$l(k) \leftarrow l(k)+1$; $q_k \leftarrow (2^{(1/6)})^{qp_{l(k)}-4}$
13: Calculate the new bandwidth needed BW_{cur} and I_{cur}, check if the new $BW_{cur} > BW(t)$ then revert q_k as previous value
14: **end while**
15: **return** q_k for each tile k

each group (tile), such that the total weight of these selected objects does not exceed the limit ($BW(t)$ for time slot t), and the total price (impairment) is minimized.

The underlying principle of the *VR-PAS* algorithm is as follows:

Algorithm 3 shows the pseudo-code of the *VR-PAS* algorithm. Initially we set the encoding quality of all tiles to be Low ($l = 1$, quantization step max). Then we keep adjusting the encoding quality of tiles using a while loop, as long as the total bitrate does not exceed the bitrate budget ($BW(t)$ for time slot t). During each iteration, the algorithm will first iterate over all the tiles, and for each tile k, the algorithm computes the possible degradation in its encoding impairment (ΔI_k), and the possible increase in the consumed bandwidth (ΔBW_k), if we set its encoding quality to be one level higher (corresponding to Line 5). Among all tiles, the algorithm will choose the one

with the highest ratio of $\Delta I_k/\Delta BW_k$. Note that the encoding impairment is calculated based on predicted FOV probability p_{fov} and MSE, thus a higher p_{fov} may lead to a higher $\Delta I_k/\Delta BW_k$ when parameters including MSE and bitrate R are fixed. Therefore, for a tile with higher p_{fov}, our algorithm will be more likely to choose this tile in the while loop and assign it with high quality. Algorithm will stop when 1) there is no more bandwidth available or 2) all tiles set encoding quality to be High. The proposed *VR-PAS* algorithm has a run time of less than 70ms with a Quad-core i7 processor, and thus can meet the real-time execution requirement of our predictive adaptive streaming method.

- VR-OPT Algorithm

To be able to compare the performance of our proposed real time heuristic algorithm *VR-PAS*, we next present a dynamic programming based algorithm, *VR-OPT*, which can produce an optimal solution for the adaptive streaming problem (Equation 3.2), though the latter has high runtime and hence cannot be used in practice. The core formula for the dynamic programming is as follow:

$$f[k][c] = min\{f[k-1][c-w[l]] + v[l] \mid object\ l \in group\ \#k\}$$

where $f[k][c]$ denotes the minimum price for the first k groups using cost c (condition: $0 < c \leq BW(t), 1 \leq k \leq K, 1 \leq l \leq L$), and different object l in group k corresponding to different encoding quality for a tile. The minimum $f[K][:]$ corresponds to the final solution. Note that this formula requires $BW(t)$ for time slot t and all $w[i]$ are integrals, assuming in our experiments the $w[i]$ can be as small as 0.06Mbps, then we magnify them 100 times simultaneously to make them become integral value. In this algorithm shown in Algorithm 4, Lines 3-20 achieve the core formula described above, which are solving the problems by breaking it apart into a sequence of smaller decisions. Specifically, we calculate the value of $f[k][c]$ for each combination of variables k and c (condition: $1 \leq k \leq K, 0 < c \leq BW(t)$), which means solving the problem of the minimum price for first k groups using cost c and obtaining the optimal solutions of all these smaller decisions.

Then by considering all these smaller decisions, the algorithm gets the minimum $f[K][:]$ as the final solution, corresponding to the minimum impairment achieved with a total bitrate within the bandwidth limit. Finally, in Line 21, Procedure *FindQuality* is a function tracing back the selected quantization step (i.e., selected quality) for each tile in the optimal solution.

3.6.5 Complexity Analysis

Since this problem is a variant of *knapsack problem*, it is also a NP-complete problem. Our proposed heuristic algorithm *VR-PAS* has the worst-case time complexity of $O(K \cdot K(L-1))$. The worst-case happens when the bandwidth limit equals to or larger than the bandwidth needed for highest quality for the whole 360-degree view. In this case, the iteration (i.e., Lines 2-14 of *VR-PAS*) will be conducted for $K(L-1)$ times, where one of the tiles is set one-level quality better during each iteration and finally all tiles are set with highest quality. As for the *VR-OPT* algorithm, the worst-case time complexity is $O(K \cdot 100BW(t) \cdot L)$ due to the three nested loops in Lines 3, 4, and 6 (whose number of iterations are K, $100BW(t)$, and L respectively). This algorithm can obtain optimal results but will be much more time-consuming than *VR-PAS*, which will be further discussed in the next section.

3.7 Experimental Results

In this section, we report on experiments conducted to evaluate the performance of our proposed predictive LSTM model, as well as our proposed predictive adaptive streaming algorithms. We also do an end-to-end timeline analysis of our predictive adaptive streaming approach.

Algorithm 4 VR-OPT Algorithm

Inputs:

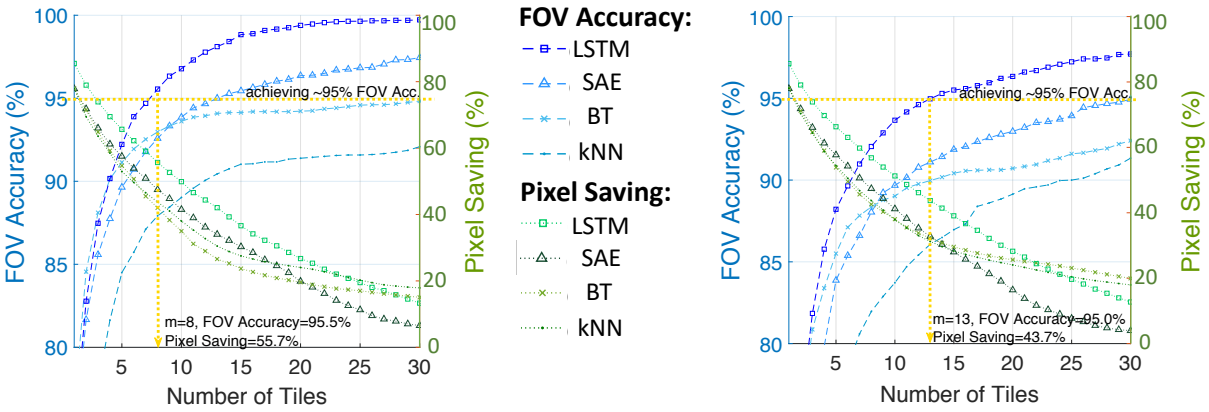
1) Network bandwidth limit $BW(t)$ for time slot t
2) Predicted FOV probability $p_{fov}(k)$ for each tile k, and total number of tiles K
3) Quantization parameter set $\{qp_1, qp_2, \ldots, qp_L\}$
4) Model parameters including a, b, θ, γ

Output: The optimal quantization step q_k for each tile k to minimize impairment.

$f \leftarrow zeros(K, BW(t) * 100)$
$mark \leftarrow zeros(K, BW(t) * 100)$
for $k = 1 : K$ **do**
  for $c = 0 : 0.01 : BW(t)$ **do**
    $temp \leftarrow INT\_MAX$
    for $l = 1 : L$ **do**
      $q \leftarrow (2^{(1/6)})^{qp_l - 4}$
      $w[l] \leftarrow R(q)$
      $v[l] \leftarrow p_{fov}(k) MSE(q)$
      if $c - w[l] \geq 0$ **then**
        $f_{temp} \leftarrow f[k-1][100 * (c - w[l])] + v[l]$
        if $f_{temp} < temp$ **then**
          $temp \leftarrow f_{temp}$
          $mark[k][c] \leftarrow l$
        end if
      end if
    end for
    $f[k][100 * c] \leftarrow temp$
  end for
end for
Get q_k from $FindQuality()$
return q_k for each tile k

Procedure $FindQuality()$:

$pre \leftarrow BW(t) * 100$
for $k = K : -1 : 1$ **do**
  $tile_l \leftarrow mark[k][pre]$
  Set the change of quantization step setting of tile k as $l(k) \leftarrow tile_l$; $q_k \leftarrow (2^{(1/6)})^{qp_{l(k)} - 4}$
  Denote bandwidth needed for the tile k as bw and calculate the new pre as $pre \leftarrow pre - bw * 100$
end for
return q_k for each tile k



(a) Medium Motion Sequence (b) High Motion Sequence

Figure 3.16: (a)(b) show FOV prediction accuracy and pixel saving versus number of tiles selected for FOV (for two sequences in *Kong VR*).

3.7.1 Predictive LSTM Model

Next we present the experimental results for our predictive LSTM model, including the experimental setup. We use 90% of the dataset for training the LSTM viewpoint prediction model, and 10% for testing, ensuring the test data is from viewers which are different than those in training data. Specifically, we have 32400 samples as training data and 3600 samples as test data for both medium motion and high motion sequences in Fig. 3.16 and Table 3.6, while we take 45000 samples as training data and 5000 samples as test data for each of three sequences in Table 3.7. As for the experimental setup, we use an Intel Core i7 Quad-Core processor with 32GB RAM and implement our approach in Python using Keras [99]. We compare the performance of our LSTM model with state-of-the-art methods as follows:

- *Stacked sparse autoencoders **(SAE):*** We use SAE [83, 100] with tile information during 10 timestamps as input to predict the tile where the viewpoint is within for next timestamp. The SAE model contains two fully-connected layers with 100 and 80 nodes respectively for training.

- *Bootstrap-aggregated decision trees **(BT):*** Following the work of [72], we also compare against BT using 10-timestamp tile information as input. The BT model ensembles with 30

bagged decision trees, which reduces the effects of overfitting and improves generalization.

- *Weighted k-nearest neighbors **(kNN):*** We implement a kNN [101] using 10-timestamp tile information as input and set 100 as the number of nearest neighbors.

Note that while the training time for BT and kNN are less than 20 minutes for the above training set for a 10-second sequence, the training time for the deep learning models including LSTM and SAE are up to one hour.

After training the various models with both two representations described in Section 3.5.1, we decide on using the one-hot encoding representation to train SAE and LSTM models, while using the simple representation for BT and kNN, since the simple representation works better for the latter two approaches in our experiments.

Table 3.6: Experimental results for two sequences in *Kong VR*.

Model	*Medium Motion Sequence*		*High Motion Sequence*	
	FOV Accuracy(%)	*Pixel Saving(%)*	*FOV Accuracy(%)*	*Pixel Saving(%)*
SAE	95.0	34.0	95.0	3.9
LSTM	95.5	**55.7**	95.0	**43.7**
BT	95.0	14.8	95.2	14.4
kNN	94.8	12.0	95.3	12.0

Table 3.7: Experimental results for three video sequences.

Model	*Fashion Show*		*Whale Encounter*		*Roller Coaster*	
	FOV Accuracy(%)	*Pixel Saving(%)*	*FOV Accuracy(%)*	*Pixel Saving(%)*	*FOV Accuracy(%)*	*Pixel Saving(%)*
SAE	95.4	52.7	95.1	46.8	95.3	29.9
LSTM	95.2	**69.7**	95.5	**66.8**	95.2	**71.0**
BT	95.3	19.1	95.0	18.6	95.2	48.9
kNN	94.9	12.0	95.2	10.3	95.1	21.2

Note in Fig. 3.16, Table 3.6 and Table 3.7, *FOV accuracy* refers to *FOV prediction accuracy*. We first show results of experiments with the medium and high motion sequences of *Kong VR* in Fig. 3.16 and Table 3.6. We show the FOV prediction accuracy and pixel savings obtained when selecting different number of tiles (i.e., the choice of m) to generate FOV. The blue plots show the FOV prediction accuracy achieved by each of the models for specific number of tiles (i.e., the choice of m) selected to generate the FOV, while the green plots show the

corresponding pixel saving of the generated FOV compared to the whole 360-degree view. Lines with blue triangle markers, blue square markers, blue cross markers and blue point markers represent FOV prediction accuracy for SAE, LSTM, BT and kNN models respectively while lines with green triangle markers, green square markers, green cross markers and green point markers represent corresponding pixel saving for these models.

From Fig. 3.16, we observe the following. As number of tiles m increases, the FOV prediction accuracy continuously increases and pixel saving simultaneously decreases. This shows the tradeoff between FOV prediction accuracy and pixel saving. Furthermore, we can see that our proposed LSTM model outperforms the other three methods. For instance, in both Fig. 3.16(a) and (b), the line with blue square markers (denoting FOV prediction accuracy achieved by LSTM) is significantly higher than the other three blue lines when the number of selected tiles (i.e., the choice of m) is larger than 5. We also observe that high FOV prediction accuracy can be achieved by LSTM (and other models) with smaller FOV and hence higher pixel savings for medium motion sequences compared to high motion sequences. For example, in Fig. 3.16(a), LSTM achieves a high FOV prediction accuracy of 95.5% when selects 8 tiles (i.e., $m = 8$) to generate FOV, leading to pixel savings of 55.7%, while in Fig. 3.16(b) to achieve a comparable FOV prediction accuracy of 95.0%, LSTM needs a larger FOV generated by 13 tiles (i.e., $m = 13$) with lower pixel savings of 43.7%. Table 3.6 summarizes the experimental results shown in Fig. 3.16. When we set FOV prediction accuracy as around 95%, we can observe that our LSTM model achieves significantly larger pixel savings than the other three models, achieving 55.7% and 43.7% pixel savings for medium and high motion sequences respectively. In our experiments, the inference time for all the models including LSTM is less than 2ms.

We further perform more experiments on three relatively low motion video sequences including *Fashion Show, Whale Encounter* and *Roller Coaster* in our dataset to evaluate our LSTM model. It corresponds to the fact that for instance in *Fashion Show* sequence, viewers have similar area of interest (e.g., the stage) and seldom change viewpoint out of this area to other tiles.

Table 3.8: QP and total bandwidth needed for different quality.

Quality	***QP***	***Total Bandwidth Needed***
Low	30	8.64Mbps
Medium	20	28.2Mbps
High	15	57.6Mbps

Similarly, in *Roller Coaster* sequence, viewers tend to look towards front more time than other directions when roller coaster keeps up high speed. Moreover, note that we select 10s-duration for each sequence to keep consistency with experiments done with *Kong VR*. The inference time for all the models including LSTM is still less than 2ms. Table 3.7 exhibits the experimental results for the three video sequences. Our LSTM model can achieve a very high FOV prediction accuracy of approximately 95% with selecting 4 tiles (i.e., $m = 4$) to generate FOV and corresponding pixel savings of around 70% for *Fashion Show* and *Roller Coaster*, and choosing 5 tiles (i.e., $m = 5$) to generate FOV and corresponding pixel savings of 66.8% for *Whale Encounter*. Note that the above savings are significantly higher than achieved by the other three models. Therefore, our experimental results above demonstrate that our LSTM model and FOV generation approach can achieve very high FOV prediction accuracy while significantly reducing pixels needed. In a separate work involving different VR applications, we have shown empirically that there is a high correlation between pixel and bitrate savings [102]. Thus, our experimental results also illustrate the tradeoff between FOV prediction accuracy and bandwidth savings.

While the above results demonstrate the accuracy and bandwidth savings potential of our viewpoint prediction and FOV generation approach, the network bandwidth available may not be sufficient to transmit the predicted FOV, thereby necessitating the predictive adaptive streaming approach we described in Section 3.6. We next describe experiments we conducted to evaluate our proposed predictive adaptive streaming algorithm *VR-PAS*, including comparison with the optimal algorithm *VR-OPT* as well as a recent related work on adaptive 360-degree video streaming.

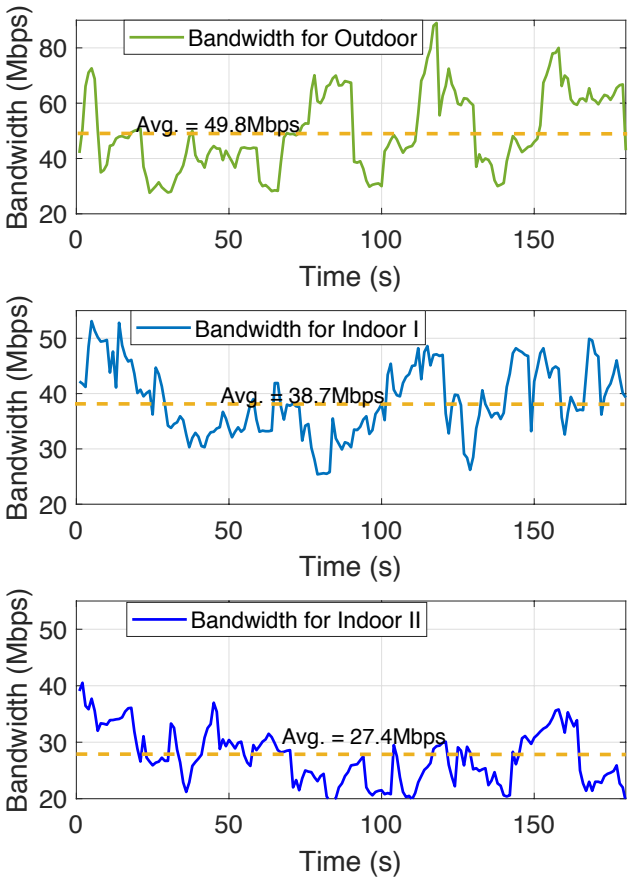


Figure 3.17: LTE bandwidth trace (a) top, bandwidth for outdoor (b) middle, bandwidth for indoor location I (c) bottom, bandwidth for indoor location II.

3.7.2 Predictive Adaptive Streaming

We first collect 4G-LTE network traces by using network bandwidth testing software Speedtest [103] to record the bandwidth. Fig. 3.17 shows the bandwidth measured during 180s in one outdoor location and two indoor locations respectively (for indoor environment, we select two indoor locations under different bandwidth conditions). We can see the average bandwidth for outdoor location, indoor location I, and indoor location II are 49.8Mbps, 38.7Mbps and 27.4Mbps respectively. The bandwidth in outdoor location is relatively largest among three locations while the bandwidth in indoor location II is more limited than in indoor location I. Table 3.8 presents the bandwidth needed for the video sequence called *Fashion Show* if all tiles are encoded in different quality (QP high, medium and low). We take it as an example to do our experiments and similar

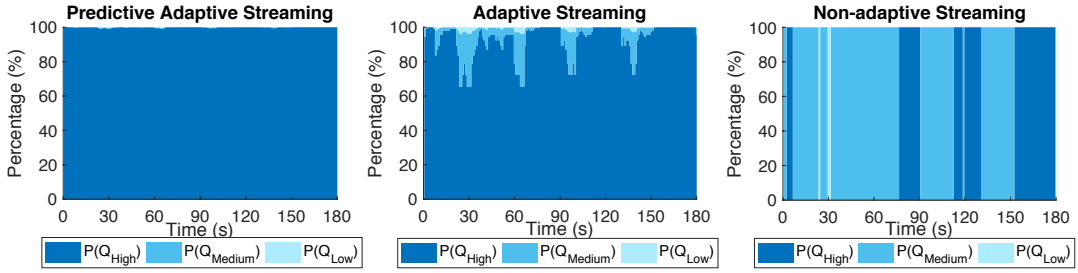


Figure 3.18: In outdoor location, average percentage of different quality view in actual FOV (a) left, predictive adaptive streaming (b) middle, adaptive streaming (c) right, no adaption streaming.

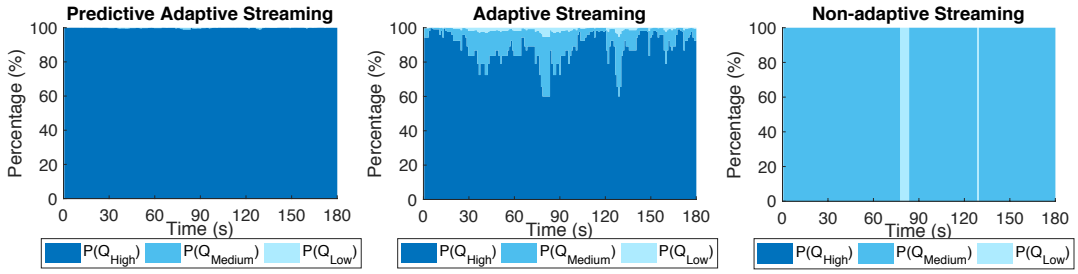


Figure 3.19: In indoor location I, average percentage of different quality view in actual FOV (a) left, predictive adaptive streaming (b) middle, adaptive streaming (c) right, no adaption streaming.

results can be obtained for other videos. From Table 3.8, we can see that all tiles can be encoded in high quality when bandwidth limit is larger than or equal to 57.6Mbps. In our experiments, for each time slot (e.g., 200ms), we consider the bandwidth value at the beginning of the time slot in the network traces in Fig. 3.17(a)(b)(c) as the ongoing bandwidth limit for the current time slot, and run our proposed algorithm with head motion traces.

In addition, for comparison reason, we also implemented two more algorithms as follows:

1) *Adaptive streaming*: We implement the *adaptive streaming* method, which is called viewport-driven rate-distortion optimized streaming from [89]. This method enables the adaptive video streaming according to the heatmaps that capture the likelihood of navigation of different spatial segments of a 360-degree video over time. Specifically, during one GOP (e.g., 1s), corresponding to 30 frames, for each tile k they count the number of times that the tile is occupied by user FOV, denoted as w_k. This technique computes the likelihoods

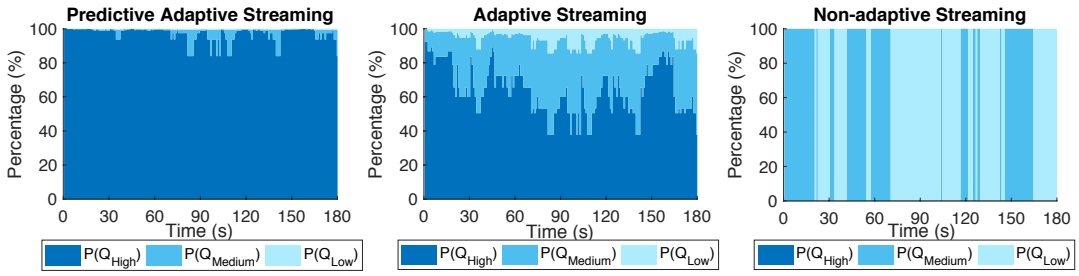


Figure 3.20: In indoor location II, average percentage of different quality view in actual FOV (a) left, predictive adaptive streaming (b) middle, adaptive streaming (c) right, no adaption streaming.

of navigating different tiles during the given GOP as $\frac{w_k}{\sum_k w_k}$ and then do the bitrate allocation (assign video quality) for tiles based on these likelihoods.

2) *Non-adaptive streaming*: We also compare against *non-adaptive streaming* method, which is the normal approach to stream the whole 360-degree view in the same quality (e.g., low/medium/high) according to the bandwidth condition.

Fig. 3.18, Fig. 3.19, and Fig. 3.20 show the results for the three different network bandwidth traces in outdoor location, indoor location I, and indoor location II respectively. We employ following metrics to evaluate the performance of our algorithm: $P(Q_{High})$ is defined as percentage of tiles in the actual user FOV which are encoded with High quality (i.e., low QP), while $P(Q_{Medium})$ and $P(Q_{Low})$ are defined as percentage of tiles in the actual user FOV which are encoded with Medium quality (i.e., medium QP) and Low quality (i.e., high QP) respectively. Using 1000 head motion traces for the video sequence called *Fashion Show*, we calculate the $P(Q_{High})$, $P(Q_{Medium})$, $P(Q_{Low})$ and PSNR (Equations 3.11 and 3.12) in actual user FOV under different bandwidth conditions. In Equations 3.11 and 3.12 [104], MSE_{avg} represents the average MSE of the actual user FOV while MSE_H, MSE_M, and MSE_L denotes the average MSE of high, medium, low quality tile respectively.

$$PSNR = 10log_{10}\frac{255^2}{MSE_{avg}} \tag{3.11}$$

$$MSE_{avg} = P(Q_{High})MSE_H + P(Q_{Medium})MSE_M + P(Q_{Low})MSE_L \quad (3.12)$$

Fig. 3.18(a), Fig. 3.19(a), and Fig. 3.20(a) plot the average $P(Q_{High})$ for all these traces while Fig. 3.18(b), Fig. 3.19(b), and Fig. 3.20(b) plot the average $P(Q_{Medium})$ and Fig. 3.18(c), Fig. 3.19(c), and Fig. 3.20(c) plot the average $P(Q_{Low})$. The corresponding PSNR in the actual FOV is also illustrated in Fig. 3.21 for these three locations. In each figure, we compare the performance of the three algorithms (our proposed *VR-PAS* as the predictive adaptive streaming method, adaptive streaming method [89] and non-adaptive streaming method). From the figures, we can make the following observations:

1) From Fig. 3.18 and Fig. 3.19, we can observe that our predictive adaptive streaming achieves high average $P(Q_{High})$ (i.e., larger than 99%) in outdoor location and indoor location I. In Fig. 3.20, our predictive adaptive streaming also achieves high average $P(Q_{High})$ (i.e., larger than 80%) in indoor location II, while both adaptive and non-adaptive streaming produces significantly lower user experience with $P(Q_{High})$ less than 65% and 0 respectively.

2) In all three locations, our predictive adaptive streaming algorithm *VR-PAS* performs significantly better (result in higher $P(Q_{High})$ and PSNR) compared to adaptive streaming [33] and non-adaptive streaming. For example, in Fig. 21(c), our predictive adaptive streaming achieves an average PSNR more than 1dB and 4dB larger compared to adaptive streaming and non-adaptive streaming respectively.

To evaluate the bandwidth savings and thereby wireless network capacity gain that can be achieved by *VR-PAS*, we measure $P(Q_{High})$, $P(Q_{Medium})$, $P(Q_{Low})$ and PSNR of streaming using *VR-PAS*, and compare with adaptive and non-adaptive streaming. Fig. 3.22 and Fig. 3.23

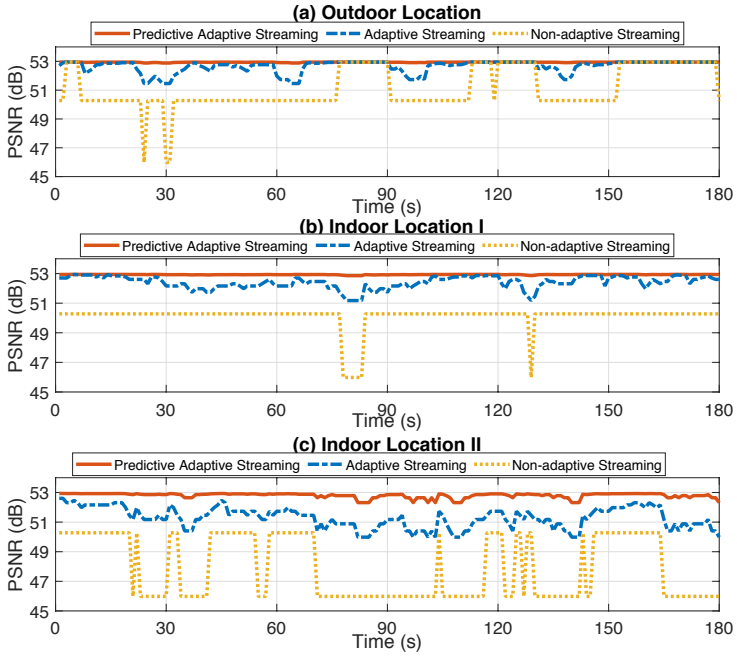


Figure 3.21: PSNR results (a) top, bandwidth for outdoor (b) middle, bandwidth for indoor location I (c) bottom, bandwidth for indoor location II.

illustrate the results for different bandwidth limits (x-axes) using three algorithms. Fig. 3.22 presents the $P(Q_{High})$, $P(Q_{Medium})$, $P(Q_{Low})$ under different bandwidth limits while Fig. 3.23 shows the PSNR of actual user FOV under different bandwidth limits. We provide a summary of key observations as follows:

1) Fig. 22 shows that our predictive adaptive streaming method has an average $P(Q_{High})$ of around 98% using 25Mbps while 57.6Mbps is the total bandwidth needed for the whole

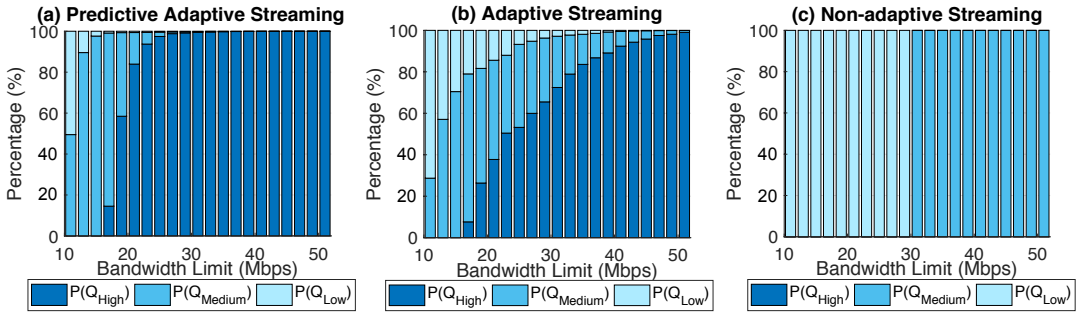


Figure 3.22: Percentages of different quality in actual user FOV under different fixed bandwidth limits (a) left, predictive adaptive streaming (b) middle, adaptive streaming (c) right, no adaption streaming.

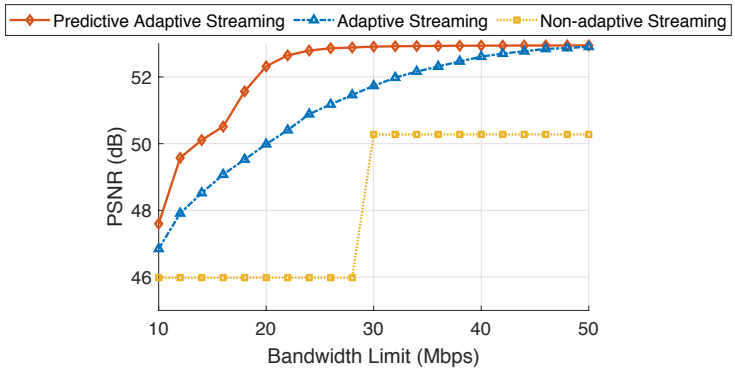


Figure 3.23: PSNR results versus the different fixed bandwidth limits.

360-degree view in High quality (Table 3.8), meaning that the user can have around 98% view in High quality with 56.6% bandwidth savings. This shows that *VR-PAS* can lead to significant bandwidth savings while ensuring very high user experience and satisfying ultra-low latency requirement due to accurate viewpoint prediction and advanced streaming.

2) In Fig. 23, we can observe that significantly lower bandwidth is needed by *VR-PAS* to produce the same high-quality experience compared to adaptive and non-adaptive streaming. For example, to achieve PSNR of 50dB in actual user FOV, from Fig. 23, *VR-PAS* requires around 14Mbps while the adaptive and non-adaptive streaming methods need around 20Mbps and 30Mbps respectively. Thus, our proposed predictive adaptive streaming method can provide more than 53.3% bandwidth saving compared to normal streaming (i.e., non-adaptive streaming) to offer the user FOV with PSNR of 50dB.

   The bandwidth savings achieved by the predictive adaptive streaming method can enable network operators to significantly improve capacity of their network, being able to serve significantly more users their 360-degree/VR video applications. For example, 2X more users can be served a high-quality experience of 50dB PSNR, compared to adaptive and non-adaptive techniques. This in turn can significantly improve the economics of delivering wirelessly 360-degree/VR experiences by service providers and/or network operators, while ensuring high user experience including visual quality and ultra-low latency requirements.

Next, we do a comparison for predictive adaptive streaming algorithms *VR-PAS* and *VR-OPT* in Table 3.9. Similar impairment $I(VE)$ is achieved by both algorithms while our *VR-PAS* algorithm can be completed within around 70ms and *VR-OPT* cannot be finished in real time (e.g., runtime is 8.3s approximately when bandwidth limit is 40Mbps). This shows that our *VR-PAS* algorithm is a real time algorithm and can generate similar results compared to the optimal results obtained by *VR-OPT* algorithm.

Table 3.9: Comparison for predictive adaptive streaming algorithms *VR-PAS* and *VR-OPT*.

Bandwidth Limit	***Algorithm***	***I(VE)***	***Runtime***
$10Mbps$	*VR-PAS*	21.807	13.9ms
	VR-OPT	21.807	2051ms
$20Mbps$	*VR-PAS*	7.929	27.8ms
	VR-OPT	7.877	4116ms
$30Mbps$	*VR-PAS*	5.501	39.4ms
	VR-OPT	5.496	6159ms
$40Mbps$	*VR-PAS*	5.224	51.3ms
	VR-OPT	5.223	8294ms
$50Mbps$	*VR-PAS*	5.186	63.4ms
	VR-OPT	5.186	10398ms

3.7.3 Timeline Analysis

In this subsection, we give an analysis of the timeline for our proposed predictive adaptive streaming method in Table 3.10. Specifically, we can see that the latency for transmission from HMD to edge and from edge to HMD depend on distance between them. Thus, our proposed predictive adaptive streaming algorithm *VR-PAS* can run in real-time, with less than 70ms runtime as shown in Table 3.9; with added round-trip transmission latency of under 25ms and the other delays due to viewpoint prediction and decoding (Table 3.10), our algorithm can be executed in real-time about every 100ms. Since we predict the user view 200ms in advance, we have adequate time to send the predicted view in advance and display the needed view for users on the HMD with no additional latency, hence satisfying the ultra-low latency requirement of 360-degree video and VR immersive experiences. Note that Table 3.10 does not include the time of encoding tiles since

Table 3.10: Time needed for different procedures.

Procedure	***Time Needed***
Transmission from HMD to edge	*Depends on distance*
Viewpoint Prediction	$< 2ms$
Predictive Adaptive Streaming Alg.	$< 70ms$
Transmission from edge to HMD	*Depends on distance*
Decoding	$\approx 3ms$

video tiles are known and have been encoded before streaming in the current 360-degree video streaming scenario; the time consumption of encoding tiles will have to be added if real-time tile encoding is needed.

3.8 Conclusion

In this chapter, we propose a predictive adaptive streaming approach in order to reduce the latency and bandwidth needed to deliver 360-degree videos and cloud/edge-based VR applications, leading to better mobile VR experiences. We present a multi-layer LSTM model which can learn general head motion pattern and predict the future viewpoint based on past traces. Our method outperforms state-of-the-art methods on a real head motion trace dataset and shows great potential to reduce bandwidth while keeping a good user experience (i.e., high PSNR).

3.9 Acknowledgments

Chapter 3 contains the reprints of Xueshi Hou, Sujit Dey, Jianzhong Zhang and Madhukar Budagavi, "Predictive View Generation to Enable Mobile 360-degree and VR Experiences," *Proc. of the 2018 Morning Workshop on Virtual Reality and Augmented Reality Network*, 2018; and Xueshi Hou, Sujit Dey, Jianzhong Zhang and Madhukar Budagavi, "Predictive Adaptive Streaming to Enable Mobile 360-degree and VR Experiences," *IEEE Transactions on Multimedia*, 2020. The dissertation author is the primary investigator and author of each of these papers.

Chapter 4

Motion Prediction and Pre-Rendering at the Edge to Enable Ultra-Low Latency Mobile 6DoF Experiences

This chapter presents the methodology that can be used for mobile VR application streaming. As virtual reality (VR) applications become popular, the desire to enable high-quality, lightweight, and mobile VR can potentially be achieved by performing the VR rendering and encoding computations at the edge and streaming the rendered video to the VR glasses. However, if the rendering has to be performed after the edge gets to know of the user's new head and body position, the ultra-low latency requirements of VR will not be met by the roundtrip delay. In this chapter, we introduce edge intelligence, wherein the edge can predict, pre-render and cache the VR video in advance, to be streamed to the user VR glasses as soon as needed. The edge-based predictive pre-rendering approach can address the challenging six Degrees of Freedom (6DoF) VR content. Compared to 360-degree videos and 3DoF (head motion only) VR, 6DoF VR supports both head and body motion, thus not only viewing direction but also viewing position can change. Hence, our proposed VR edge intelligence comprises of predicting both the head

and body motions of a user accurately using past head and body motion traces. In this chapter, we develop a multi-task long short-term memory (LSTM) model for body motion prediction and a multi-layer perceptron (MLP) model for head motion prediction. We implement the deep learning-based motion prediction models and validate their accuracy and effectiveness using a dataset of over 840,000 samples for head and body motion.

4.1 Introduction

Virtual reality (VR) systems have triggered enormous interest over the last few years in various fields including entertainment, enterprise, education, manufacturing, transportation, etc. However, several key hurdles need to be overcome for businesses and consumers to get fully on board with VR technology [1]: cheaper price and compelling content, and, most importantly, a truly mobile VR experience. Of particular interest is how to develop mobile (wireless and lightweight) head-mounted displays (HMDs), and how to enable VR experience on the mobile HMDs using bandwidth-constrained mobile networks, while satisfying the ultra-low latency requirements.

Currently, there are several categories of HMDs [2]: PC VR, standalone VR, and mobile VR. Specifically, PC VR has high visual quality with rich graphics contents as well as high frame rate, but the HMD is usually tethered with PC [3,4]; standalone VR HMD has a built-in processor and is mobile, but may have relative low-quality graphics and low refresh rate [5,6]; mobile VR is with a smartphone inside, leading to a heavy HMD to wear [7,8]. Therefore, current HMDs still cannot offer us a lightweight, mobile, and high-quality VR experience. To solve this problem, we propose an edge computing based solution. By performing the rendering on an edge computing node and streaming videos to users, we can complete the heavy computational tasks on the edge computing node and thus enable mobile VR with lightweight VR glasses. The most challenging part of this solution is ultra-high bandwidth and ultra-low latency requirements, since streaming 360-degree video causes tremendous bandwidth consumption and good VR user experiences

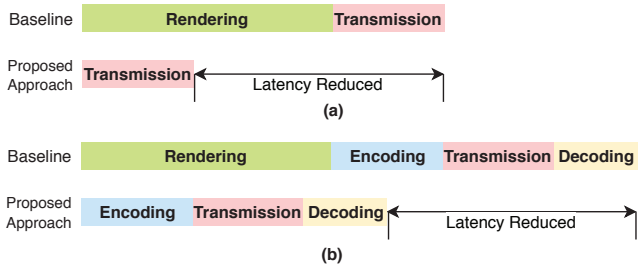


Figure 4.1: Illustration of rendering and streaming pipeline to show how our *predictive pre-rendering* approach reduces latency: (a) Without encoding and decoding; (b) With encoding and decoding.

require ultra-low latency (<20ms) [9, 10].

Specifically, the total end-to-end latency of edge computing based VR system includes the following parts: time to transmit sensor data from HMD to edge computing node, time to render (and encode) on the edge node, time to transmit rendered video from the edge computing node to HMD, and time to (decode and) display the view on the HMD. The encoding and decoding are optional according to the specific application design. Once the user moves his/her head or body position, high-quality VR requires this end-to-end latency as less than 20ms [9, 10] to avoid motion sickness. For the edge computing based VR system, it is extremely challenging to meet this requirement.

Motivated by the ultra-low latency requirement challenge, in this chapter, we introduce edge intelligence for mobile VR, wherein the edge can predict, pre-render and cache the VR video in advance, to be streamed to the user VR glasses as soon as needed. Specifically, we consider six Degrees of Freedom (6DoF) VR experiences, which support both the head and body motions, thus both the viewing direction and viewing position can change. Hence, in order to pre-render the view, edge intelligence is needed to predict both the head and body motions of a user accurately. By predicting head and body motion of users in the near future with edge intelligence, we can do a predictive pre-rendering on the edge computing node and then stream (even pre-deliver) the predicted view to the HMD. The difference between *stream* and *pre-deliver* is that *stream* means holding the pre-rendered frame until determining whether prediction is 'correct' or not using

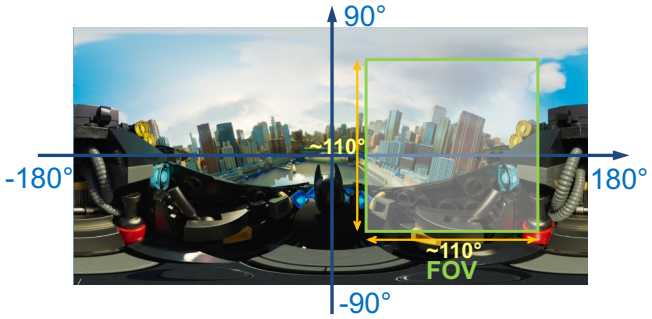


Figure 4.2: Field of view (FOV) in a 360-degree view.

the actual motion, while *pre-deliver* refers to sending the pre-rendered frame immediately to the user without this determination. Note that both *stream* and *pre-deliver* choices can significantly reduce latency: one does pre-rendering and the other does both pre-rendering and pre-delivery. The latter reduces more latency than the former but (i) needs a technique on HMD to buffer the predicted view and determine whether the predicted viewing position and direction are correct; (ii) transmits extra content when the prediction is inaccurate, leading to more bandwidth consumption. Hence, we adopt the former method, where the latency can be significantly reduced since the pre-rendered view will be transmitted if the predicted viewing position and direction are 'correct' (i.e., the error is less than a given ultra-low value); otherwise, latency remains the same with traditional streaming method because the actual view will be rendered and transmitted to the HMD. Fig. 4.1 illustrates the latency reduced by our pre-rendering approach compared to the traditional approach, in terms of rendering and streaming pipeline (from edge computing node to HMD). The key to achieving this efficient edge-based predictive pre-rendering approach is predicting body and head motion in advance accurately, and then pre-rendering the predicted view accordingly.

In our earlier work [63], we proposed techniques for head motion prediction in 360-degree videos and three Degrees of Freedom (3DoF) VR applications. In this work, we address the more challenging 6DoF VR content. Compared to 360-degree videos and 3DoF (head motion only) VR, 6DoF VR supports both head and body motions, thus not only viewing direction but also viewing position changes. Hence, our proposed VR edge intelligence has to comprise of

predicting both the head and body motions of a user accurately using past head and body motion traces. Specifically, for head motion prediction in 360-degree videos and 3DoF VR, a certain prediction error is allowed, because the error can be handled by delivering a larger field of view (FOV) with high quality or rendering larger FOV. Note that FOV is around $90^\circ \times 90^\circ$ for Samsung Gear VR and $110^\circ \times 110^\circ$ for HTC Vive while the 360-degree view is $360^\circ \times 180^\circ$ in size (as is shown in Fig. 4.2). Compared to 360-degree videos and 3DoF VR, the motion prediction in 6DoF VR is much more challenging, where the body motion prediction needs high precision to pre-render the user's view (otherwise may cause dizzy feeling). For 360-degree videos and 3DoF VR, the 360-degree view at a time point is known and unchanged by any head motion, but for 6DoF VR it can be totally different due to the body motion. Therefore, this chapter will explore the feasibility of doing motion prediction with high precision in 6DoF VR using edge intelligence, and its main contributions can be summarized as follows:

- For 6DoF VR applications, we propose a new *edge-based predictive pre-rendering* approach involving both body and head motion prediction, in order to enable high-quality, lightweight, and mobile VR with low latency.

- We develop a prediction method using edge intelligence to predict where a user will be standing (i.e., viewing position) and looking into (i.e., viewing direction) in the 360-degree view based on their past behavior. Using a dataset of real head and body motion traces from VR applications, we show the feasibility of our multi-task long short-term memory (LSTM) model for body motion prediction and multi-layer perceptron (MLP) model for head motion prediction with high precision.

- We propose a FOV selection technique for pre-rendering a larger FOV to further reduce head motion prediction error, and a motion error determination technique as the system mechanism of our edge-based predictive pre-rendering approach.

- To the best of our knowledge, we are the first to come up with this edge-based predictive

pre-rendering idea using edge intelligence for 6DoF VR applications and show good results on a real motion trace dataset in the VR applications. We demonstrate the potential of our approach with high accuracy of head and body motion prediction.

Note that a preliminary version of our work has been published in [105], where we reported on edge-based predictive single-task models for head (MLP model) and body (LSTM model) motions, and some preliminary results. In this article, we develop (i) a new multi-task LSTM model for body motion prediction to reduce body motion prediction error, (ii) head and body motion prediction based FOV selection for pre-rendering, such that the selected FOV minimizes the effects of motion prediction error while also minimizing the selected FOV size, and (iii) motion error determination as the system mechanism of our edge-based predictive pre-rendering approach. Note that the methodology proposed in this chapter applies to single-user scenarios, and we plan to further study more complex multi-user scenarios as part of future work.

The rest of the chapter is organized as follows. Section 4.2 reviews related work. Section 4.3 presents a system overview and problem definition. Section 4.4 describes our dataset. The methodology for head and body motion prediction is described in Section 4.5. We present our experimental results in Section 4.6 and conclude our work in Section 4.7.

4.2 Related Work

In this section, we review current work in the following topics related to our research.

Enable High-Quality Mobile VR: Some recent studies [106–110] explore solutions to enable lightweight and mobile VR experiences, and improve the performance of the current VR system. To provide high-quality VR on a mobile device, [106] presents a pre-rendering and caching design called FlashBack, which pre-renders all possible views for different positions as well as orientations at each 3D grid point with a density of 2-5cm, stores them on a local cache, and delivers frames on demand according to current position and orientations. This method may lead to high inaccuracy and overwhelming storage overhead of pre-caching all possible views

(e.g., 50GB for an app). [107] introduces a parallel rendering and streaming mechanism to reduce the add-on streaming latency, by pipelining the rendering, encoding, transmission, and decoding procedures. This method focuses on minimizing streaming latency, thus the latency for rendering part remains the same as the traditional rendering method. [108] presents a collaborative rendering method to reduce overall rendering latency by offloading costly background rendering to an edge computing node and only performing foreground rendering on the mobile device. In contrast, our method proposes to pre-render based on head and body motion predictions, reducing the latency of rendering more drastically. To reduce latency needed, [109] proposes to stream VR scenes containing only the user's FOV and a latency-adaptive margin area around the FOV. [110] aims to address the ultra-high bandwidth challenge in high-quality mobile VR by adaptively reusing the redundant VR pixels across multiple VR frames. The reason these two methods cannot be applied to our scenario is that [109] cannot address 6DoF VR content and [110] reduces network transmission latency to some extent but also brings the larger rendering latency.

Human Motion Prediction: Learning statistical models of human motion are challenging due to the stochastic nature of human movement to explore the environment, and many works [78, 111–114] propose methods to address it. Based on classical mechanics, there are some studies [78, 111, 112] showing the efficiency of linear acceleration model (Lin-A) by doing motion prediction or estimation with an assumption of linear acceleration, especially in a small time interval (e.g., order of tens of milliseconds). [111] describes a good performance of a simple first-order linear motion model for tracking human limb segment orientation, and [78, 112] reveal acceptable results when employing the linear model as a baseline to predict human trajectory. Meanwhile, deep learning approaches [78, 112–114] for human body prediction have also achieved remarkable accomplishments. Specifically, [78, 112] propose their LSTM models to predict human future trajectories, but their models aim to learn general human movement from a massive number of videos and the corresponding precision of predicted position does not achieve the requirement of pre-rendering in VR scenarios. [113, 114] propose various recurrent neural network (RNN)

models for human motion prediction to learn human kinematics from skeletal data. But these models are designed to learn the patterns from a series of skeletal data and cannot be applied to our VR scenarios directly.

Moreover, [63, 84, 85, 115] also explore the feasibility of doing head motion prediction, however, head motion prediction in 6DoF is quite different than 360-degree video (3DoF), since in the latter, for each time point, the whole 360-degree view displayed for viewers is fixed and more regularity and pattern exist in their viewing directions. By learning viewers' traces, for 3DoF applications, the models can well predict the viewing position since at a certain time point, there are always some areas attracting most attention and viewers are more likely to look at them. Head motion in 6DoF is more difficult to predict because both position and viewing direction may continuously change, and there is a much larger virtual space to explore for users. Therefore, the above approaches cannot be used to address our scenario: we aim to explore the high-precision human body and head motion prediction in 6DoF VR applications for pre-rendering.

Multi-task Learning: Multi-task learning aims to improve learning efficiency and prediction accuracy for each task, compared to training a separate model for each task. Some recent studies [116–118] explore solutions to improve prediction accuracy by learning multiple tasks from a shared representation, and formulate the multi-task learning problems which involve joint learning of various regression and classification tasks with different units and scales. [116] shows that a shared representation with multi-task learning can improve accuracy on depth regression and instance segmentation over separately trained single tasks because of cues from other tasks. [117] presents that multi-task learning benefits and achieves better results compared with single-task models on event detection in social media by doing text analysis with Twitter datasets. [118] proposes a multi-task RNN for simultaneous recognition of surgical gestures with kinematic signals, and demonstrates that the recognition performance improves with the multi-task learning model compared with single-task models. The reason why we cannot use above methods for body motion is that most of these methods [116, 118] address computer vision

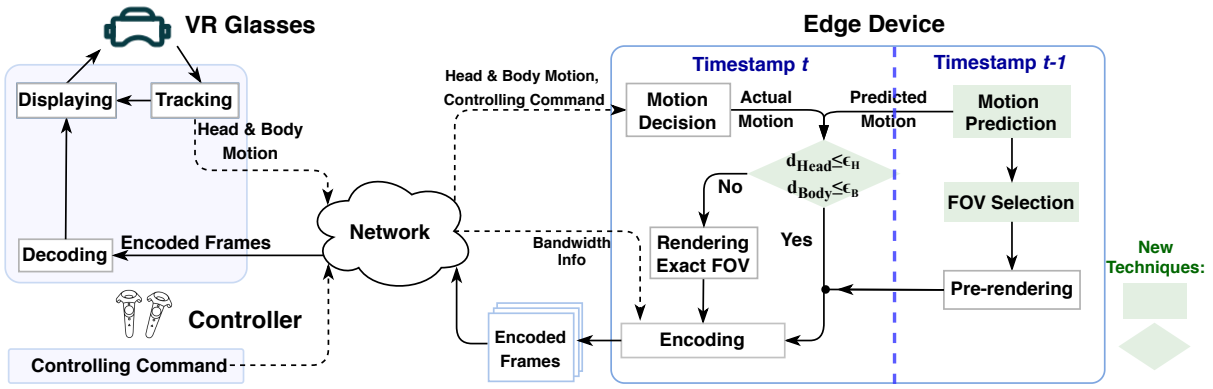


Figure 4.3: System overview.

recognition problem instead of predicting variables ahead of time and [117] considers event detection based on texts in social media which also cannot be applied to body motion prediction scenario. Our proposed multi-task model distinguishes from the above methods by addressing the real-time body motion prediction problem using real motion traces in the VR scenario and aiming for an ultra-low prediction error.

4.3 System Overview

In this section, we describe our system overview. In Fig. 4.3, a user's head motion, body motion as well as other controlling commands will firstly be sent to the edge, which performs the *edge-based predictive pre-rendering* approach. Based on the past few seconds of head motion, body motion and control data received from the user, the edge device will do three things: (i) perform motion prediction (*motion prediction*); (ii) do pre-render based on the predicted viewing position and direction (*motion decision* and *pre-rendering*); (iii) cache the predicted frames in advance. Later, if the predicted viewing position and direction are 'correct' (i.e., the error is less than a given ultra-low value), the cached predicted frames can be streamed from the edge device to the HMD and displayed on HMD immediately; otherwise, the actual view will be rendered by the edge device and transmitted to the HMD. For the former case, latency needed will be significantly reduced since the view is pre-rendered and cached on the edge computing

node before it is needed; for the latter, latency remains the same with the conventional method of streaming from the edge computing node. Note that although the controller can affect the rendered frame by pointing at a certain place to teleport in virtual space, we do not need to predict for the new location triggered by the controller, as in this case, users will expect much larger latency than 20ms. We will describe motion prediction, FOV selection, and motion error determination (highlighted in green in Fig. 4.3) with more details in Section 4.5.

Note that the edge device can be either a Mobile Edge Computing node (MEC) in the mobile radio access or core network, or a Local Edge Computing node (LEC) located in the user premises or even his/her mobile device, connecting to the HMD through WiFi or WiGig. While each of the above choices has tradeoffs, this chapter will not specifically address these tradeoffs and select either MEC or LEC. Instead, we focus on developing accurate head and body motion prediction techniques, which can be used for the edge-based predictive pre-rendering approach shown in Fig. 4.3, and will apply to either of the edge device options.

Problem Statement: In each time point, the user can have a specific viewing position and viewing direction, corresponding to the body and head motion. Given previous and current viewing directions and viewing positions, our goal is to predict viewing direction and position for the next time point. After rendering pixels based on predicted viewing position and direction, frames can be further encoded to a video and delivered to users. Specifically, we describe the problem formulation for motion prediction below. The notations used in our approach are described in Table 4.1.

4.3.1 Problem Formulation

Trajectory Sequence: Spatiotemporal point q_t is a tuple of time stamp t, viewing position b, and viewing direction h, *i.e.*, $q_t = (t, b, h)$. The trajectory sequence from time point t_w to time point t_{w+n-1} is a spatiotemporal point sequence, which can be denoted as $S(t_w, t_{t+n-1}) = q_{t_w} q_{t_{w+1}} \ldots q_{t_{w+n-1}}$.

Table 4.1: Notations used.

Notation	Meaning
t	Timestamp (time counted since application launches)
RTT	Round-trip latency
(α, β, γ)	Euler angles for head pose (pitch α, yaw β, roll γ)
(x, y, z)	Position for body pose
$\vec{v}_{head}$	Head motion speed $(v_\alpha, v_\beta, v_\gamma)$
$\vec{v}_{body}$	Body motion speed (v_x, v_y, v_z)
d_{head}	Angular distance between actual and predicted head poses
d_{body}	Distance between actual and predicted body positions
$d_\alpha, d_\beta, d_\gamma$	d_{head} in α, β, γ-axis
d_x, d_y, d_z	d_{body} in x, y, z-axis
$\varepsilon_1, \varepsilon_2$	Thresholds of acceptable head and body prediction errors
L_i	Objective loss function for individual task i
w_i	Weight for individual task i
L_{total}	Loss function for multi-task learning model
θ_h, θ_v	Horizontal FOV and vertical FOV
θ'_h, θ'_v	Selected new horizontal FOV and vertical FOV
n_w	Number of frames in a sliding window in FOV selection
$\hat{d}_\alpha$, $\hat{d}_\beta$, $\hat{d}_\gamma$	Estimated value of $d_\alpha, d_\beta, d_\gamma$ in FOV selection
I_1, I_2	Two grayscale intensity images
$I_{dif}(i)$	Difference between two intensity images for pixel i
R_{dif}	Percentage of mismatched pixels
N_{dif}	Number of pixels having difference in grayscale intensity
N_{frame}	Total number of pixels per frame

Thus, the problem can be formulated as follows:

- Input: a trajectory sequence from time point t_w to time point t_{w+n-1}, *i.e.*, $S(t_w, t_{t+n-1}) = q_{t_w} q_{t_{w+1}} \dots q_{t_{w+n-1}}$;

- Output: predicted spatiotemporal point $\widehat{q_{t_{w+n}}}$ at time point t_{w+n};

In this chapter, we aim to predict the viewing position b and viewing direction h for the next time point using current and previous viewing positions and directions.

4.3.2 Time Analysis

In this subsection, we give an analysis of the time taken for the various tasks of our proposed edge-based predictive pre-rendering method, as shown in Table 4.2. Specifically, we can see that the latency for transmission from HMD to the edge and from edge to HMD depends on the distance between them. Since we predict the user view 11ms in advance (1 frame ahead, assuming 90 frames/second), we have adequate time to (i) predict motion and do FOV selection (i.e., $< 1ms$, which is described in details in Section 4.6.4) and (ii) pre-render the predicted view (i.e., $5ms - 10ms$) in advance with no additional latency, hence satisfying the ultra-low latency requirement of 6DoF VR immersive experiences. The round-trip transmission latency, latency of rendering, latency of encoding, and latency of decoding can be denoted as RTT, $T_{rendering}$, $T_{encoding}$, and $T_{decoding}$ respectively.

As for the conventional method, the latency without motion prediction and pre-rendering is

$$RTT + T_{rendering} + T_{encoding} + T_{decoding},$$

where the lower boundary and upper boundary of latency are $RTT + 11ms$ and $RTT + 21ms$ respectively. Thus, given added round-trip transmission latency of around 9ms, the end-to-end latency for conventional method is $20ms - 30ms$.

For our proposed edge-based predictive pre-rendering approach, the latency with 'correct' motion prediction is

$$RTT + T_{encoding} + T_{decoding},$$

where the lower boundary and upper boundary of latency are $RTT + 6ms$ and $RTT + 11ms$ respectively. Otherwise, when the motion prediction is not 'correct', the latency is the same with conventional method. Thus, given added round-trip transmission latency of around 9ms, the end-to-end latency for the proposed edge-based predictive pre-rendering approach is $15ms - 20ms$ with 'correct' motion prediction and $20ms - 30ms$ with 'incorrect' motion prediction. We present

experimental results in Section 4.6 which shows high accuracy of our proposed motion prediction techniques, achieving 'correct' motion predictions in most of the time points during 6DoF VR applications.

Table 4.2: Time needed for different procedures.

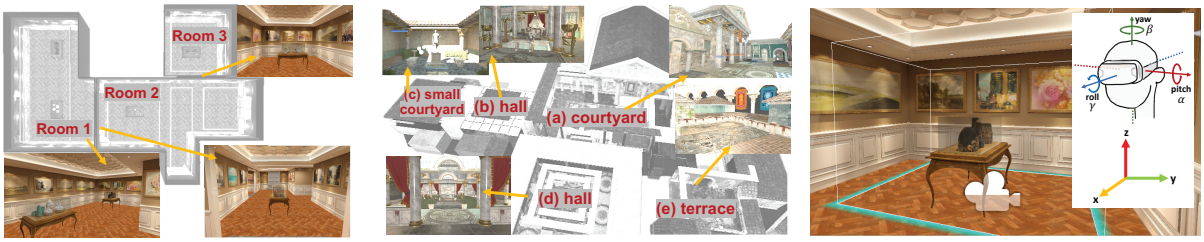
Procedure	***Time Needed***
Transmission from HMD to edge	*Depends on distance*
Rendering	$5ms - 10ms$
Encoding	$3ms - 8ms$
Transmission from edge to HMD	*Depends on distance*
Decoding	$\approx 3ms$
Motion Prediction & FOV Selection	$< 1ms$

4.4 Dataset and Its Characteristics

In this section, we first describe the dataset we use and then show characteristics of the dataset using certain metrics we define.

4.4.1 Dataset

To investigate head and body prediction in 6DoF VR applications, we conduct our study on a real motion trace dataset we collected from 20 users using HTC Vive to experience two 6DoF VR applications called Virtual Museum [119] and Virtual Rome [120] in our laboratory. The system setup will be described in Section 4.6.1. The trace consists of 840,000 sample points of head and body motion data collected from the users. Fig. 4.4(a)(b) show the illustration of the two virtual applications, where Virtual Museum has three exhibition rooms and Virtual Rome contains larger space including different courtyards and halls. The *walkable area* is restricted by the size of the tracked space in the room and constrained to a fixed regular shape. Users can explore each virtual space by walking in the *walkable area* or teleporting by pointing at a place with a controller. The top subplot in Fig. 4.4(c) uses light blue lines to show the boundary of the *walkable area* in the VR. As shown in Table 4.3, we set three sessions respectively for each



(a) (b) (c)

Figure 4.4: Illustration of two virtual applications and other settings: (a) Virtual Museum and (b) Virtual Rome; (c) Boundary of *walkable area*, and coordinates for head and body motions.

Table 4.3: Experimental settings for different sessions in the Virtual Museum and Virtual Rome.

Session	***Virtual Museum (VM)***			***Virtual Rome (RM)***		
	VM1	*VM2*	*VM3*	*RM1*	*RM2*	*RM3*
With Guidance	✔			✔		
Use Controller			✔			✔

application: (i) in session 1, users are given rough guidance of taking a stroll about the room at the beginning of the session, without a controller in their hand; (ii) in session 2, users walk around freely in the room, without a controller in their hand; (iii) in session 3, users walk around freely in the room and have a controller in their hand; the controller allows them to teleport to any position in virtual space by pointing at that place, and the position of the *walkable area* in VR also changes accordingly.

Motion traces include the user ID, session timestamp, euler angles for the head pose (pitch α, yaw β, roll γ), and position for body pose (x, y, z). The session timestamp refers to the time counted since application launches in milliseconds, and timestamps appear each 11ms (corresponding to 90Hz, which is the refresh rate of HTC Vive). The middle and bottom subplots of Fig. 4.4(c) exhibit the coordinates for head pose using euler angles and for body pose using position.

4.4.2 Dataset Characteristics

To depict key characteristics of the head motion and viewpoint changes in the dataset quantitatively, we offer the following definitions.

Head Motion Vector: The corresponding head poses at time points t_1 and t_2 (where $t_1 < t_2$) are denoted by $(\alpha(t_1), \beta(t_1), \gamma(t_1))$ and $(\alpha(t_2), \beta(t_2), \gamma(t_2))$ respectively. Head motion vector $(\triangle\alpha, \triangle\beta, \triangle\gamma) = (\alpha(t_2) - \alpha(t_1), \beta(t_2) - \beta(t_1), \gamma(t_2) - \gamma(t_1))$.

Head Motion Speed: Head motion speed $\vec{v}_{head}$ is defined as the angular distance the head moved divided by time, i.e., $\vec{v}_{head} = (v_\alpha, v_\beta, v_\gamma) = (\frac{\triangle\alpha}{t_2 - t_1}, \frac{\triangle\beta}{t_2 - t_1}, \frac{\triangle\gamma}{t_2 - t_1})$.

Body Motion Vector: The corresponding body poses at time points t_1 and t_2 (where $t_1 < t_2$) are denoted by $(x(t_1), y(t_1), z(t_1))$ and $(x(t_2), y(t_2), z(t_2))$ respectively. Body motion vector $(\triangle x, \triangle y, \triangle z) = (x(t_2) - x(t_1), y(t_2) - y(t_1), z(t_2) - z(t_1))$.

Body Motion Speed: Body motion speed $\vec{v}_{body}$ is defined as the distance the body moved divided by time, i.e., $\vec{v}_{body} = (v_x, v_y, v_z) = (\frac{\triangle x}{t_2 - t_1}, \frac{\triangle y}{t_2 - t_1}, \frac{\triangle z}{t_2 - t_1})$, and the value of it is

$$v_{body} = |\vec{v}_{body}| = \sqrt{{v_x}^2 + {v_y}^2 + {v_z}^2}. \tag{4.1}$$

Table 4.4: Description of variables.

	Variable	Seq.	Unit
Measured	Timestamp	✔	Millisecond (ms)
	Euler angles	✔	Degree ($^\circ$)
	Position	✔	Meter (m)
Derived	Head Motion Speed	✔	Degree per Millisecond ($^\circ/ms$)
	Body Motion Speed	✔	Centimeter per Millisecond (cm/ms)

Table 4.4 presents the description of variables. Apart from measured variables in the dataset, for each sample point, we can obtain the derived variables including head motion speed and body motion speed using definitions above. In Fig. 4.5, we plot the cumulative distribution function (CDF) of body motion speed in each axis (i.e., v_x, v_y, v_z) and head motion speed in each axis (i.e., $v_\alpha, v_\beta, v_\gamma$) for different sessions. We can see that (i) over 95% of v_x, v_y, v_z are less than 0.8m/s, 0.8m/s, and 0.15m/s respectively, and around 90% of $v_\alpha, v_\beta, v_\gamma$ are less than 30°/s, 100°/s, and 25°/s respectively; (ii) for the body motion speed distribution, the speed in

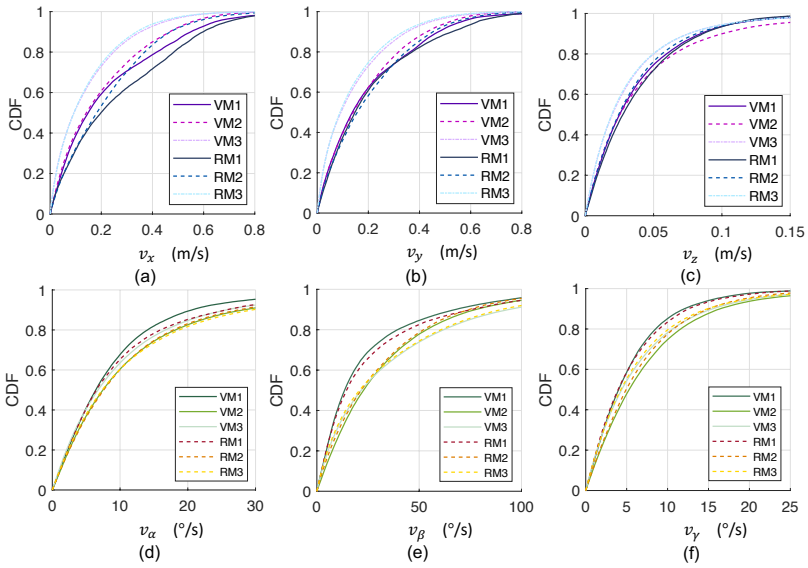


Figure 4.5: CDF of motion speed for different sessions: (a)(b)(c) for body motion; (d)(e)(f) for head motion.

each session is as follow from high to low: RM1>VM1>RM2>VM2>VM3>RM3; and (iii) for the head motion speed distribution, the speed in each session is as follow from low to high: VM1<RM1<RM2&VM2<RM3&VM3. Thus among six sessions of two applications, there are more body motion and less head motion in Session 1 (i.e., RM1, VM1) while less body motion and more head motion in Session 3 (i.e., RM3, VM3).

4.5 Our Approach

In this section, we describe our proposed approach of preprocessing and modeling for head and body motion predictions.

4.5.1 Preprocessing

We aim to remove noise within head and body motion in the preprocessing step. We first calculate head motion speed and body motion speed for each time point. Fig. 4.6 presents the body motion and head motion speed in $x, y, z, \alpha, \beta, \gamma$-axis respectively for a sample in the motion trace of one user in the Virtual Museum application. The blue line in each subplot shows

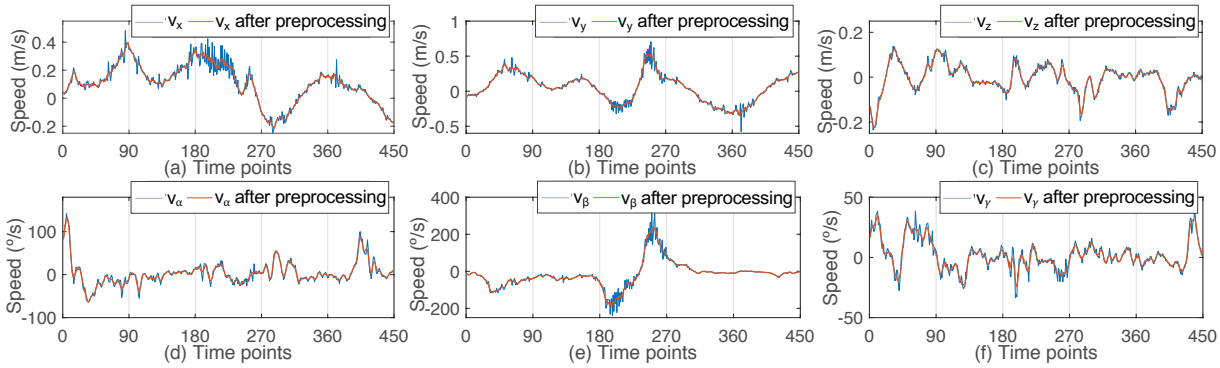


Figure 4.6: Motion speed obtained before and after the preprocessing step: (a)(b)(c) for body motion; (d)(e)(f) for head motion.

there can be at times significant noise in each of motion speed, due to sensor noise and other measuring errors from HTC Vive HMD and base stations. This noise is identifiable since the speed cannot change so rapidly and intensively within several milliseconds. To remove the noise in body motion and head motion, we propose to use the Savitzky-Golay filter method [121] because of its high accuracy and efficiency. This filter approximates (using least-square fitting) the underlying function within the moving window by a polynomial of a higher order. The blue and red lines in Fig. 4.6 show the speed before and after the preprocessing step. We can see the noise is significantly reduced after preprocessing step.

4.5.2 Predictive Modeling

To represent motion features, we select 60 time points as the prediction time window (i.e., predict head and body speed according to speed traces in the latest 60 time points), since it achieves better performance than 40, 50, 70, 80, 90 time points based on our experiments. For training the model, we choose a simple representation for motion as a 1×60 vector, where each element equals to i when the speed is i at that time point, and the dimension of 60 corresponds to 60 time points.

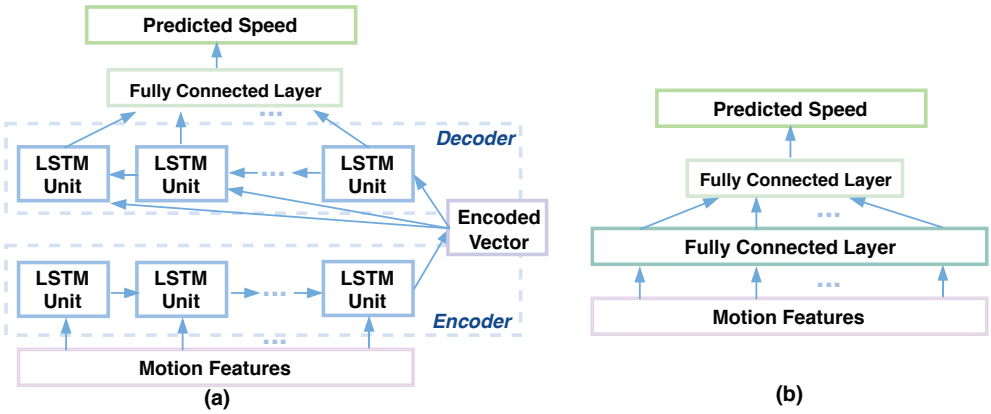


Figure 4.7: (a) LSTM model and (b) MLP model used for motion prediction.

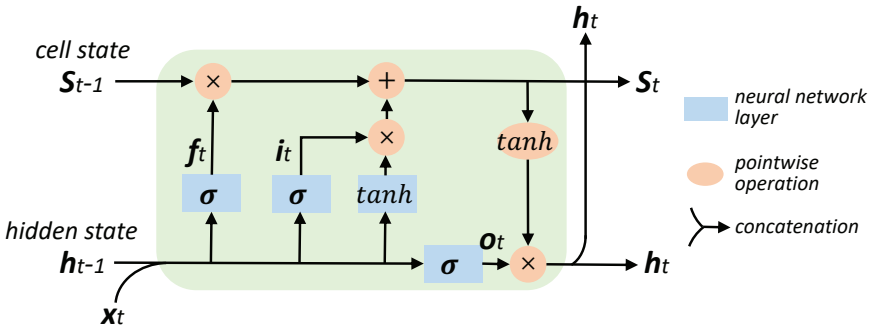


Figure 4.8: The structure of LSTM unit.

1) Single-task Model

We investigate a LSTM model as well as an MLP model to be trained for single task separately, where the single task refers to prediction for body motion speed in each axis (i.e., x, y, or z-axis) or head motion speed in each axis (i.e., α, β, or γ-axis).

LSTM Model: Inspired by the success of the RNN Encoder-Decoder in modeling sequential data [122] and good performance of LSTM to capture transition regularities of human movements since they have memory to learn the temporal dependence between observations [78, 79], we implement an Encoder-Decoder LSTM model which can learn general body motion as well as head motion patterns, and predict the future viewing direction and position based on the past traces. Fig. 4.7(a) shows the LSTM model we designed and used in our training, where first

and second LSTM layers both consist of 60 LSTM units, and the fully connected layer contains 1 interconnected node. Note the interconnected node refers to the general neuron-like processing unit $a = \phi(\sum_j w_j x_j + b)$, where x_j are the inputs to the unit, w_j are the weights, b is the bias, ϕ is the nonlinear activation function, and a is the unit's activation [123].

Our Encoder-Decoder LSTM model predicts what the motion speed will be for next time point, given the previous sequence of motion speed. The outputs are the values of predicted speed for next time point. Note that the settings including 60 LSTM units and 60 time points as window length are selected during experiments and proved to be good by empirical results. For the head and body motion prediction, we use the mean square error (MSE) as our loss function:

$$Loss = \frac{1}{|N_{train}|} \sum_{y \in S_{train}} \sum_{t=1}^{L} (y_t - \hat{y}_t)^2, \tag{4.2}$$

where $|N_{train}|$ is the number of total time steps of all trajectories on the train set S_{train}, and L is the total length of each corresponding trajectories. The proposed LSTM model learns parameters by minimizing the mean square error.

Specifically, encoder and decoder sections work as follows. Given the input sequence $\boldsymbol{X} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_t}, \ldots, \boldsymbol{x_T})$ with $\boldsymbol{x_t} \in \mathbb{R}^n$, where n is the number of driving series (e.g., dimension of feature representation), the encoder learns a mapping from $\boldsymbol{x_t}$ to $\boldsymbol{h_t}$ with

$$\boldsymbol{h_t} = f(\boldsymbol{h}_{t-1}, \boldsymbol{x_t}), \tag{4.3}$$

where $\boldsymbol{h_t} \in \mathbb{R}^m$ is the hidden state of the encoder at time t, m is the size of the hidden state, and f is a non-linear activation function of LSTM unit. As shown in Fig. 4.8, each LSTM unit has (i) a memory cell with the cell state $\boldsymbol{s_t}$, and (ii) three sigmoid gates to control the access to memory cell (forget gate $\boldsymbol{f_t}$, input gate $\boldsymbol{i_t}$ and output gate $\boldsymbol{o_t}$). We follow the LSTM structure

from [122, 124]:

$$\boldsymbol{f_t} = \sigma(\boldsymbol{W_f}[\boldsymbol{h}_{t-1}; \boldsymbol{x_t}] + \boldsymbol{b_f}), \tag{4.4}$$

$$\boldsymbol{i_t} = \sigma(\boldsymbol{W_i}[\boldsymbol{h}_{t-1}; \boldsymbol{x_t}] + \boldsymbol{b_i}), \tag{4.5}$$

$$\boldsymbol{o_t} = \sigma(\boldsymbol{W_o}[\boldsymbol{h}_{t-1}; \boldsymbol{x_t}] + \boldsymbol{b_o}), \tag{4.6}$$

$$\boldsymbol{s_t} = \boldsymbol{f_t} \odot \boldsymbol{s}_{t-1} + \boldsymbol{i_t} \odot (tanh(\boldsymbol{W_s}[\boldsymbol{h}_{t-1}; \boldsymbol{x_t}] + \boldsymbol{b_s})), \tag{4.7}$$

$$\boldsymbol{h_t} = \boldsymbol{o_t} \odot tanh(\boldsymbol{s_t}), \tag{4.8}$$

where $[\boldsymbol{h}_{t-1}; \boldsymbol{x_t}] \in \mathbb{R}^{m+n}$ is a concatenation of the previous hidden state $\boldsymbol{h}_{t-1}$ and current input $\boldsymbol{x_t}$. $\boldsymbol{W_f}$, $\boldsymbol{W_i}$, $\boldsymbol{W_o}$, $\boldsymbol{W_s} \in \mathbb{R}^{m \times (m+n)}$ as well as $\boldsymbol{b_f}$, $\boldsymbol{b_i}$, $\boldsymbol{b_o}$, $\boldsymbol{b_s} \in \mathbb{R}^m$ are parameters to learn. Notations of σ and $\odot$ are the logistic sigmoid function and element-wise multiplication. After reading the end of input sequence sequentially and updating the hidden state as above, the hidden state of LSTM is a summary (i.e., encoded vector $\boldsymbol{c}$) of the whole input sequence. Subsequently, the decoder is trained to generate the target sequence $(y_1, \ldots, y_t, \ldots, y_T)$ by predicting y_t given hidden state $\boldsymbol{d_t}$ of LSTM units in decoder at timestep t. Note that $y_t \in \mathbb{R}$, and $\boldsymbol{d_t} \in \mathbb{R}^p$, where p is the size of the hidden state in decoder. The update of hidden state is denoted by

$$\boldsymbol{d_t} = f(\boldsymbol{d}_{t-1}, y_{t-1}, \boldsymbol{c}). \tag{4.9}$$

Since the nonlinear function is the LSTM unit function, similarly, $\boldsymbol{d_t}$ can be updated as:

$$\boldsymbol{f'_t} = \sigma(\boldsymbol{W'_f}[\boldsymbol{d}_{t-1}; y_{t-1}; \boldsymbol{c}] + \boldsymbol{b'_f}), \tag{4.10}$$

$$\boldsymbol{i'_t} = \sigma(\boldsymbol{W'_i}[\boldsymbol{d}_{t-1}; y_{t-1}; \boldsymbol{c}] + \boldsymbol{b'_i}), \tag{4.11}$$

$$\boldsymbol{o'_t} = \sigma(\boldsymbol{W'_o}[\boldsymbol{d}_{t-1}; y_{t-1}; \boldsymbol{c}] + \boldsymbol{b'_o}), \tag{4.12}$$

$$\boldsymbol{s'_t} = \boldsymbol{f'_t} \odot \boldsymbol{s'}_{t-1} + \boldsymbol{i'_t} \odot (tanh(\boldsymbol{W'_s}[\boldsymbol{d}_{t-1}; y_{t-1}; \boldsymbol{c}] + \boldsymbol{b'_s})), \tag{4.13}$$

$$\boldsymbol{d_t} = \boldsymbol{o'_t} \odot tanh(\boldsymbol{s'_t}), \tag{4.14}$$

where $[\boldsymbol{d}_{t-1}; y_{t-1}; \boldsymbol{c}] \in \mathbb{R}^{p+m+1}$ is a concatenation of the previous hidden state $\boldsymbol{d}_{t-1}$, decoder input

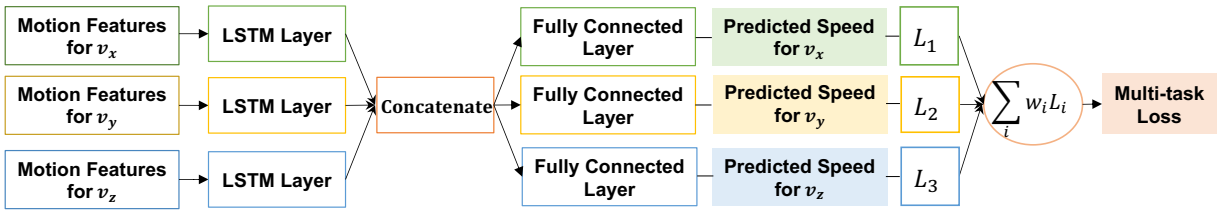


Figure 4.9: Multi-task LSTM model for body motion prediction.

y_{t-1}, and encoded vector $\boldsymbol{c}$. $\boldsymbol{W'_f}, \boldsymbol{W'_i}, \boldsymbol{W'_o}, \boldsymbol{W'_s} \in \mathbb{R}^{p\times(p+m+1)}$ as well as $\boldsymbol{b'_f}, \boldsymbol{b'_i}, \boldsymbol{b'_o}, \boldsymbol{b'_s} \in \mathbb{R}^{p}$ are parameters to learn. Subsequently, the output of the decoder is further fed to the fully connected layer.

MLP Model: Apart from the LSTM model, we propose to use an MLP [123] model presented in Fig. 4.7(b) to do motion prediction. Using the same representation and loss function described above, this model takes the motion speed during the latest 60 time points as input to predict the motion speed for next time point. The MLP model contains two fully-connected layers with 60 and 1 interconnected nodes respectively for training. The MLP model also learns parameters by minimizing the mean square error.

We build up single-task models including *LSTM* and *MLP* models for body motion and head motion speed in $x, y, z, \alpha, \beta, \gamma$-axis respectively. Given the current and previous speed traces, our predictive models can predict the speed for next time point and thus predict the viewing position b and viewing direction h for next time point (described in Section 4.3.1).

2) Multi-task Model

Motivated by achieving better body motion prediction to reduce the potential adverse effect on user experience caused by prediction error, we explore more models to predict body motion more accurately. Since single-task models in Section 4.5.2 1) predict body motion speed of each axis separately (using the information in one axis), we explore multi-task models to take advantage of body motion speed in all three axes to predict the body motion for each axis. We investigate a multi-task LSTM model as well as a multi-task MLP model, sharing some layers to

determine common features between multiple tasks, where each task refers to the prediction for body motion speed in each axis (i.e., x, y, or z-axis).

Multi-task LSTM Model: We implement a multi-task LSTM model that can learn a shared representation for body motion pattern, and predict the body motion speed (corresponding to viewing position) for the next time point based on the past traces. Fig. 4.9 shows our proposed multi-task LSTM model that we have designed and used for training, where the first three LSTM layers after the three motion features layers consist of 60, 60, and 60 LSTM units (Fig. 4.8) respectively, and the three fully-connected layers after a concatenate layer contain 1, 1, and 1 interconnected node. Our multi-task LSTM model predicts what the body motion speed in x, y and z-axis will be for the next time point, given the previous sequence of the body motion speed. The outputs are the values of predicted speed (i.e., v_x, v_y, v_z) for the next time point. Note that the settings including 60 LSTM units and 60 time points as window length are selected during experiments and proved to be good by empirical results. For the body motion prediction, we define the multi-task loss function as the weighted linear sum of the losses for each individual task:

$$L_{total} = \sum_i w_i L_i, \tag{4.15}$$

where w_i is the weight for individual task i and L_i is the single task loss function for individual task i (defined as the MSE, which is described before in the LSTM model subsection 4.5.2 1). Specifically, as shown in Fig. 4.9, tasks 1, 2, and 3 refer to the prediction for body motion speed in x, y, and z-axis respectively. In our training, we use $w_1 = w_2 = w_3 = 0.333$ as the task weight setting based on good empirical performance and following theoretical observation. Specifically, for body motion prediction, the distance between actual body motion and predicted motion can be defined as

$$d = \sqrt{{d_x}^2 + {d_y}^2 + {d_z}^2}, \tag{4.16}$$

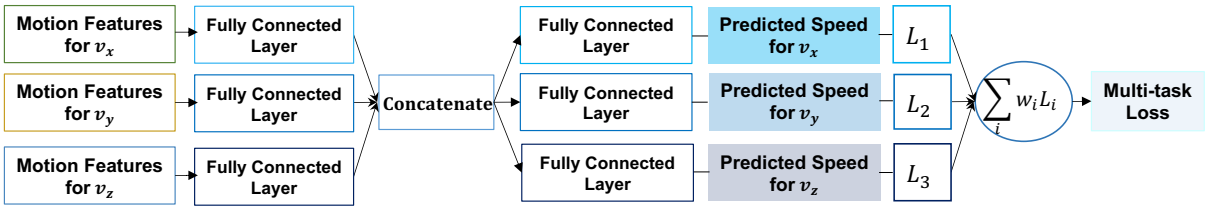


Figure 4.10: Multi-task MLP model for body motion prediction.

where d_x, d_y, d_z are the distance between actual body motion and predicted body motion in the x, y, z-axis respectively. Thus, the theoretical observation is that with the setting of $w_1 = w_2 = w_3$, minimizing the multi-task learning loss function for body motion prediction is equivalent to minimizing the square of distance d between the actual viewing position (obtained from body motion) and predicted viewing position. Note that the proposed multi-task LSTM and MLP models learn parameters by minimizing the multi-task loss function. Note that we have considered and conducted experiments to models of sharing LSTM layer and fully-connected layer between x, y, and z, but their performances are worse than the performance of our proposed model (Fig. 4.9).

Multi-task MLP Model: Apart from the multi-task LSTM model, we also implement a multi-task MLP model for comparison to do body motion prediction. Using the same representation and multi-task loss function described above, this model also takes the body motion speed during the latest 60 time points in x, y, z-axis as input to predict the body motion speed in the next time point. Our proposed multi-task MLP model has a similar structure like the multi-task LSTM model, shown in Fig. 4.10, where the first three fully-connected layers after the three motion feature layers consist of 60, 60, and 60 interconnected nodes respectively, and the three fully-connected layers after a concatenate layer contain 1, 1, and 1 interconnected node. The multi-task MLP model predicts what the body motion speed in x, y, and z-axis will be for the next time point, given the previous sequence of the body motion. The outputs are values of predicted speed (i.e., v_x, v_y, v_z) for the next time point.

Given the current and previous speed traces, we build up our multi-task models including *multi-task LSTM* and *multi-task MLP* models to predict the body motion speed in three axes for

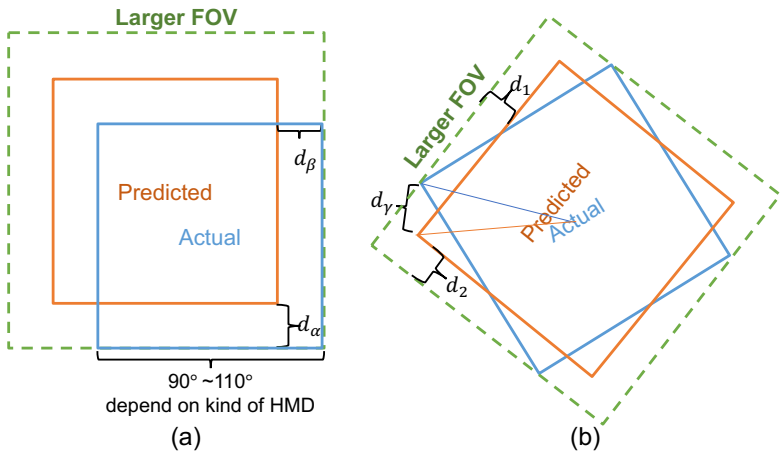


Figure 4.11: Selected FOV for two different types of relative positions between predicted FOV and actual FOV: (a) to address d_α and d_β, (b) to address d_γ.

the next time point and thus predict the viewing position b for the next time point (described in Section 4.3.1).

4.5.3 FOV Selection

After predicting body and head motion, we propose a *sliding window based FOV selection* method for pre-rendering, such that the selected FOV minimizes the effects of motion prediction error while also minimizing the selected FOV size. This method is also head motion prediction based since it selects the FOV size according to the estimated prediction error calculated by recent head motion prediction errors. Note that the method can only be applied to address head motion error since body motion prediction error can only be reduced by exploring better prediction models with higher precision (e.g., multi-task models presented in Section 4.5.2 2). Fig. 4.11 shows several different types of relative positions between predicted FOV and actual FOV, where blue square, orange square, and dashed green rectangle represent the actual FOV, predicted FOV, and pre-rendered larger FOV. The size of FOV is determined by the kind of HMD device, represented as the horizontal FOV of θ_h times vertical FOV of θ_v (e.g., $90° \times 90°$ for Samsung Gear VR, $110° \times 110°$ for HTC Vive).

Fig. 4.11(a) exhibits the angular distance between the actual and predicted FOVs in α and

β-axis as d_α and d_β, with no angular distance in the γ-axis. We can see that the actual FOV can be covered by the pre-rendered larger FOV via increasing the horizontal FOV to $\theta_h + 2d_\beta$ and the vertical FOV to $\theta_v + 2d_\alpha$. Fig. 4.11(b) demonstrates the angular distance between the actual and predicted FOVs in the γ-axis as d_γ without any angular distance in α and β-axis. The actual FOV can be covered by the pre-rendered larger FOV via increasing the horizontal FOV to $\theta_h + 2d_2$ and the vertical FOV to $\theta_v + 2d_1$. Since $d_1 \leq d_\gamma$ and $d_2 \leq d_\gamma$ (due to Pythagoras theorem [125]) shown in Fig. 4.11(b), in this case, we can select a larger FOV by increasing the horizontal FOV to $\theta_h + 2d_\gamma$ and the vertical FOV to $\theta_v + 2d_\gamma$. This is the minimal increase in FOV size compared to predicted FOV such that it minimizes adverse effects due to head motion prediction error. Therefore, by selecting a larger FOV of $\theta_h + 2d_\beta + 2d_\gamma$ as horizontal FOV and $\theta_v + 2d_\alpha + 2d_\gamma$ as vertical FOV, the actual FOV can be completely covered, eliminating the adverse effect of head motion prediction error. The new selected horizontal FOV θ'_h and vertical FOV θ'_v can be represented as follows in Equations 4.17 and 4.18:

$$\theta'_h = \theta_h + 2d_\beta + 2d_\gamma, \tag{4.17}$$

$$\theta'_v = \theta_v + 2d_\alpha + 2d_\gamma. \tag{4.18}$$

Note that when performing the FOV selection task before pre-rendering the view, the exact head motion prediction error for the next frame is unknown. Hence, in our *sliding window based FOV selection* method, we propose to use a sliding window of n_w frames and n_w denotes the sliding window size. Then we define the estimated value of d_α, d_β, d_γ (i.e., $\hat{d_\alpha}$, $\hat{d_\beta}$, $\hat{d_\gamma}$) as the average head motion prediction error $\overline{d_\alpha}$, $\overline{d_\beta}$, $\overline{d_\gamma}$ of the past n_w frames (i.e., frames in the sliding window) so as to calculate the new selected horizontal FOV θ'_h and vertical FOV θ'_v.

4.5.4 Prediction Error Determination

In Fig. 4.3, when the head and body motion, as well as the controlling command, arrive at the edge device, the actual motion can be obtained immediately after the motion decision and

there will be a prediction error determination comparing the actual motion with the prediction motion. We will see whether the head motion and body motion prediction error is within the thresholds using the following steps. For head motion, since we pre-render a larger FOV than actual FOV to reduce the effect of head motion prediction error. The determination of $d_{Head} \leq \varepsilon_H$ will be achieved by checking whether $|\hat{d}_\alpha - d_\alpha|, |\hat{d}_\beta - d_\beta|, |\hat{d}_\gamma - d_\gamma|$ are all within a given threshold ε_1. For body motion, the determination of $d_{Body} \leq \varepsilon_B$ will be achieved by checking whether d_x, d_y, d_z are all within a given threshold ε_2. To be sure that the actual view always within the pre-rendered view, the thresholds should be selected as low as possible. However, this will increase the probability of error determination, and hence doing the rendering and encoding again live, thereby increasing latency. On the other hand, setting this threshold too large may cause that the extreme case (e.g., having large head motion prediction error) cannot be efficiently identified. We empirically discuss different choices of given thresholds $\varepsilon_1, \varepsilon_2$ in Sections 4.6.3, 4.6.4, and 4.6.5.

4.6 Experimental Results

In this section, we describe our system setup, evaluation metrics, and experimental results.

4.6.1 System Setup and Dataset

The system setup of our experiments is shown in Fig. 4.12, where the rendering edge device is an Intel Core i7 Quad-Core processor with GeForce RTX 2060. It is equipped with a WiGig card connecting with the HTC Vive's link box using a cable. This link box is within the user's room and transmits rendered frames in a video format from the rendering edge device to the HMD. On the user side, there are the link box and two HTC lighthouse base stations in the room. Users were wearing an HTC Vive HMD equipped with Vive wireless adaptor [126] and using a controller if needed. Note the wireless adaptor and link box aim to transmit and receive the rendered frames using WiGig communications, while the HTC lighthouse base stations are set

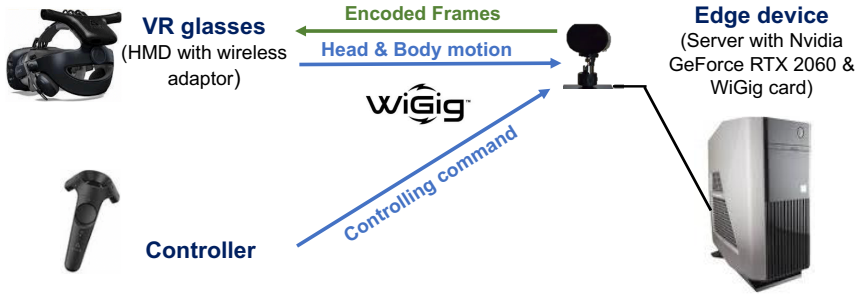


Figure 4.12: System setup.

Table 4.5: Dataset statistics.

Virtual Application	*Session*	*#Samples for Training*	*#Samples for Testing*
Museum	*VM1*	41,600	10,354
	VM2	80,484	20,076
	VM3	195,197	48,754
Rome	*RM1*	24,912	6,183
	RM2	48,586	12,103
	RM3	280,540	70,091

for capturing 6DoF motions (e.g., including head and body motion). The *walkable area* is around 3m×3m of free space in our experiments, which cannot exceed 4.5m×4.5m since the maximum distance between base stations is 5m [127]. All head and body motions on HMD were captured accurately using this HTC Lighthouse tracking system while the controller detected the user's controlling commands. For a software implementation, we implement our proposed techniques based on SteamVR SDK [128], OpenVR SDK [129] as well as the Unity game engine [13] for data collection, and use Keras [99] in Python for motion prediction.

We use 80% of the dataset for training the prediction model, and 20% for testing, ensuring the test data is from viewers which are different than those in training data. Table 4.5 presents the number of samples used as training data and testing data for each type of session of the two applications Virtual Museum and Virtual Rome (described in Section 4.4 and listed in Table 4.3). Moreover, in our experiments, proposed *single-task LSTM* and *single-task MLP* models learn parameters by minimizing mean square error, and training is terminated after 50 epochs in our experiments, while proposed *multi-task LSTM* and *multi-task MLP* models learn parameters by

minimizing multi-task loss function and training is terminated after 20 and 50 epochs respectively with a batch size of 32.

4.6.2 Evaluation Metrics and Baselines

Evaluation Metrics: We choose several popular metrics in sequential modeling to evaluate the performance on our prediction task:

- *Root Mean Square Error (RMSE)*:

$$RMSE = \sqrt{\frac{1}{|N_{test}|} \sum_{y \in S_{test}} \sum_{t=1}^{L} (y_t - \hat{y}_t)^2}, \tag{4.19}$$

- *Mean Absolute Error (MAE)*:

$$MAE = \frac{1}{|N_{test}|} \sum_{y \in S_{test}} \sum_{t=1}^{L} (y_t - \hat{y}_t), \tag{4.20}$$

where $|N_{test}|$ is the number of total time steps of all trajectories on the test set S_{test}.

Baselines: We consider the following baselines to compare against the performance of our proposed model:

- **Linear Acceleration Model (Lin-A)**: Following the work of [78, 111, 112], we compare against this linear regression model, which extrapolates trajectories with an assumption of linear acceleration. The Lin-A model employs the motion speed of the latest 3 time points to predict the expected motion speed.

- **Equal Acceleration Model (Eql-A)**: The Eql-A model is our modified version of Lin-A, where we assume the acceleration is approximately equal during a small time interval (e.g., 22ms). The advantage of this modification is as follows: by employing a smaller number of time points, the acceleration estimated may approach more the actual value for the following 11ms, than is achieved by the Lin-A model. We implement the Eql-A model

using motion speed of the latest 2 time points to predict the expected motion speed of the next time point.

4.6.3 Prediction Accuracy

1) Single-task Model

Tables 4.6, 4.7, 4.8, and 4.9 exhibit the results of our body motion and head motion prediction for the two applications respectively. Specifically, Tables 4.6 and 4.8 show the distance between actual and predicted body position in x, y, z-axis (denoted as d_x, d_y, d_z), while Tables 4.7 and 4.9 present the angular distance between actual and predicted head pose in α, β, γ-axis (denoted as $d_\alpha, d_\beta, d_\gamma$). Note that we use MSE as the loss function when doing training. In each table, we compare four models and can make the following observations:

- Tables 4.6 and 4.8, which report on the accuracy of body motion prediction, show that our LSTM model achieves smallest RMSE in each session and smallest MAE in most sessions except VM2 compared to Lin-A, Eql-A, and MLP models. It demonstrates the effectiveness of using our proposed LSTM model to predict body motion positions.

- Tables 4.7 and 4.9, which report on the accuracy of head motion prediction, show that while the LSTM model has smallest RMSE for session 1, the MLP model performs better (results in smaller RMSE) than other three models in sessions 2 and 3 for both the applications. Compared to session 1 (where users take a stroll about the room and have a relatively fixed trajectory), sessions 2 and 3 are more general and closer to normal 6DoF VR scenario. Thus, we can see that MLP is a more feasible model to do head motion prediction in general cases.

We can observe that (i) LSTM model achieves a better performance in every session of body motion prediction and session 1 of head motion prediction. These sessions have a relatively small range (e.g., body motion speed is mostly smaller than $\pm$1m/s), gradual variation and more

Table 4.6: Body motion prediction for Virtual Museum.

Session	***Model***	d_x (mm)		d_y (mm)		d_z (mm)	
		RMSE	***MAE***	***RMSE***	***MAE***	***RMSE***	***MAE***
VM1 (w/ Guidance; w/o Controller)	*Lin-A*	0.139	0.068	0.167	0.061	0.030	0.018
	Eql-A	0.079	0.037	0.096	0.033	0.021	**0.013**
	MLP	0.083	0.051	0.080	0.037	0.025	0.018
	LSTM	**0.061**	**0.035**	**0.074**	**0.030**	**0.019**	**0.013**
VM2 (w/o Guidance; w/o Controller)	*Lin-A*	0.094	0.045	0.099	0.041	0.048	0.021
	Eql-A	0.053	0.025	0.056	**0.023**	0.029	**0.013**
	MLP	0.044	0.029	0.047	0.030	0.032	0.015
	LSTM	**0.039**	**0.021**	**0.046**	0.029	**0.026**	**0.013**
VM3 (w/o Guidance; w/ Controller)	*Lin-A*	0.063	0.035	0.074	0.037	0.024	0.015
	Eql-A	0.036	**0.020**	0.042	0.022	0.017	0.011
	MLP	**0.032**	0.021	0.034	0.021	0.017	0.012
	LSTM	**0.032**	0.021	**0.033**	**0.019**	**0.015**	**0.010**

Table 4.7: Head motion prediction for Virtual Museum.

Session	***Model***	d_α $(')$		d_β $(')$		d_γ $(')$	
		RMSE	***MAE***	***RMSE***	***MAE***	***RMSE***	***MAE***
VM1 (w/ Guidance; w/o Controller)	*Lin-A*	0.64	0.34	0.96	0.43	0.48	0.21
	Eql-A	0.47	0.29	0.57	**0.27**	0.33	0.18
	MLP	0.51	0.35	0.77	0.48	0.40	0.27
	LSTM	**0.44**	**0.28**	**0.54**	0.30	**0.30**	**0.17**
VM2 (w/o Guidance; w/o Controller)	*Lin-A*	0.80	0.35	1.31	0.52	0.41	0.23
	Eql-A	0.49	**0.27**	0.78	**0.34**	0.32	0.19
	MLP	**0.47**	0.30	**0.64**	0.41	**0.31**	**0.18**
	LSTM	0.66	0.34	0.72	0.42	0.55	0.28
VM3 (w/o Guidance; w/ Controller)	*Lin-A*	0.61	0.35	1.38	0.61	0.33	0.21
	Eql-A	0.45	0.29	0.82	0.39	0.26	0.17
	MLP	**0.41**	**0.27**	**0.66**	**0.37**	**0.22**	**0.15**
	LSTM	0.48	0.30	0.99	0.55	0.28	0.17

regularity. (ii) MLP model performs better in sessions 2 and 3 of head motion prediction. These two sessions have a large value range (e.g., head motion can be up to $\pm 300^\circ$/s), quicker variation and more frequent fluctuations (e.g., head motion speed v_β has a large and abrupt change from -180°/s to 200°/s within 1s, shown in Fig. 4.6(e)). Note that although RMSE of head motion prediction achieved by MLP model is quite small, we still need to use proposed *FOV selection* method to address the possible challenging case (the extreme case where head motion prediction error is large) in head motion prediction, and minimize effects of motion prediction error while also minimizing selected FOV size.

Table 4.8: Body motion prediction for Virtual Rome.

Session	*Model*	d_x (mm)		d_y (mm)		d_z (mm)	
		RMSE	***MAE***	***RMSE***	***MAE***	***RMSE***	***MAE***
RM1 (w/ Guidance; w/o Controller)	*Lin-A*	0.174	0.086	0.299	0.084	0.046	0.022
	Eql-A	0.100	0.051	0.172	0.047	0.031	**0.017**
	MLP	0.118	0.075	0.098	0.062	**0.024**	0.018
	LSTM	**0.032**	**0.021**	**0.073**	**0.044**	**0.024**	0.019
RM2 (w/o Guidance; w/o Controller)	*Lin-A*	0.125	0.053	0.145	0.048	0.036	0.020
	Eql-A	0.074	0.032	0.085	0.030	**0.025**	**0.015**
	MLP	0.066	0.037	0.064	0.030	0.064	0.021
	LSTM	**0.058**	**0.030**	**0.065**	**0.032**	**0.025**	**0.015**
RM3 (w/o Guidance; w/ Controller)	*Lin-A*	0.074	0.041	0.077	0.041	0.034	0.019
	Eql-A	0.044	0.025	0.046	0.025	**0.023**	**0.013**
	MLP	0.040	0.025	0.041	0.026	0.077	0.040
	LSTM	**0.040**	**0.024**	**0.040**	**0.024**	**0.023**	**0.013**

Table 4.9: Head motion prediction for Virtual Rome.

Session	*Model*	d_α ($'$)		d_β ($'$)		d_γ ($'$)	
		RMSE	***MAE***	***RMSE***	***MAE***	***RMSE***	***MAE***
RM1 (w/ Guidance; w/o Controller)	*Lin-A*	0.71	0.47	1.32	0.61	0.40	0.27
	Eql-A	0.55	0.38	0.79	**0.39**	0.30	**0.21**
	MLP	0.55	0.38	0.80	0.49	0.30	**0.21**
	LSTM	**0.53**	**0.36**	**0.73**	0.47	**0.29**	0.22
RM2 (w/o Guidance; w/o Controller)	*Lin-A*	0.92	0.57	2.53	0.66	0.56	0.34
	Eql-A	0.66	0.43	1.48	**0.44**	0.39	0.26
	MLP	**0.63**	**0.42**	**1.34**	0.46	**0.37**	**0.25**
	LSTM	0.64	0.43	1.52	0.55	1.23	0.30
RM3 (w/o Guidance; w/ Controller)	*Lin-A*	0.88	0.50	1.57	0.72	0.44	0.27
	Eql-A	0.63	0.38	0.98	0.49	0.33	0.21
	MLP	**0.57**	**0.36**	**0.82**	**0.43**	**0.28**	**0.18**
	LSTM	0.60	0.39	0.89	0.52	0.35	0.25

Next, we study what the values of ε_2 should be in the prediction error determination technique (Section 4.5.4), where the prediction motion is compared with actual motion when the head and body motion, as well as controlling command, arrive at the edge device. The determination of body motion prediction error is checking whether d_x, d_y, d_z are all within a given threshold ε_2. Fig. 4.13 presents the body motion prediction error using the LSTM model in the RM3 session. In Fig. 4.13, body motion prediction using the LSTM model achieves that around 99.99% (i.e., 0.9999) of time points satisfy the d_x <0.6mm, d_y <0.7mm, d_z <0.45mm. Thus, we can observe that if we set the given threshold ε_2 as 1mm, less than 99.99% of time points

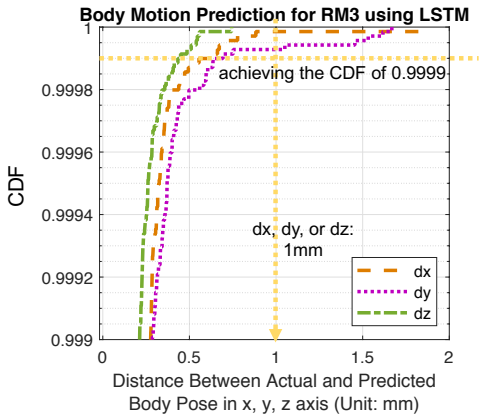


Figure 4.13: Body motion prediction error using the LSTM model in the RM3 session.

can be determined as 'correct' for body motion prediction in the proposed system, meaning that there is less than 1 frame on average among 10,000 pre-rendered frames will be 'incorrect' while the rest of more than 9,999 pre-rendered frames will pass the body motion error determination successfully.

2) Multi-task Model

Fig. 4.14(a) and (b) exhibit the results of our body motion prediction for two application sessions VM1 and VM3 respectively. In each figure, we compare six models (single-task and multi-task models) and can make the following observations.

- Fig. 4.14(a) shows the RMSE of d for body motion prediction error in VM1, where we can see the *multi-task LSTM* model achieves 56.2%, 23.7%, 18.5%, 3.2%, 26.7% improvement compared to *Lin-A, Eql-A, MLP, LSTM, multi-task MLP* models. Our proposed *multi-task LSTM* model achieves the smallest RMSE of d (i.e., 0.096mm) compared to other models.

- Fig. 4.14(b) presents the RMSE of d for body motion prediction error in VM3, where we can see the *multi-task LSTM* model achieves 53.5%, 19.7%, 6.4%, 4.4%, 10.5% improvement compared to *Lin-A, Eql-A, MLP, LSTM, multi-task MLP* models respectively. Our proposed *multi-task LSTM* model achieves the smallest RMSE of d (i.e., 0.046mm) compared to

other models. In Fig. 4.14(a)(b), the reason that the prediction error for body motion in VM1 is larger than VM3 is that users continuously walk without stopping by any place in VM1 while they tend to have less body motion and teleport to other place using the controller in VM3.

- Similarly, in other sessions such as RM3, for RMSE of d for body motion prediction error, the *multi-task LSTM* model achieves 46.6%, 11.9%, 37.5%, 1.3%, 5.9% improvement compared to *Lin-A, Eql-A, MLP, LSTM, multi-task MLP* models. The *multi-task LSTM* model still works better than other five models. Moreover, in some cases like VM2, the *multi-task LSTM* model achieves the same RMSE of d for body motion prediction error with *LSTM* model (i.e., *multi-task LSTM* model has 54.1%, 19.3%, 8.0%, 0%, 6.2% improvement in RMSE of d compared to *Lin-A, Eql-A, MLP, LSTM, multi-task MLP* models). The *multi-task LSTM* model has a smaller RMSE of d_y (i.e., 0.042mm) compared to the *LSTM* model (i.e., 0.046mm) as well as a larger RMSE of d_x and d_z. To achieve the smallest RMSE of d for body motion prediction error, in this case, we can consider using *multi-task LSTM* model to predict body motion in the y-axis and two single-task *LSTM* models to predict body motion in the x and z-axis, so that the RMSE of d for this combined models choice is 0.063mm, smaller than 0.066mm obtained by single-task *LSTM* models as well as *multi-task LSTM* model. Thus, we can always improve the performance by combing three trained models (*multi-task LSTM* with single-task *LSTM* models) if each of them has the smallest RMSE of d_x, d_y, and d_z to achieve the smallest RMSE of d.

4.6.4 Runtimes

Training and Prediction Times

Next, we briefly discuss the training times and inference times taken by our proposed prediction models on the edge device selected (Intel Core i7 Quad-Core processor with GeForce RTX 2060). Note that the training for a proposed model is done offline only once for a session

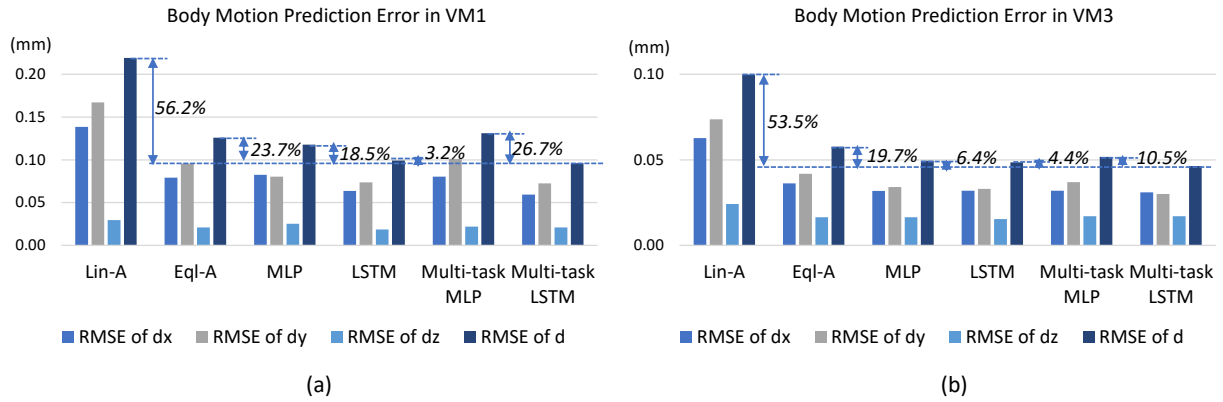


Figure 4.14: Body motion prediction error in VM1 and VM3 sessions comparing the multi-task LSTM model with other predictive models.

with the training samples for that session, and the prediction (testing) is done online for new users, however one frame ahead to predict motion in advance. Hence, the training times do not affect the end-to-end latency of the system. Since the prediction is done for the user's head and body motion one frame ahead in advance, the prediction time as well as FOV selection time have to be less than 1ms (presented in Table 4.2).

The training time can be different for each session depending on the number of training samples used. In our experiments, for RM3 (which has the largest training data size among all sessions), the training times of one epoch for each *single-task LSTM* and *single-task MLP* models are around 150 seconds and 40 seconds respectively, while the training times of one epoch for the *multi-task LSTM* and *multi-task MLP* models are around 270 seconds and 45 seconds respectively. Thus for RM3, the training times for each *single-task LSTM* model and *multi-task LSTM* model are around 2 hours and 1.5 hours respectively, while the training times for each *single-task MLP* model and *multi-task MLP* model are around 0.55 hours and 0.6 hours respectively. The training times for all the other applications/sessions are lower than RM3 (e.g., for VM1, the training time for each *single/multi-task LSTM/MLP* model is lower than 18 minutes).

The prediction (testing) times, on the other hand, only marginally varies between different applications and sessions. The average prediction times over all the application sessions consid-

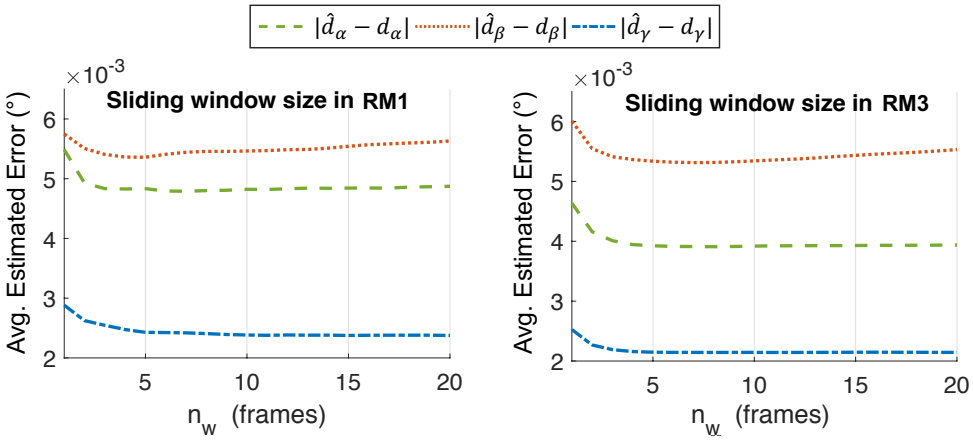


Figure 4.15: Average estimated error in α, β, γ-axis caused by different choices of sliding window size in RM1 and RM3 sessions.

ered in our experiments are the following: 0.09ms for each *single-task LSTM* model, 0.04ms for each *single-task MLP* model, 0.38ms for the *multi-task LSTM* model, and 0.04ms for the *multi-task MLP* model. The above shows that our proposed head and body motion prediction models can execute in real-time on the edge node, and since they are well below the time of 1ms (described in the next paragraph), the predictions are feasible to be performed in advance for the user's head and body motion of next time point.

Total Times of Prediction and FOV Selection

Prediction time consists of head and body motion predictions: head motion prediction using three *single-task MLP* (i.e., 0.12ms) and body motion prediction using either *option (a) multi-task LSTM* (i.e., 0.38ms) or *option (b) multi-task LSTM* combined with one or two *single-task LSTM* models (i.e., 0.38+0.09ms or 0.38+0.18ms). Thus prediction time for head and body motions is 0.5ms – 0.68ms. FOV selection includes two parts: two simple addition operations to calculate horizontal FOV and vertical FOV in Equations 4.17 and 4.18, and three averaging operations to calculate the estimated value of d_α, d_β, d_γ (i.e., $\hat{d}_\alpha$, $\hat{d}_\beta$, $\hat{d}_\gamma$) as the average head motion prediction error $\overline{d_\alpha}$, $\overline{d_\beta}$, $\overline{d_\gamma}$ of the past n_w frames. FOV selection can be achieved in 0.00016ms when $n_w = 5$ (proved to be a good choice in Section 4.6.5). Thus, the total times of *prediction* and *FOV selection* is within 1ms.

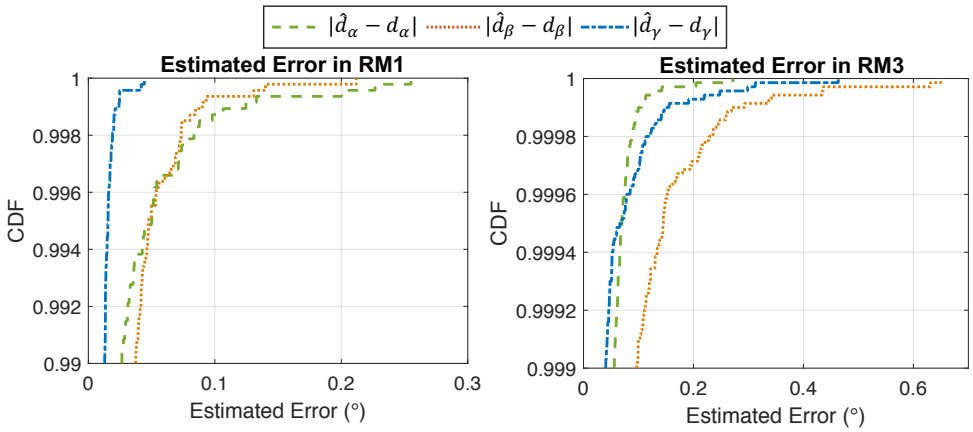


Figure 4.16: CDF of estimated error in α, β, γ-axis when the sliding window size $n_w = 5$ in RM1 and RM3 sessions.

4.6.5 FOV Selection

Next, we evaluate the performance of our proposed *sliding window based FOV selection* method, described in Section 4.5.3. As mentioned before, a sliding window of n_w frames (n_w denotes sliding window size) is used to estimate and obtain the new $d_\alpha, d_\beta, d_\gamma$, the new selected horizontal FOV θ'_h, and vertical FOV θ'_v before pre-rendering. We have described how to calculate the estimated value of d_α, d_β, d_γ (i.e., $\hat{d}_\alpha$, $\hat{d}_\beta$, $\hat{d}_\gamma$) in Section 4.5.3. Fig. 4.15 shows the absolute value of the average estimated error in each axis (i.e., average $|\hat{d}_\alpha - d_\alpha|, |\hat{d}_\beta - d_\beta|, |\hat{d}_\gamma - d_\gamma|$) caused by different choices of sliding window size n_w (ranging from 1 to 20 frames) in RM1 and RM3 sessions. We can see that the smallest average estimated error can be achieved when $n_w = 5$ in both the sessions. The average estimated error can be as low as less than 5.5×10^{-3} degree in each axis, showing the efficiency of our approach. By achieving the low average estimated error, we can have a better estimation of $d_\alpha, d_\beta, d_\gamma$, so that can finally reduce the adverse effect of head motion prediction error.

We also study what the values of ε_1 should be in the prediction error determination technique (Section 4.5.4), where the prediction motion is compared with actual motion when the head and body motion, as well as controlling command, arrive at the edge device. The determination of head motion prediction error is checking whether $|\hat{d}_\alpha - d_\alpha|, |\hat{d}_\beta - d_\beta|, |\hat{d}_\gamma - d_\gamma|$

are all within a given threshold ε_1. Fig. 4.16 shows the CDF of estimated error in α, β, γ-axis when the sliding window size $n_w = 5$ in RM1 and RM3 sessions respectively. Thus, we can observe that if a given threshold ε_1 is set as 1°, the estimated errors in each axis (i.e., $|\hat{d}_\alpha - d_\alpha|, |\hat{d}_\beta - d_\beta|, |\hat{d}_\gamma - d_\gamma|$) are smaller than ε_1 all the time for both RM1 and RM3 sessions, meaning that the head motion prediction is always 'correct' in this case.

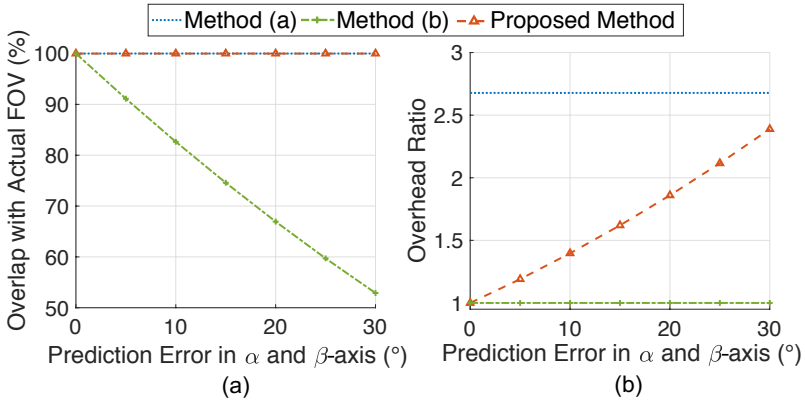


Figure 4.17: (a) Overlap of pre-rendered predicted view with actual FOV versus head motion prediction error in α and β-axis, (b) Overhead ratio versus head motion prediction error in α and β-axis.

For further performance evaluation, we compare our proposed *sliding window based FOV selection method* (Section 4.5.4) with ***method (a)*** selected FOV is a fixed larger FOV, and ***method (b)*** using predicted FOV as the selected FOV. Specifically, for our proposed *sliding window based FOV selection* method, we use experimental results that average estimated in α, β-axis are 4.8×10^{-3} and 5.5×10^{-3} degrees respectively, shown in Fig. 15. For ***method (a)***, the fixed larger FOV has the size of $(110+60)^\circ \times (110+60)^\circ$ to cover potential prediction error within 30°. For ***method (b)***, the selected FOV is predicted FOV in size of $110^\circ \times 110^\circ$. As for evaluation metrics, we calculate (i) *overlap* of pre-rendered predicted view with actual FOV (e.g., $110^\circ \times 110^\circ$ for HTC Vive), and (ii) *overhead ratio* for pre-rendering computation and network bandwidth needed to transmit rendered FOV from edge device to VR glasses, defined as the *pre-rendering view size* divided by the *actual FOV size*. For instance, when the pre-rendering view size is $120^\circ \times 120^\circ$, the *overhead ratio* for pre-rendering computation and network bandwidth needed can be calculated

as 1.19 according to our definition.

As described in Section 4.6.3, our proposed *sliding window based FOV selection method* can address the extreme cases where head motion prediction error is large. We compare the above three methods when dealing with head motion prediction error ranging from 0 to 30° in α and β-axis. Fig. 4.17(a)(b) show the overlap of pre-rendered predicted view with actual FOV and overhead ratio versus head motion prediction error in the α and β-axis (the coordinate of head motion shown in Fig. 4.4(c)). Note that when the x-axis of Fig. 4.17(a)(b) equals to 10°, we consider the situation of head motion prediction error in α and β-axis (i.e., d_α, d_β) are both 10° and no head motion prediction error in the γ-axis (i.e., d_γ). In Fig. 4.17, we can see that our proposed sliding window based FOV selection method achieves (i) the overlap with actual FOV as high as 99.991% (which is close to 100% achieved by *method (a)* and better than *method (b)*), and (ii) the corresponding overhead ratio is always smaller than *method (a)*.

For example, in our experiments, we observe that when the prediction errors in α and β-axis equal to 5°, our proposed *sliding window based FOV selection* method achieves an *overhead ratio* of 1.19, compared to an *overhead ratio* of 2.39 for *method (a)*, which corresponds to around 50% reduction of *overhead ratio* and 47% saving of bitrates (bandwidth needed) compared to *method (a)*. Thus, the high overlap with actual FOV and small overhead ratio illustrate that our proposed *sliding window based FOV selection* method has a good user experience (almost no miss of actual FOV) and low *overhead ratio* for pre-rendering computation as well as network bandwidth needed to transmit rendered FOV from edge device to VR glasses, compared to *methods (a)* and *(b)*.

4.6.6 Effect on User Experience

To evaluate the effect on user experience caused by the prediction error between the actual view and the predicted view which will be pre-rendered and delivered to the user, we propose following metric. Assume that we have two views V_1 and V_2 in the RGB format. Firstly, we

Figure 4.18: (a) Actual user's view; (b) Predicted user's view with x-axis error $\Delta x = 0.1m$; (c) I_{dif} obtained from views in (a)(b).

convert the RGB images (V_1 and V_2) to grayscale intensity images I_1 and I_2 by eliminating the hue and saturation information while retaining the luminance [130]. For each pixel i in the grayscale intensity images, we calculate the difference between the two intensity images, I_{dif}, as follows.

$$I_{dif}(i) = \begin{cases} I_1(i) - I_2(i), \text{if } I_1(i) \geq I_2(i) \\ \\ 0, \quad \text{otherwise} \end{cases} \tag{4.21}$$

Note that we set the I_{dif} as 0 in the second case of Equation 4.21, because otherwise the motion change of the same object will be presented in I_{dif} twice: positive and negative respectively. Thus we only keep the positive one (i.e., the first case in Equation 4.21) to evaluate the difference between the two views. Fig. 4.18 presents an example of two views and the corresponding I_{dif}. In Fig. 4.18(c), we can see that most of pixels in the view have the intensity value of 0 while the residual pixels have intensity values larger than or equal to 1. We define the *percentage of mismatched pixels* as

$$R_{dif} = \frac{N_{dif}}{N_{frame}}, \tag{4.22}$$

where N_{dif} represents the number of pixels which have difference in grayscale intensity and N_{frame} is the total number of pixels per frame.

Fig. 4.19 illustrates the average *percentage of mismatched pixels* caused by body motion prediction error. Due to the large number for each session in the test dataset, we calculate this value by doing body motion prediction and rendering corresponding predicted as well as actual

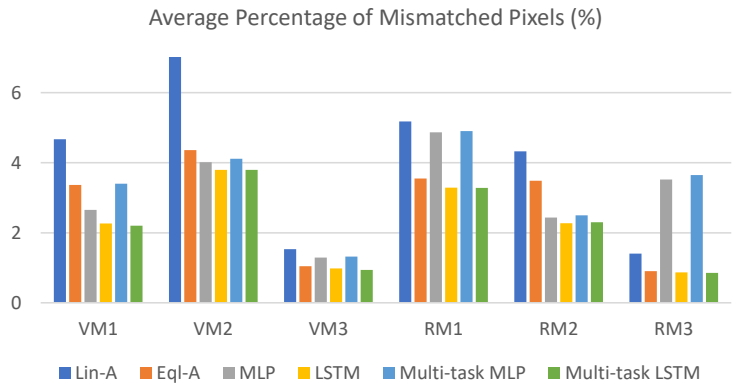


Figure 4.19: The average *percentage of mismatched pixels* for different models during each session.

views for 300 randomly selected samples from test data for every session. Fig. 4.19 demonstrates that compared to other models in each session, our proposed *multi-task LSTM* and *LSTM* models achieves less adverse effect on user experience caused by the body prediction error (denoted with green and yellow bars). Using the *multi-task LSTM* model, the average *percentage of mismatched pixels* can be smaller than 1% in both VM3 and RM3 sessions.

Next, we define the *percentage of pixels* as

$$R_p = \frac{N_p}{N_{frame}}, \tag{4.23}$$

where N_p represents the number of pixels and N_{frame} is the total number of pixels per frame. For each pixel, it can have a value of grayscale intensity difference in I_{dif} (which equals to a integer between 0 to 255). Apart from discussion in Section 4.6.3 1), by using this metric of the *percentage of pixels*, we further study what the values of ε_2 should be in the prediction error determination technique (Section 4.5.4), where the prediction motion is compared with actual motion. The determination of body motion prediction error is checking whether d_x, d_y, d_z are all within a given threshold ε_2. Fig. 4.20 shows an example of the percentage of pixel versus different d_x, d_y, and d_z in Virtual Museum application. We can observe that when $d_x = 1$mm, $d_y = 1$mm, $d_z = 1$mm, the *percentage of pixels* is more than 97%, 95%, 97% respectively corresponding to *pixel difference* less than 3 (pixel difference = 0, 1, or 2), which means ε_2 = 1mm can be a good

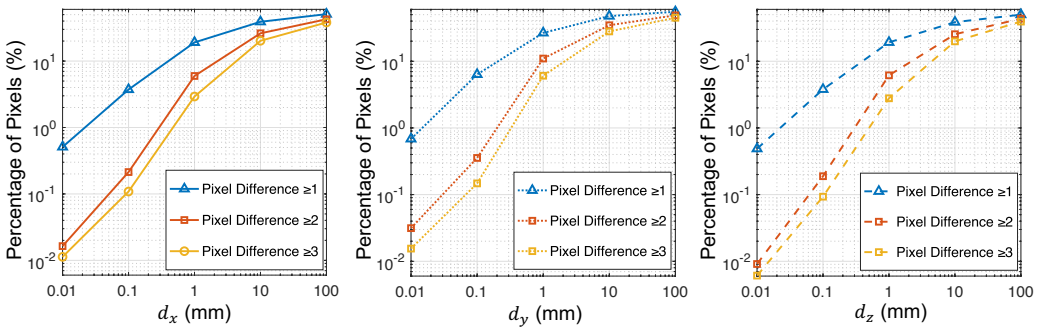


Figure 4.20: The percentage of pixels having pixel difference for versus d_x, d_y, and d_z.

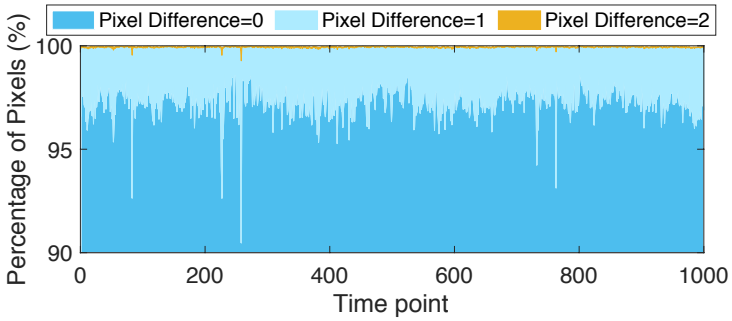


Figure 4.21: The percentage of pixels versus time points achieved by the multi-task LSTM model during VM2.

choice for the body prediction error determination.

Furthermore, for our proposed multi-task LSTM model, we evaluate the adverse effect caused by body motion prediction error using metric of the *percentage of pixels*. Fig. 4.21 presents the *percentage of pixels* versus the time points achieved by the multi-task LSTM model during the VM2 session. We can see that most of the time, the *percentage of pixels* for pixel difference = 0 is larger than 96% (equivalent to the average *percentage of mismatched pixels* smaller than 4%). The average *percentage of pixels* for pixel difference = 0, 1, 2, 3, 4, 5 equals to 97.43%, 2.49%, 0.03%, 0.009%, 0.006%, 0.002% respectively, illustrating that the difference between actual view and predicted view is very small. Thus, our proposed multi-task LSTM model performs well in terms of small adverse effect caused by body motion prediction error.

4.6.7 Latency Measurement

We first collect network traces by doing ping tests to record roundtrip transmission latency. Fig. 4.22 shows the roundtrip transmission latency measured during 60s in two indoor locations respectively. Specifically, we ping www.google.com in *Location 1* using wired wide area network (WAN) with WiFi connection, and www.taobao.com in *Location 2* using wired WAN. We can see the average roundtrip transmission latency for Locations 1 and 2 as 10.93ms and 9.17ms respectively. We also collect a rendering latency trace in Virtual Museum using Unity. Fig. 4.23 presents the rendering latency during 60s, where the average rendering latency is around 6.66ms.

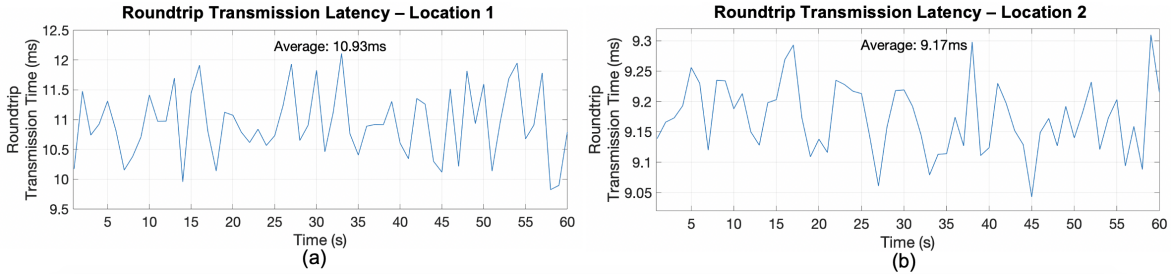


Figure 4.22: Roundtrip transmission latency traces (a) for Location 1, and (b) for Location 2.

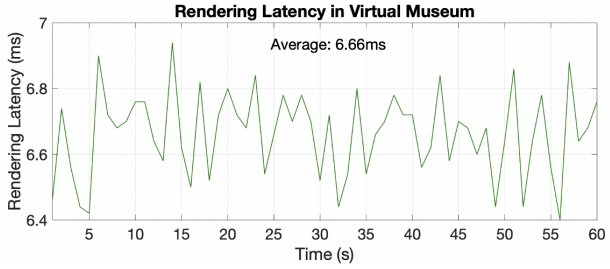


Figure 4.23: Rendering latency trace in Virtual Museum.

As described in Section 4.5.4 and Fig. 4.3, the prediction error determination is implemented with the parameter setting of ε_1 and ε_2. Fig. 4.24 shows the determination results for two settings of ε_1 and ε_2 in VM2 session, covering around 20k time points. From Fig. 4.24, we can see that the accuracy for 'correct' cases is larger than 99.99% (i.e., number of 'incorrect' equals to 3 within 20076 time points) for $\varepsilon_1 = 1^\circ$ and $\varepsilon_2 = 1mm$, and is larger than 99.94% (i.e., number of 'incorrect' equals to 13 within 20076 time points) for $\varepsilon_1 = 1^\circ$ and $\varepsilon_2 = 0.5mm$ respectively.

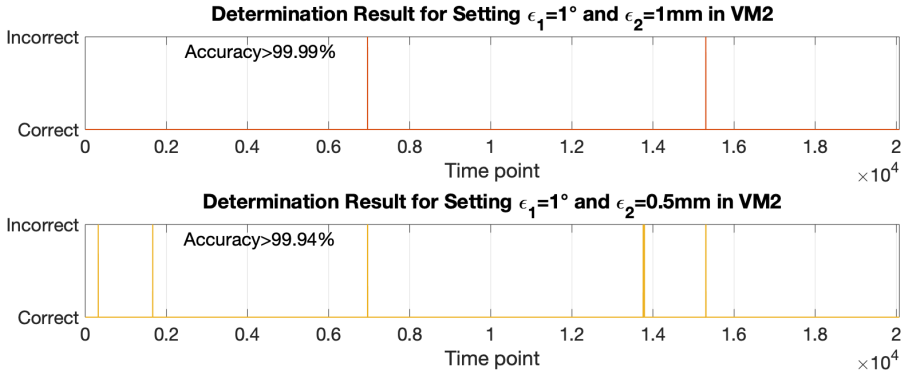
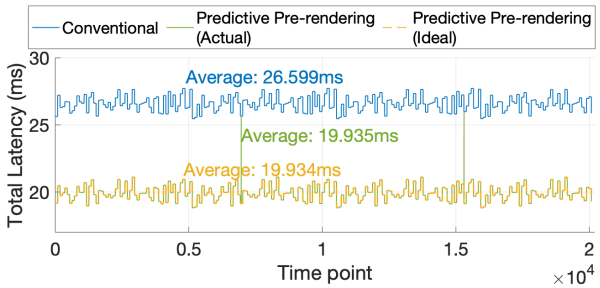


Figure 4.24: Determination results for two settings of ε_1 and ε_2 in VM2 session.

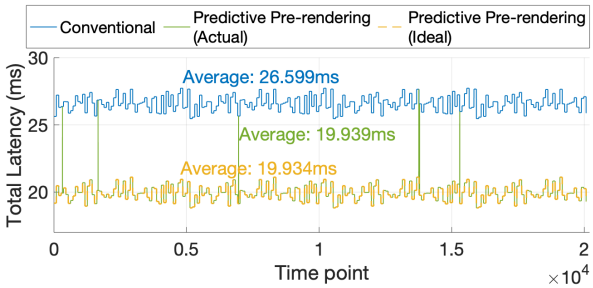
In our experiments, for each time point (e.g., 11ms as time slot), we consider the roundtrip transmission latency and rendering latency values at the beginning of time slot in the roundtrip transmission latency traces of Fig. 4.22(a)(b) and rendering latency trace of Fig. 4.23 as the ongoing roundtrip transmission latency and rendering latency condition for the current time slot, and run our proposed algorithm with head and body motion traces. Note that we emulate the roundtrip transmission latency and rendering latency condition for 20k time points in Fig. 4.24 (corresponding to around 220s) by repeating latency values of 60s-traces in Figs. 4.22 and 4.23. We also assume the latency of encoding and decoding as 6ms and 3ms.

Figs. 4.25 and 4.26 present total latency results for Locations 1 and 2 respectively with two parameter settings of ε_1 and ε_2. The blue line represents the total latency for the conventional method, while the green line represents the actual total latency for our proposed predictive pre-rendering method. The yellow dashed line represents the ideal total latency using our proposed method with 100% prediction accuracy in motion prediction. From the figures, we can make the following observations:

1) From Figs. 4.25 and 4.26, we can observe that our proposed predictive pre-rendering achieves around 25% reduction of total latency compared to the conventional method in both Locations 1 and 2. Specifically, from Figs. 4.25(a) and (b), for Location 1, we can see that the average total latency for our proposed predictive pre-rendering is 19.935ms and

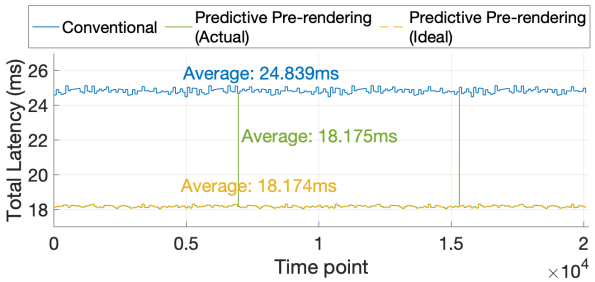


(a) Location 1

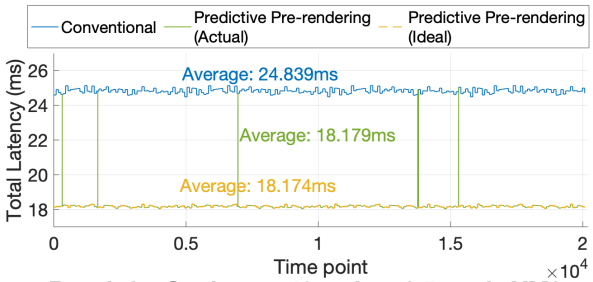


(b) Location 1

Figure 4.25: Total latency for Location 1 with two settings of ε_1 and ε_2 in VM2 session.



(a) Location 2



(b) Location 2

Figure 4.26: Total latency for Location 2 with two settings of ε_1 and ε_2 in VM2 session.

19.939ms respectively, achieving a latency reduction of 25.05% and 25.04% compared to the conventional method. Moreover, for Location 2, Figs. 4.26(a) and (b) also demonstrate that the average total latency for our proposed predictive pre-rendering is 18.175ms and 18.179ms respectively, achieving a latency reduction of 26.83% and 26.81% compared to the conventional method.

2) In Locations 1 and 2, our predictive pre-rendering method performs significantly better (less total latency) than the conventional method, and achieves similar average total latency compared to the ideal results of predictive pre-rendering (i.e., no more than 0.005ms difference in average total latency).

4.7 Conclusion

In this chapter, we propose a head and body motion prediction model for 6DoF VR applications, to enable predictive pre-rendering using edge intelligence and thus address latency challenge in edge computing-based 6DoF VR. We present a multi-task LSTM model and an MLP model to learn general head and body motion patterns and predict the future viewing direction and position based on past traces. We also develop a FOV selection technique for pre-rendering a larger FOV to reduce head motion prediction error and the motion error determination technique as part of the system mechanism. Our method shows good performance on a real motion trace dataset with high precision and a reduction of around 25% for average total latency compared to the conventional method.

4.8 Acknowledgments

Chapter 4 contains the reprints of Xueshi Hou, Jianzhong Zhang, Madhukar Budagavi and Sujit Dey, "Head and Body Motion Prediction to Enable Mobile VR Experiences with Low Latency," *Proc. of the 2019 IEEE Global Communications Conference*, 2019; and Xueshi Hou and Sujit Dey, "Motion Prediction and Pre-Rendering at the Edge to Enable Ultra-Low Latency Mobile 6DoF Experiences," *IEEE Open Journal of the Communications Society*, 2020. The dissertation author is the primary investigator and author of each of these papers.

Chapter 5

Conclusion

This thesis has presented several methodologies to address existing challenges in enabling high-quality, lightweight, and mobile VR experiences. The presented methodologies help to address the ultra-high bandwidth and ultra-low latency challenges in edge/cloud-based VR solution.

Chapter 2 has presented a multi-user hybrid-cast approach to significantly reduce the total bitrate needed to stream high-quality videos to multiple users in a virtual space application. Instead of unicasting the video of each user view, we introduce the novel approach which allows unicasting much lower-bandwidth residual views, together with one or more common view(s). Then we propose an efficient way of identifying common and residual views. To minimize the total bitrate, we develop a smart real-time algorithm for grouping the users of the virtual space, using a novel grouping metric. Our experimental results demonstrate the effectiveness of our proposed grouping algorithm both in terms of optimal performance and speed. Furthermore, the results show that the total bitrate needed to transmit multiple user views can be significantly reduced by up to 55%, and thus provide better user experience (less delay) under constrained network.

Chapter 3 has presented a predictive adaptive streaming approach in order to reduce the

latency and bandwidth needed to deliver 360-degree videos and cloud/edge-based VR applications, leading to better mobile VR experiences. We present a multi-layer LSTM model which can learn general head motion pattern and predict the future viewpoint based on past traces. Our method outperforms state-of-the-art methods on a real head motion trace dataset and shows great potential to reduce bandwidth while keeping a good user experience (i.e., high PSNR).

Chapter 4 has presented a head and body motion prediction model for 6DoF VR applications, to enable predictive pre-rendering using edge intelligence and thus address latency challenge in edge computing-based 6DoF VR. We present a multi-task LSTM model and an MLP model to learn general head and body motion patterns and predict the future viewing direction and position based on past traces. We also develop a FOV selection technique for pre-rendering a larger FOV to reduce head motion prediction error and the motion error determination technique as part of the system mechanism. Our method shows good performance on a real motion trace dataset with high precision and a reduction of around 25% for average total latency compared to the conventional method.

In the future, we would like to extend our research in the following directions. Firstly, for 360-degree video streaming, we would like to study and develop more comprehensive models considering the video content including saliency map and video type to improve viewpoint prediction accuracy further. We would also like to explore various tile as well as projection options, develop further refined formulation and algorithms for adaptive bitrate streaming, and perform more detailed timing analysis considering real-time tile encoding.

Secondly, for live streaming of 6DoF VR applications, we plan to further develop and evaluate the proposed edge-based predictive pre-rendering approach from latency perspectives. Besides, we would like to explore predicting more time points and pre-delivering from edge to HMD, and consider multiple users and more possible gaming effects of the controller in applications, so as to address more challenging latency scenario.

Thirdly, we would like to perform subjective studies to understand and quantify user

experience using our proposed approaches for 3DoF and 6DoF VR experiences. Apart from using the metrics such as PSNR and MSE of video content to evaluate encoded video distortion, we plan to implement subjective tests to study the possible effect on user experience caused by prediction error or fluctuation of total latency.

Finally, we would like to explore more streaming options or strategies within the FOV. According to the observation that a user can be more focusing on the central area of FOV when moving (i.e., different sensibility), we would like to consider streaming relatively high quality for central view and low quality for the rest of FOV. For the size of central area, we plan to study how the motion speed can impact the acceptable central area size for users. All of these issues need to be considered and explored in our future work.

Bibliography

[1] C. Wiltz, "Five major challenges for vr to overcome," April 2017. [Online]. Available: https://www.designnews.com/electronics-test/5-major-challenges-vr-overcome/187151205656659/

[2] L. Cherdo, "Types of vr headsets," 2019. [Online]. Available: https://www.aniwaa.com/guide/vr-ar/types-of-vr-headsets/

[3] Oculus, "Oculus rift," 2019. [Online]. Available: https://www.oculus.com/rift/

[4] HTC, "Htc vive," 2019. [Online]. Available: https://www.vive.com/us/

[5] ——, "Htc focus," 2019. [Online]. Available: https://www.vive.com/cn/product/vive-focus-en/

[6] Oculus, "Oculus go," 2019. [Online]. Available: https://www.oculus.com/go/

[7] Samsung, "Samsung gear vr," 2019. [Online]. Available: https://www.samsung.com/us/mobile/virtual-reality/

[8] Google, "Google daydream," 2019. [Online]. Available: https://vr.google.com/daydream/

[9] X. Hou, Y. Lu, and S. Dey, "Wireless vr/ar with edge/cloud computing," in *Proc. ICCCN*, 2017, pp. 1–8.

[10] Qualcomm, "Whitepaper: Making immersive virtual reality possible in mobile," 2016. [Online]. Available: https://www.qualcomm.com/media/documents/files/whitepaper-making-immersive-virtual-reality-possible-in-mobile.pdf

[11] S. Wang and S. Dey, "Adaptive mobile cloud computing to enable rich mobile multimedia applications," *IEEE Trans. on Multimedia*, vol. 15, no. 4, pp. 870–883, June 2013.

[12] S. Dey, Y. Liu, S. Wang, and Y. Lu, "Addressing response time of cloud-based mobile applications," in *Proc. of Int. Workshop on Mobile Cloud Comput. & Netw.*, 2013, pp. 3–10.

[13] U. Technologies, "Unity," 2019. [Online]. Available: https://unity3d.com/

[14] Oculus, “Oculus rift,” 2018. [Online]. Available: https://www.oculus.com

[15] Samsung, “Samsung gear vr,” 2018. [Online]. Available: http://www.samsung.com/global/galaxy/gear-vr/

[16] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, “A survey of mobile cloud computing: architecture, applications, and approaches,” *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[17] Y. Xu and S. Mao, “A survey of mobile cloud computing for rich media applications,” *IEEE Wireless Communications*, vol. 20, no. 3, pp. 46–53, 2013.

[18] X. Hou, Y. Lu, and S. Dey, “A novel hyper-cast approach to enable cloud-based virtual classroom applications,” in *Proc. ISM*, 2016, pp. 533–536.

[19] Wikipedia, “Rendering pipeline,” 2018. [Online]. Available: https://en.wikipedia.org/wiki/Graphics_pipeline

[20] Y. Lu and S. Dey, “Javre: a joint asymmetric video rendering and encoding approach to enable optimized cloud mobile 3d virtual immersive user experience,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 4, pp. 544–559, 2016.

[21] R. H. CCITT, “Video codec for audiovisual services at px 64 kbit/s,” *Draft Revision*, Mar. 1990.

[22] ——, “Video coding for low bit rate communication,” Nov. 1995.

[23] I. J. 1, “Coding of moving pictures and associated audio for digital storage media at up to about 1.5 mbit/s-part 2: Video,” *ISO/IEC 11172-2 (MPEG-1)*, 1993.

[24] ——, “Coding of audio-visual objects - part 2: Visual,” *ISO/IEC 14496-2 (MPEG-4 Vis. Version 1)*, Apr. 1999.

[25] I.-T. I. J. 1, “Generic coding of moving pictures and associated audio information part 2: Video,” *ITU-T Rec. H.262 ISO/IEC 13818-2 (MPEG-2 Video)*, Nov. 1994.

[26] ——, “Advanced video coding for generic audio-visual services,” *ITU-T Rec. H.264 ISO/IEC 14496-10 (AVC)*, May 2003.

[27] ITU-T and I. JTC, “High efficiency video coding,” *ITU-T Rec. H.265 and ISO/IEC 23008-2 (HEVC)*, Jan. 2013.

[28] W. project, “Vp9 video codec,” 2018. [Online]. Available: http://www.webmproject.org/vp9

[29] Wikipedia, “Av1,” 2018. [Online]. Available: https://en.wikipedia.org/wiki/AOMedia_Video_1

[30] Y. Liu, S. Wang, and S. Dey, "Content-aware modeling and enhancing user experience in cloud mobile rendering and streaming," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 4, no. 1, pp. 43–56, Mar. 2014.

[31] Y. Lu, Y. Liu, and S. Dey, "Cloud mobile 3d display gaming user experience modeling and optimization by asymmetric graphics rendering," *IEEE Journal of Selected Topics in Signal Processing*, vol. 9, no. 3, pp. 517–532, April 2015.

[32] W. Cai and V. C. Leung, "Multiplayer cloud gaming system with cooperative video sharing," in *Proc. CloudCom*, 2012, pp. 640–645.

[33] edX, "Mooc," 2018. [Online]. Available: https://www.edx.org

[34] A. Checko, H. L. Christiansen, Y. Yan, L. Scolari, G. Kardaras, M. S. Berger, and L. Dittmann, "Cloud ran for mobile networks – a technology overview," *IEEE Communications surveys & tutorials*, vol. 17, no. 1, pp. 405–426, 2015.

[35] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, "Softcell: Scalable and flexible cellular core network architecture," in *Proc. CoNEXT*, 2013, pp. 163–174.

[36] D. Lecompte and F. Gabin, "Evolved multimedia broadcast/multicast service (embms) in lte-advanced: overview and rel-11 enhancements," *IEEE Communications Magazine*, vol. 50, no. 11, 2012.

[37] G.-X. Project, "Ptm," 2018. [Online]. Available: http://5g-xcast.eu/about

[38] Wikipedia, "Ip multicast," 2018. [Online]. Available: https://en.wikipedia.org/wiki/IP_multicast

[39] ——, "Pragmatic general multicast," 2018. [Online]. Available: https://en.wikipedia.org/wiki/Pragmatic_General_Multicast

[40] S. K. Miller, S. K. Robertson, C. A. Tweedly, and S. M. White, "Starburst multicast file transfer protocol (mftp) specification," 1998. [Online]. Available: https://tools.ietf.org/html/draft-miller-mftp-spec-03

[41] Wikipedia, "Real-time transport protocol," 2018. [Online]. Available: https://en.wikipedia.org/wiki/Real-time_Transport_Protocol

[42] ——, "Resource reservation protocol," 2018. [Online]. Available: https://en.wikipedia.org/wiki/Resource_Reservation_Protocol

[43] OpenGL, "Opengl," 2018. [Online]. Available: https://www.opengl.org

[44] S. H. Ahn, "Opengl transformation," 2018. [Online]. Available: http://www.songho.ca/opengl/gl_transform.html

[45] ——, "Opengl projection matrix," 2018. [Online]. Available: http://www.songho.ca/opengl/gl_projectionmatrix.html

[46] C. Everitt, "Projective texture mapping," *White paper, NVidia Corporation*, vol. 4, 2001.

[47] Amazon, "Aws," 2018. [Online]. Available: https://aws.amazon.com

[48] M. Carbone and L. Rizzo, "Dummynet revisited," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 2, pp. 12–20, 2010.

[49] Amazon, "Amazon ec2 pricing," 2018. [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand

[50] A. Patait and E. Young, "High performance video encoding with nvidia gps," 2018. [Online]. Available: http://on-demand.gputechconf.com/gtc/2016/presentation/s6226-abhijit-patait-high-performance-video.pdf

[51] Oculus, "Performance head-up display," 2018. [Online]. Available: https://developer.oculus.com/documentation/pcsdk/latest/concepts/dg-hud

[52] F. Wong, "Performance analysis and optimization for pc-based vr applications: From the cpu's perspective," *Intel Virtual Reality Documentation*, 2016. [Online]. Available: https://software.intel.com/en-us/articles/performance-analysis-and-optimization-for-pc-based-vr-applications-from-the-cpu-s

[53] H. Bellini, "The real deal with virtual and augmented reality," 2016. [Online]. Available: http://www.goldmansachs.com/our-thinking/pages/virtual-and-augmented-reality.html

[54] Adobe, "Capitalizing on viewers' hunger for virtual and augmented reality white paper," 2016. [Online]. Available: https://www.creativeplanetnetwork.com/videoedge/362797

[55] O. Rift, "Oculus," 2018. [Online]. Available: https://www.oculus.com

[56] Sony, "Playstation vr," 2018. [Online]. Available: https://www.playstation.com/en-us/explore/playstation-vr/

[57] Samsung, "Samsung gear vr," 2018. [Online]. Available: https://www.samsung.com/us/mobile/virtual-reality/

[58] Z. Wang, L. Sun, C. Wu, W. Zhu, Q. Zhuang, and S. Yang, "A joint online transcoding and delivery approach for dynamic adaptive streaming," *IEEE Trans. Multimedia*, vol. 17, no. 6, pp. 867–879, 2015.

[59] G. Gao, H. Zhang, H. Hu, Y. Wen, J. Cai, C. Luo, and W. Zeng, "Optimizing quality of experience for adaptive bitrate streaming via viewer interest inference," *IEEE Trans. Multimedia*, vol. 20, no. 12, pp. 3399–3413, 2018.

[60] C. Li, L. Toni, J. Zou, H. Xiong, and P. Frossard, "Qoe-driven mobile edge caching placement for adaptive video streaming," *IEEE Trans. Multimedia*, vol. 20, no. 4, pp. 965–984, 2018.

[61] S. Almowuena, M. M. Rahman, C.-H. Hsu, A. A. Hassan, and M. Hefeeda, "Energy-aware and bandwidth-efficient hybrid video streaming over mobile networks," *IEEE Trans. Multimedia*, vol. 18, no. 1, pp. 102–115, 2016.

[62] J. Yang, E. Yang, Y. Ran, Y. Bi, and J. Wang, "Controllable multicast for adaptive scalable video streaming in software-defined networks," *IEEE Trans. Multimedia*, vol. 20, no. 5, pp. 1260–1274, 2018.

[63] X. Hou, S. Dey, J. Zhang, and M. Budagavi, "Predictive view generation to enable mobile 360-degree and vr experiences," in *Proc. VR/AR Network*, 2018, pp. 20–26.

[64] X. Liu, Q. Xiao, V. Gopalakrishnan, B. Han, F. Qian, and M. Varvello, "360 innovations for panoramic video streaming," in *Proc. HotNets*, 2017, pp. 50–56.

[65] S. Mangiante, G. Klas, A. Navon, Z. GuanHua, J. Ran, and M. D. Silva, "Vr is on the edge: How to deliver 360 videos in mobile networks," in *Proc. VR/AR Network*, 2017, pp. 30–35.

[66] V. R. Gaddam, M. Riegler, R. Eg, C. Griwodz, and P. Halvorsen, "Tiling in interactive panoramic video: Approaches and evaluation," *IEEE Trans. Multimedia*, vol. 18, no. 9, pp. 1819–1831, 2016.

[67] R. Ju, J. He, F. Sun, J. Li, F. Li, J. Zhu, and L. Han, "Ultra wide view based panoramic vr streaming," in *Proc. VR/AR Network*, 2017, pp. 19–23.

[68] X. Corbillon, G. Simon, A. Devlic, and J. Chakareski, "Viewport-adaptive navigable 360-degree video delivery," in *Proc. ICC*, 2017, pp. 1–7.

[69] X. Ge, L. Pan, Q. Li, G. Mao, and S. Tu, "Multipath cooperative communications networks for augmented and virtual reality transmission," *IEEE Trans. Multimedia*, vol. 19, no. 10, pp. 2345–2358, 2017.

[70] Y. Liu, J. Liu, A. Argyriou, and S. Ci, "Mec-assisted panoramic vr video streaming over millimeter wave mobile networks," *IEEE Trans. Multimedia*, 2018.

[71] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," in *Proc. NIPS*, 2015, pp. 1171–1179.

[72] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[73] Y. Amit and D. Geman, "Shape quantization and recognition with randomized trees," *Neural Comput.*, vol. 9, no. 7, pp. 1545–1588, 1997.

[74] T. Kim, Y. Yue, S. Taylor, and I. Matthews, "A decision tree framework for spatiotemporal sequence prediction," in *Proc. KDD*, 2015, pp. 577–586.

[75] C. Yan, Y. Zhang, J. Xu, F. Dai, L. Li, Q. Dai, and F. Wu, "A highly parallel framework for hevc coding unit partitioning tree decision on many-core processors," *IEEE Signal Process. Lett.*, vol. 21, no. 5, pp. 573–576, 2014.

[76] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[77] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[78] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social lstm: Human trajectory prediction in crowded spaces," in *Proc. CVPR*, 2016, pp. 961–971.

[79] J. Liu, A. Shahroudy, D. Xu, and G. Wang, "Spatio-temporal lstm with trust gates for 3d human action recognition," in *Proc. ECCV*, 2016, pp. 816–833.

[80] J. Bütepage, M. J. Black, D. Kragic, and H. Kjellström, "Deep representation learning for human motion prediction and classification," in *Proc. CVPR*, 2017.

[81] M. Längkvist, L. Karlsson, and A. Loutfi, "A review of unsupervised feature learning and deep learning for time-series modeling," *Pattern Recognit. Lett.*, vol. 42, pp. 11–24, 2014.

[82] A.-r. Mohamed, G. E. Dahl, and G. Hinton, "Acoustic modeling using deep belief networks," *IEEE Trans. Audio, Speech, Language Process.*, vol. 20, no. 1, pp. 14–22, 2012.

[83] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[84] F. Qian, L. Ji, B. Han, and V. Gopalakrishnan, "Optimizing 360 video delivery over cellular networks," in *Proc. AllThingsCellular*, 2016, pp. 1–6.

[85] 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. BigData*, 2016, pp. 1161–1170.

[86] S. M. LaValle, A. Yershova, M. Katsev, and M. Antonov, "Head tracking for the oculus rift," in *Proc. ICRA*, 2014, pp. 187–194.

[87] Y. Xu, Y. Dong, J. Wu, Z. Sun, Z. Shi, J. Yu, and S. Gao, "Gaze prediction in dynamic 360 immersive videos," in *Proc. CVPR*, 2018, pp. 5333–5342.

[88] C.-L. Fan, J. Lee, W.-C. Lo, C.-Y. Huang, K.-T. Chen, and C.-H. Hsu, "Fixation prediction for 360 video streaming in head-mounted virtual reality," in *Proc. NOSSDAV*, 2017, pp. 67–72.

[89] J. Chakareski, R. Aksu, X. Corbillon, G. Simon, and V. Swaminathan, "Viewport-driven rate-distortion optimized 360º video streaming," in *Proc. ICC*, 2018, pp. 1–7.

[90] Y. Ban, L. Xie, Z. Xu, X. Zhang, Z. Guo, and Y. Hu, "An optimal spatial-temporal smoothness approach for tile-based 360-degree video streaming," in *Proc. VCIP*, 2017, pp. 1–4.

[91] Samsung, "Samsung vr," 2018. [Online]. Available: https://samsungvr.com

[92] W. Bros., "Kong vr," 2018. [Online]. Available: https://samsungvr.com/view/AJXWuVU5E3Q

[93] Wikipedia, "Equirectangular map," 2018. [Online]. Available: https://www.wikiwand.com/en/Equirectangular_projection

[94] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proc. KDD*, 2016, pp. 785–794.

[95] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir *et al.*, "Wide & deep learning for recommender systems," in *Proc. DLRS*, 2016, pp. 7–10.

[96] Z. Ma, M. Xu, Y.-F. Ou, and Y. Wang, "Modeling of rate and perceptual quality of compressed video as functions of frame rate and quantization stepsize and its applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 5, pp. 671–682, 2012.

[97] W. Ding and B. Liu, "Rate control of mpeg video coding and recording by rate-quantization modeling," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, no. 1, pp. 12–20, 1996.

[98] Wikipedia, "Knapsack problem," 2018. [Online]. Available: https://www.wikiwand.com/en/Knapsack_problem

[99] Keras, "Keras," 2019. [Online]. Available: https://keras.io/

[100] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proc. NIPS*, 2007, pp. 153–160.

[101] S. Cost and S. Salzberg, "A weighted nearest neighbor algorithm for learning with symbolic features," *Machine Learning*, vol. 10, no. 1, pp. 57–78, 1993.

[102] X. Hou, Y. Lu, and S. Dey, "Novel hybrid-cast approach to reduce bandwidth and latency for cloud-based virtual space," *ACM Trans. Multimedia Comput., Commun. Appl.*, vol. 14, no. 3s, p. 58, 2018.

[103] Speedtest, "Speedtest.net," 2018. [Online]. Available: http://www.speedtest.net/

[104] S. Winkler, *Digital video quality: vision models and metrics*, 2005.

[105] X. Hou, J. Zhang, M. Budagavi, and S. Dey, "Head and body motion prediction to enable mobile vr experiences with low latency," in *Proc. Global Commun. Conf.*, 2019, pp. 1–7.

[106] K. Boos, D. Chu, and E. Cuervo, "Flashback: Immersive virtual reality on mobile devices via rendering memoization," in *Proc. Int. Conf. Mobile Syst., Appl., Services*, 2016, pp. 291–304.

[107] L. Liu, R. Zhong, W. Zhang, Y. Liu, J. Zhang, L. Zhang, and M. Gruteser, "Cutting the cord: Designing a high-quality untethered vr system with low latency remote rendering," in *Proc. Int. Conf. Mobile Syst., Appl., Services*, 2018, pp. 68–80.

[108] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, and N. Dai, "Furion: Engineering high-quality immersive virtual reality on today's mobile devices," in *Proc. Int. Conf. Mobile Comput. Netw.*, 2017, pp. 409–421.

[109] S. Shi, V. Gupta, M. Hwang, and R. Jana, "Mobile vr on edge cloud: a latency-driven design," in *Proc. Multimedia Syst. Conf.*, 2019, pp. 222–231.

[110] Y. Li and W. Gao, "Deltavr: Achieving high-performance mobile vr dynamics through pixel reuse," in *Proc. Int. Conf. Info. in Sensor Netw.*, 2019, pp. 13–24.

[111] X. Yun and E. R. Bachmann, "Design, implementation, and experimental results of a quaternion-based kalman filter for human body motion tracking," *IEEE Trans. on Robotics*, vol. 22, no. 6, pp. 1216–1227, 2006.

[112] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi, "Social gan: Socially acceptable trajectories with generative adversarial networks," in *Proc. Conf. Comput. Vision Pattern Recognit.*, 2018, pp. 2255–2264.

[113] J. Martinez, M. J. Black, and J. Romero, "On human motion prediction using recurrent neural networks," in *Proc. Conf. Comput. Vision Pattern Recognit.*, 2017, pp. 2891–2900.

[114] J. Butepage, M. J. Black, D. Kragic, and H. Kjellstrom, "Deep representation learning for human motion prediction and classification," in *Proc. Conf. Comput. Vision Pattern Recognit.*, 2017, pp. 6158–6166.

[115] C. Fan, J. Lee, W. Lo, C. Huang, K. Chen, and C.-H. Hsu, "Fixation prediction for 360 video streaming to head-mounted displays," *Proc. ACM SIGMM Workshop Netw. Operating Syst. Support Digit. Audio Video*, 2017.

[116] A. Kendall, Y. Gal, and R. Cipolla, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," in *Proc. Conf. Comput. Vision Pattern Recognit.*, 2018, pp. 7482–7491.

[117] L. Zhao, Q. Sun, J. Ye, F. Chen, C.-T. Lu, and N. Ramakrishnan, "Multi-task learning for spatio-temporal event forecasting," in *Proc. SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, 2015, pp. 1503–1512.

[118] B. van Amsterdam, M. J. Clarkson, and D. Stoyanov, "Multi-task recurrent neural network for surgical gesture recognition and progress prediction," *arXiv preprint arXiv:2003.04772*, 2020.

[119] Unity, "Virtual museum," 2019. [Online]. Available: https://assetstore.unity.com/packages/3d/environments/museum-117927

[120] ——, "Virtual rome," 2019. [Online]. Available: https://assetstore.unity.com/packages/3d/environments/landscapes/rome-fantasy-pack-ii-111712

[121] R. W. Schafer, "What is a savitzky-golay filter?[lecture notes]," *IEEE Signal Processing Magazine*, vol. 28, no. 4, pp. 111–117, 2011.

[122] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[123] R. Grosse, "Lecture 5: Multilayer perceptrons," 2019. [Online]. Available: https://www.cs.toronto.edu/~mren/teach/csc411_19s/lec/lec10_notes1.pdf

[124] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[125] Wikipedia, "Pythagoras theorem," 2020. [Online]. Available: https://en.wikipedia.org/wiki/Pythagorean_theorem

[126] HTC, "Htc vive wireless adaptor," 2019. [Online]. Available: https://www.vive.com/us/wireless-adapter/

[127] Valve, "Steamvr faq," 2019. [Online]. Available: https://support.steampowered.com/kb_article.php?ref=7770-WRUP-5951

[128] ——, "Steamvr sdk," 2018. [Online]. Available: https://valvesoftware.github.io/steamvr_unity_plugin/

[129] ——, "Openvr sdk," 2018. [Online]. Available: https://github.com/ValveSoftware/openvr/

[130] Matlab, "Convert rgb image or colormap to grayscale," 2019. [Online]. Available: https://www.mathworks.com/help/matlab/ref/rgb2gray.html/