國立清華大學電機資訊學院資訊系統與應用研究所
碩士論文
Institute of Information Systems and Applications
College of Electrical Engineering and Computer Science
National Tsing Hua University
Master Thesis

運用深度學習方法最佳化沉浸式影片編碼設定
Optimizing Immersive Video Coding Configurations Using Deep Learning Approaches

洪澤厚
Tse-Hou Hung

學號：108065534
Student ID:108065534

指導教授：徐正炘 博士
Advisor: Cheng-Hsin Hsu, Ph.D.

中華民國 110 年 十一 月
November, 2021

國立清華大學
資訊系統與應用研究所

碩士論文

運用深度學習方法最佳化沉浸式影片編碼設定

洪澤厚

109

致謝

碩士兩年多的生活是我人生中非常特別的一段經歷。感謝我的指導老師徐正炘教授，在研究、生涯規劃、英文等方面給我很多指導及意見，讓我受益良多。感謝許之凡學長，在我一開始做研究的時候教我許多知識，也包容我做實驗時的粗心大意。感謝實驗室的朋友及學長姊弟妹。感謝同期的吳丞浩、蔡旻翰、陳昭文，給我很多研究及生活上的建議，還有吃了很多好吃的火鍋跟羊肉爐之類的。特別感謝吳丞浩，剛進來的時候互相分享學到的東西，碩一的生活過的非常充實和快樂。感謝樊慶玲學姊，教我很多研究上的知識，也不厭其煩的回答我的各種問題。最後感謝我的家人及女朋友，在我低潮的時候給我支持。感謝以上所有人，讓碩士生活成為我人生最特別的一段時光。

Abstract

Immersive video streaming technologies improve Virtual Reality (VR) user experience by providing users with more intuitive ways to move in simulated worlds, e.g., with 6 Degree-of-Freedom (6DoF) interaction mode. A naive method to achieve 6DoF is deploying cameras in numerous different positions and orientations that may be required based on users' movements, which unfortunately is expensive, tedious, and inefficient. A better solution for realizing 6DoF interactions is to synthesize target views on-the-fly from a limited number of source views. While such view synthesis is enabled by the recent Test Model for Immersive Video (TMIV) codec, TMIV dictates manually-composed configurations, which cannot exercise the tradeoff among video quality, decoding time, and bandwidth consumption. In this thesis, we study the limitations of TMIV and solve its configuration optimization problem by searching for the optimal configuration in a huge configuration space. We first identify the critical parameters of the TMIV configurations. Then, we introduce two Neural Network (NN)-based algorithms from two heterogeneous aspects: (i) a Convolutional Neural Network (CNN) algorithm solving a regression problem and (ii) a Deep Reinforcement Learning (DRL) algorithm solving a decision making problem, respectively. We conduct both objective and subjective experiments to evaluate the CNN and DRL algorithms on two diverse datasets: a perspective and an equirectangular projection dataset. The objective evaluations revealed that both algorithms significantly outperformed the default configurations. In particular, with the perspective (equirectangular) projection dataset, the proposed algorithms only required 23% (95%) decoding time, streamed 23% (79%) of views, and improved the utility by 73% (6%) on average. The subjective evaluations confirm that the proposed algorithms consume fewer resources while achieving comparable Quality of Experience (QoE) than the default and the optimal TMIV configurations.

中文摘要

沉浸式視頻流技術通過為用戶提供更直觀的方式在模擬世界中移動，例如使用六自由度 (6DoF)互動模式，改善了虛擬現實 (VR) 用戶體驗。實現 6DoF 的一種簡單方法是根據用戶的移動在許多不同的位置和方向部署攝像頭，不幸的是，這既昂貴又繁瑣且效率低下。實現 6DoF 交互的更好解決方案是從有限數量的源視圖中即時合成目標視圖。雖然最近的沉浸式視頻測試模型 (TMIV) 編解碼器支持這種視圖合成，但 TMIV 需要手動選擇編碼配置，無法在視頻品質、解碼時間和頻寬消耗之間進行權衡。在本文中，我們研究了 TMIV 的局限性，並通過在巨大的搜尋空間中尋找最優配置來解決其配置優化問題。我們首先確定 TMIV 配置中的關鍵參數。然後，我們從兩個不同的方面介紹了兩種基於神經網絡的算法針對兩種問題：(i) 卷積神經網絡 (CNN) 算法解決回歸問題和 (ii) 深度强化學習 (DRL) 算法解決決策問題。我們進行了客觀和主觀實驗，以在兩個不同的數據集上評估 CNN 和 DRL 算法：透視和等距柱狀投影數據集。客觀評估表明，這兩種算法都顯著優於默認配置。對於透視（等距柱狀）投影數據集，所提出的算法平均只需要23%（95%）個解碼時間，傳送23%（79%）的視圖，並且將效用提高73%（6%）。主觀評估證實，與默認和最佳 TMIV 配置相比，所提出的算法消耗更少的資源，同時實現可比的體驗質量 (QoE)。

Contents

致謝 i

Abstract ii

中文摘要 iii

1 Introduction 1
- 1.1 Contribution 3
- 1.2 Limitation 4
- 1.3 Organization 4

2 Background 5
- 2.1 From 2D Video Toward 360° Video 5
- 2.2 3D Representation for Realizing 6DoF Interaction 6
- 2.3 View Synthesis 10
- 2.4 Recent Standard Activity 11

3 Related Work 13
- 3.1 Immersive Video Streaming 13
- 3.2 Machine Learning Algorithms for Optimizing Video Streaming 14
- 3.3 Machine Learning Algorithms for Optimizing Video Coding 15

4 Test Model of Immersive Video 16
- 4.1 Components 16
- 4.2 Workflow 18
- 4.3 Limitations 18

5 The Configuration Optimization Problem 21
- 5.1 Problem Statement 21
- 5.2 The Configuration Optimizer In the Immersive Video Codec 22

6 Machine-Learning-Based Configuration Optimizers 24
- 6.1 The input Preprocessing Procedure 25
- 6.2 Output Post-Processing Procedure 25
- 6.3 The Convolutional Neural Network (CNN) Algorithm 25
- 6.4 The Deep Reinforcement Learning (DRL) Algorithm 26
- 6.5 Training and Testing Datasets 29
- 6.6 Training Procedure 30

7 Objective Evaluations **32**
7.1 Experiment Setup 32
7.2 Qualitative Evaluations 33
7.3 Quantitative Evaluations 34
7.4 Robustness Evaluation Results 39

8 Subjective Evaluations **41**
8.1 Experiment Setup 41
8.2 Results and Analysis 42

9 Summary of Our Findings **46**

10 Use Case: Real Estate Virtual Tour **48**
10.1 Usage Scenario 48
10.2 System Overview 49
10.3 Data Collection from the Photo-Realistic Simulator 49
10.4 Experiment Setup 50
10.5 Results 52

11 Conclusion and Future Work **55**
11.1 Conclusion 55
11.2 Future Work 55

Bibliography **59**

List of Figures

1.1 Three VR interaction modes, where the spheres represent the virtual worlds. 1
1.2 A sample process of view synthesis: two source views (top and bottom) are used to synthesize a target view (middle). 2
2.1 Two light field video representations with: (a) 7- and (b) 4-dimensional functions. 7
2.2 The example objects of: (a) point cloud and (b) mesh. 8
4.1 Overview of the TMIV codec. 17
4.2 Operations of the view selector in TMIV [74]. 17
4.3 The numbers of source views and their corresponding encoding size. . . . 19
4.4 The decoding time and the video quality for synthesizing one target view with different configurations: (a) different numbers of source views and (b) different numbers of passes. 19
5.1 The high-level architecture of immersive video streaming systems. 23
6.1 The preprocessing procedure for generating the inputs of the CNN algorithm. 24
6.2 The network architecture of the CNN algorithm. The architecture comprises a convolutional and a fully-connected part. 26
6.3 The transition diagram of the configuration space. The circles and the triangles represent states and actions, respectively. 27
6.4 The proposed DRL algorithm: (a) the agent observes the response of the environment, integrates the observation, and takes an action based on the observation and (b) the network architecture of the agent. 28
6.5 Sample video frames from: (a) Classroom (ERP), (b) Museum (ERP), (c) Painter (PTP), and (d) Kitchen (PTP). 30
6.6 The training procedure of the DRL algorithm. We sampled the historical trajectories from memory to train the DRL agent. 31

7.1 The synthesized target views with the configurations generated by different algorithms: (a) Kitchen, (b) and (c) Hijack, and (d) Museum. 34
7.2 The objective results from the training set with ERP video sequences: (a) number of views, (b) video quality, (c) decoding time, and (d) optimal score. 35
7.3 The objective results from the testing set with ERP video sequences: (a) number of views, (b) video quality, (c) decoding time, and (d) optimal score. 36
7.4 The objective results from the training set with PTP video sequences: (a) number of views, (b) video quality, (c) decoding time, and (d) optimal score. 37
7.5 The objective results from the testing set with PTP video sequences: (a) number of views, (b) video quality, (c) decoding time, and (d) optimal score. 38
7.6 The optimal scores from PTP video sequences for various Q_J values: (a) training set without retraining, (b) testing set without retraining, (c) training set with retraining, and (d) testing set with retraining. 40

8.1 The environment setup for subjective evaluations. 42
8.2 The pairwise comparison matrix of the ERP video sequences. 43
8.3 The pairwise comparison matrices of: (a) experienced and (b) non-experienced subjects. 43
8.4 The pairwise comparison matrix of the PTP video sequences. 44

9.1 Usage scenarios of our proposed algorithms for: (a) ERP video and (b) PTP video sequences. 46

10.1 The usage scenario of a real-estate virtual tour. 48
10.2 System overview of our implementation. 49
10.3 Sample screen-shots from: (a) ArchVizInterior, (b) XoioBerlinFlat, (c) LightroomInteriorDayLight, (d) RealisticRendering, and (e) office. 51
10.4 The camera placement. 52
10.5 The sampling area of target views. The positions of target views are sampled from this area. The orientation of the target views is the same as the source view s camera. 52
10.6 The ground truth and synthesized target views with the configurations generated by different algorithms 52
10.7 The results from the training set with five real-estate scenes: (a) number of views, (b) video quality, (c) decoding time, and (d) optimal score. . . . 53
10.8 The results from the training set with five real-estate scenes: (a) number of views, (b) video quality, (c) decoding time, and (d) optimal score. . . . 54

11.1 The performance comparison of the default configuration and the redesigned DRL algorithm: (a) training set and (b) testing set. 58

List of Tables

2.1 Comparisons of 3D Representations 9

6.1 Selected MPEG Video Sequences 29
6.2 Sample Frames and Training/Testing Sets of Selected ERP Video Sequences 29

7.1 The Optimal Score from ERP Video Sequences for Various Q_J Values . . 39

8.1 Basic Statistics of The Subjective Evaluations with ERP Video Sequences 42
8.2 The Results from Bradley-Terry (Top) and Plackett-Luce (Bottom) Models with ERP Video Sequences 42
8.3 Basic Statistics of the Subjective Evaluations with PTP Video Sequences . 44
8.4 The Results from Bradley-Terry (Top) and Plackett-Luce (Bottom) Models for the PTP Video Sequences 44

9.1 Summary of Recommendation 46

11.1 The performance comparison of the default configuration and the redesigned CNN algorithm. 57

Chapter 1

Introduction

Virtual Reality (VR) technologies are thriving in various business sectors, including computer games, the tourism industry, real estate, and occupational training. The global VR market is predicted to reach 26.89 billion USD by 2022 at an annual growth rate of 54% [93]. As VR technologies are maturing, how to improve the Quality of Experience (QoE) of VR applications by delivering high viewing quality has become a key question for researchers and developers.

One critical factor affecting the QoE of VR technologies is the way in which users move in VR worlds. There are three main interaction modes: 3 Degree-of-Freedom (3DoF), 3DoF+, and 6DoF, which are illustrated in Fig. 1.1. 3DoF VR allows users to change their orientations in three dimensions, which are represented by yaw ψ, pitch θ, and roll ϕ. However, with 3DoF VR, when users change their positions, the views rendered on their displays are *not* affected. To overcome this limitation, 3DoF+ VR supports *limited* movements in three additional dimensions, represented by x, y, and z coordinates. Users can move their positions to a certain extent when sitting on fixed chairs. 6DoF VR further relaxes the restriction on movement. That is, users can not only change their orientations but also move in VR worlds freely. These interaction modes can be supported by various displays, of which 2D monitors and Head-Mounted-Displays (HMDs). Between them, 2D monitors are available virtually everywhere, while HMDs provide more immersive experience.

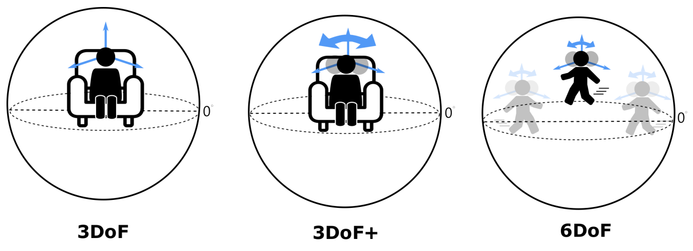


Figure 1.1: Three VR interaction modes, where the spheres represent the virtual worlds.

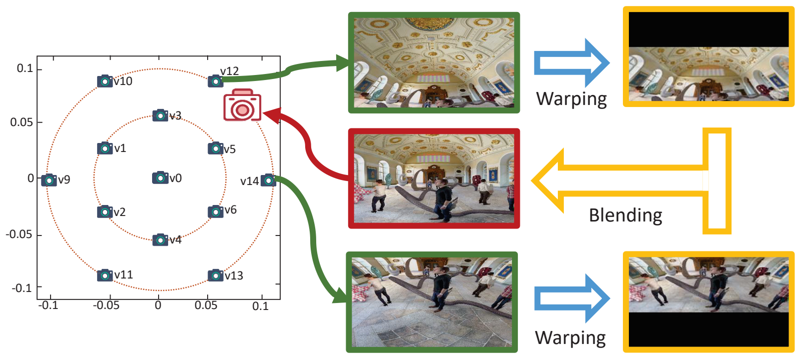


Figure 1.2: A sample process of view synthesis: two source views (top and bottom) are used to synthesize a target view (middle).

Today's VR content is mostly in the format of 360° videos, which are typically consumed with the 3DoF interaction mode. Moving toward 3DoF+ and 6DoF is no easy task. A naive way to achieve it is capturing multiple source views of natural scenes using cameras in many positions, or pre-generating multiple *source views* in virtual scenes using game engines. However, it is hard to include all possible positions of a scene. Even if we do, the data size is tremendous, leading to staggering demands on network bandwidth, storage space, and computational power. Moreover, in natural scenes, nearby cameras may be visible in source views, which require non-trivial post-processing for removal. Therefore, researchers and developers have recently been attracted to these open challenges to realizing 3DoF+ and 6DoF.

Recently, MPEG has released several standards for different 3D data representations to deliver 3DoF+ and 6DoF VR experience to users. Among the standards, Video-based Point Cloud Compression (V-PCC) [65] and MPEG Immersive Video (MIV) [66] are representative examples. V-PCC enables 3DoF+ and 6DoF VR experiences through compressing and delivering 3D *point clouds*. A point cloud consists of multiple points, where each point is described by the 3D coordinates and corresponding attributes. While V-PCC mainly focuses on compressing individual objects, MIV is designed for synthesizing *target views* in nature or virtual scenes, as illustrated in Fig. 1.2. Specifically, MIV takes multiple source views as the inputs, where each source view contains the texture, depth, and camera parameters. In this thesis, we study the Test Model for Immersive Video (TMIV) [74], which is a reference codec of MIV. The TMIV allows streaming systems to synthesize target views tailored for the diverse and dynamic positions and orientations of individuals.

Although the TMIV codec enables the 3DoF+ and 6DoF interaction modes, it does not take the tradeoff between the video quality and the number of source views into consideration. Specifically, the codec might transmit *excessive* source views to synthesize one target view, where the number of the source views is specified in the configurations. The default (pre-defined) configurations are inevitably not optimal for diverse and dynamic scenes and camera parameters, including orientations and positions. Thus, the systems might suffer from high bandwidth consumption and long computing time. One way to cope with this problem is to request a *subset* of source views to synthesize the target view, while maintaining the perceived quality of users. However, an ill-selected subset of source views may result in numerous *missing* pixels and thus degraded QoE, which could drive 3DoF+ and 6DoF VR users away.

In this thesis, we treat the above tradeoff of the TMIV codec as a *configuration optimization problem*. Our goal is to generate a configuration to maximize the perceived video quality, minimize the decoding time, and minimize the bandwidth consumption by transmitting an adequate number of source views to users. We study the optimization problem from two perspectives: as a regression problem and as a decision making problem. We then solve these two problems by two Neural Network (NN)-based algorithms: a Convolutional Neural Network (CNN)-based algorithm and a Deep Reinforcement Learning (DRL)-based algorithm, respectively. The CNN algorithm directly takes the ground-truth to train an inference model for predictions, while the DRL algorithm systematically trains an *agent* to adapt to dynamic *environments* [19, 45, 69, 79].

The two proposed algorithms are trained to adaptively generate the optimal configurations according to the video content, camera parameters, and user-specified utility functions. We conducted objective and subjective experiments to evaluate our algorithms on two datasets: an equirectangular projection dataset for HMDs and a perspective projection dataset for 2D monitors. The experiment results show that our proposed algorithms outperformed the default configurations provided by MPEG, which are the current practice. We also offer design recommendations for immersive video streaming systems based on our findings.

1.1 Contribution

This thesis makes the following contributions:

- We propose two NN-based algorithms[1] to solve the configuration optimization problem of the TMIV codec. Our approach can be readily adapted to other usage scenarios and immersive video codecs.

- We have conducted extensive experiments with real datasets to evaluate the performance of our algorithms. Our objective experiments show that the proposed algorithms consume fewer resources than the default TMIV configurations. With

[1]We use NN-based and NN algorithms interchangeably throughout the article.

the perspective (equirectangular) projection dataset, our proposed algorithms only require 23% (95%) decoding time, stream 23% (79%) of views, and improve the utility by 73% (6%) on average, compared to the default configurations.

- We conducted a user study to subjectively evaluate the quality of the synthesized view among different algorithms. A detailed analysis shows that the quality difference between our algorithms and the optimal configurations is insignificant.

1.2 Limitation

In this thesis, we use the current version of TMIV in our experiments. It may not be possible to apply our implementation directly in the future codec. However, our work shows the potential of optimizing the configuration of the 3D data representation codec by using machine-learning-based algorithms. Approaches similar to ours are also applicable to other current or future codecs that suffer from large configuration space.

1.3 Organization

The rest of this article is organized as follows. We survey the literature in Chapter 3. Chapter 4 presents the TMIV codec. This is followed by the core research problem described in Chapter 5. Chapter 6 details the designs of our proposed algorithms. We evaluate our algorithms using both objective and subjective metrics in Secs. 7 and 8. Chapter 9 summarizes our findings and discusses the sample usage scenarios of our solution. Moreover, we collect extra datasets by building a capturing system to further evaluate our algorithm in Chapter 10. Chapter 11 concludes the thesis.

Chapter 2

Background

In this chapter, we introduce the background of our research. We first describe the concepts from 2D video to 360° video. After that, various 3D representations that can be utilized to realize 6DoF interactions are introduced. Besides data representation, we also introduce the view synthesis technique, which is the essential component for realizing 6DoF interaction by using RGBD videos. Finally, we introduce recent standard activity related to 6DoF VR.

2.1 From 2D Video Toward 360° Video

2D video has been very common in our daily life for many years. Nowadays, video-related applications are everywhere, e.g., video streaming, conferencing, live streaming, and gaming. Recently, due to the maturation of the technique of 360° video, the application of 360° videos has started to come to the fore. It allows users to freely rotate their head to watch videos from different angles, providing a more immersive experience for users.

To apply 360° video to various video applications, there are several problems that need to be addressed. One of the critical problems is how to reduce the data size of 360° video, because 360° video contains more information than traditional 2D video. It is hard to transmit, store, or process 360° video if we do not compress it. To solve this problem, several approaches have been proposed [27, 87, 88]. The most common solution is to project 360° video into 2D video, utilizing a conventional video codec (e.g., h.264 and h.265) to compress it. There are several projection formats, e.g., EquiRectangular Projection (ERP), CubeMap Projection (CMP), and pyramid projection. The projection format affects the performance of the compression and the viewing experience. For instance, CMP format decreases the data size by 25% compared to ERP format. However, CMP format provides limited Field-of-View (FoV) for the user. Different projection formats may be chosen according to the scenarios. Another solution focuses on utilizing information in 360° video to compress it [33, 54, 83, 84]. Specifically, these works calculate motion vectors on the sphere rather than on the projected 2D plane. In this way, the size and position of objects on the sphere can be considered during the compression. Li et al. [54] implemented their approach in an HEVC reference codec to conduct experiments. The results showed that their approaches achieved a better compression rate compared to the original HEVC reference codec.

Although 360 provides a more immersive experience compared to 2D video, it still cannot provide a *full* immersive experience. Specifically, 360° video only records the information in a single position, which means it can only can provide 3DoF interaction. Users can rotate their head to see different views, but cannot walk or move around to see views in other positions. More descriptive 3D representations are required to enable the truly immersive experience of 6DoF applications.

2.2 3D Representation for Realizing 6DoF Interaction

To achieve 6DoF interaction, 3D data representation is required to represent the information from the 3D world. There are various 3D data representations, which can be classified into: (i) video- and (ii) volumetric-based representations [85]. The video-based representation, e.g., multi-view RGBD videos and light field video, are videos captured by cameras from several positions and orientations. The volumetric-based representation, e.g., 3D point cloud and mesh, are captured by or generated from Lidar and RGBD cameras. It is composed of geometric components (points, polygons) and a texture component. In this section, we introduce four data representations: multi-view RGBD video, light field video, 3D point cloud, and mesh, and compare their pros and cons in various aspects.

RGBD Video.

RGBD videos refer to multiple RGB and depth videos captured in different camera positions with diverse orientations. RGBD videos can be used to support 6DoF video streaming as follows. For a virtual camera that does not fall on any real camera positions, clients may synthesize their viewports using the RGBD videos from nearby real cameras through the warping and blending pipeline. MPEG-I has developed the MPEG Immersive Video (MIV) standard for realizing 6DoF video streaming with RGBD videos. While capturing, compressing, streaming, and synthesizing RGBD videos are rather mature techniques with well-optimized standards and tools, RGBD videos still impose the following limitations.

- RGBD videos are vulnerable to occlusions. When an area of the viewport is blocked by obstacles in all captured RGBD videos, the resulting synthesized viewport will contain holes.

- RGBD video cameras need to be carefully placed to avoid too many holes, which dictates systematic placement algorithms. Besides, when holes appear, some error concealment, hole filling, or inpainting algorithms are required.
- RGBD videos may not work well with directional lights, especially when the cameras are sparse. In such cases, the synthesized viewports may suffer from artifacts due to the blending of multiple warped videos with inconsistent lighting conditions.

Light Field Videos

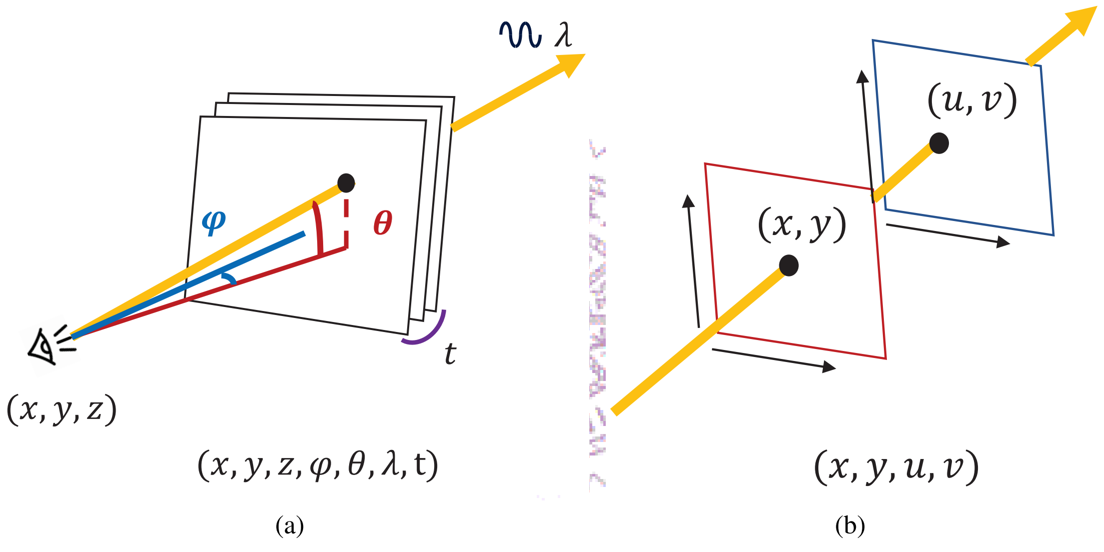


Figure 2.1: Two light field video representations with: (a) 7- and (b) 4-dimensional functions.

Light field videos aim to capture *all* the light rays at a scene, such as in an auditorium or a baseball park. The most comprehensive light field videos represent light rays using a 7-dimensional function $L(x, y, z, \phi, \theta, , \lambda, t)$ [2], where (x, y, z) represents the viewing position, (θ, ϕ) represents the viewing angles, λ represents the spectrum, and t represents the timestamp. Such light field videos may be an overkill for some applications that do not need light rays of: (i) invisible wavelengths or (ii) temporal-domain changes. Therefore, Marc et al. [55] propose a simplified 4-dimensional function $L(x, y, u, v)$ to represent light rays that go through the coordinates (x, y) on one plane and then (u, v) on another plane. Fig. 2.1 shows the difference between the two popular representations of light field videos. Recently, the MPEG-I group conducted various experiments to explore codecs for light field videos [61]. They considered an end-to-end system, which consisted of capturing, formatting, compressing, and displaying components. They also designed common test conditions for standardizing light field video codecs. Capturing light field videos is inherently difficult because we have to place cameras in too many positions. The captured setups can be classified into two classes: (i) structured and (ii) unstructured. Among them, the structured setups [6, 17] can be further classified into two subclasses: (i) camera array and (ii) microlens camera. Each camera array consists of dense cameras shooting at different orientations, whereas a compact microlens camera places a microlens array right in front of a regular image sensor to sense light rays from different directions. The unstructured setups [1] employ mobile cameras to capture content in many positions with diverse orientations, and then generate light field videos from the captured videos. Both structured and unstructured setups have their limitations. The structured setups suffer from higher hardware cost in building, installation, relocation, and maintenance. However, aggregating the captured videos into a light field video is easier considering the systematic layouts of the cameras. In contrast, the unstructured setups rely on good algorithms to generate high-quality light field videos.

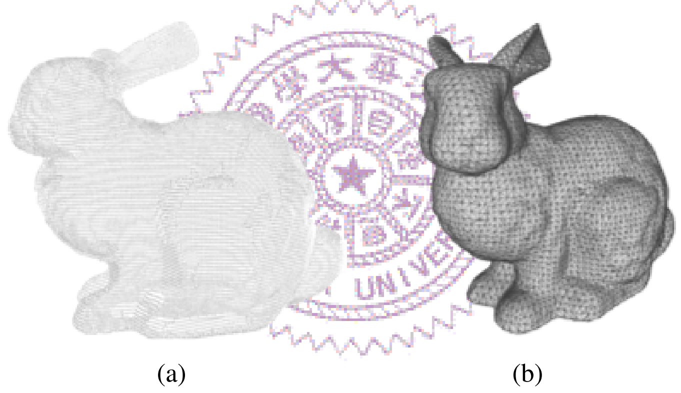


Figure 2.2: The example objects of: (a) point cloud and (b) mesh.

3D Point Cloud

Point cloud is a group of points in the 3D space. Each point consists of position (x, y, z) and attribute, e.g., color and reflection information. There are two ways to generate a point cloud. One way is captured by Lidar or other sensors. Another way is generated from multi-view RGB(D) images. Fig 2.2(a) shows an example of a point cloud object. Since a point cloud is composed of several independent points, it is easy to be manipulated or rendered. However, because a point cloud needs enough points to cover the surface of objects or scenes, the data size of a point cloud is tremendous. To solve this problem, more and more point cloud compression algorithms have been proposed. For instance, the MPEG-I group is about to finalize point cloud compression standards [63, 64].

3D Mesh

A mesh is a representation consisting of several points, edges, faces, and textures. Each point represents a position (x, y, z), and it connects to other points with edges. The point and edge compose the faces, and texture represents the color of the face. Fig. 2.2(b) shows an example of a mesh object. Meshes are widely used in various applications, e.g., 3D modeling software, animation, video games, and computer-aided design. Because of the popularity of meshes, modern GPUs have been designed to support fast and high-quality rendering of 3D meshes to support their applications. Although meshes are popular and mature, it is hard for them to represent natural scenes. The raw data from various sensors, e.g., RGB(D) camera and Lidar, need to be processed by several algorithms to become a mesh. These processes need a great deal of computing resources, which is hard to achieve in real-time. The noise from the sensor may also decrease the quality of the mesh.

Comparison

Table 2.1: Comparisons of 3D Representations

	Easy to Capture	**Large Data Size**	**Mature Codecs**	**Easy to Manipulate**	**High Rendering Quality**	**Fast Rendering Speed**
Most	RGBD	Light Field	RGBD	Point Cloud	Light Field	Light Field
Least	Light Field	Meshes	Light Field	Light Field	Point Cloud	RGBD

Table 2.1 qualitatively compares the 3D representations for 6DoF video streaming. We consider the following aspects roughly from the sender to receiver.

- **Easy to capture.** Light field videos dictate recording all light rays in the scenes, and thus are the most challenging to capture. RGBD videos can be captured by consumer-graded RGB(D) cameras, such as smartphone cameras, and thus are the least challenging to capture.

- **Large data size.** The largest representation is the light field video, while the smallest is the 3D mesh.

- **Mature codecs.** RGBD videos can be compressed using common video codecs, such as H.264, VP9, and those developed in the future. Therefore, RGBD videos may leverage the well-optimized video codecs implemented in software or hardware. Although the MPEG-I group has looked into 3D mesh and 3D point cloud codecs [62], they have not yet proposed any light field video codecs.

- **Easy to manipulate.** 3D point clouds contain no connectivities, and thus are easier to manipulate than 3D meshes. Light field videos are the most challenging to manipulate, because of the sheer amount of light rays that must remain mutually consistent after manipulation.

- **High rendering quality.** The representation that delivers the highest rendering quality is light field because of the extensive amount of light rays. On the other hand, rendered results from 3D point clouds are most vulnerable to holes and cracks.

- **Fast rendering speed.** Rendering viewports from light field videos only involve selecting the required light rays, without heavy computations, which can be done quickly. In contrast, different from 3D meshes and 3D point clouds, synthesizing viewports from RGBD videos is not currently supported by commodity hardware accelerators, and thus takes the longest time.

2.3 View Synthesis

View synthesis is a classic problem in the computer graphic area. Given single or multiple images or videos, how to render the other view which is not included in the input is an essential technique for realizing 6DoF interaction by using RGBD Videos.

The related research started in early 1990. Researchers started to study Image-Based Rendering (IBR) approaches to reduce the computation demand for rendering views from 3D models [12, 13, 57]. They proposed or utilized various IBR methods (e.g., image morphing, view interpolation, and light modeling) to render views from the captured image instead of the 3D model. In this way, the complex 3D model can be replaced by images, and the rendering time can also be independent of the scene complexity. Following these works, several IBR view synthesis approaches started to spring up [5, 11, 21, 35, 50, 53, 94]. These approaches improve on the original IBR approaches in different ways. For instance, Gortler et al. [35] proposed a new method to capture and represent 3D content. They improved on the original light modeling function by reducing the dimension. Zitnick et al. [94] proposed a method to synthesize a view in a dynamic scene. They utilized the view interpolation technique to synthesize high-quality views from multiple videos from different viewpoints. Besides IBR approaches, similar methods which leverage depth maps have been proposed to realize three dimensional television (3D TV) systems [14, 28, 68, 92]. This kind of approach is called Depth Image Based Rendering (DIBR). Comparing to IBR approaches, DIBR requires both color and depth images/videos to synthesize the view. It maps the pixels of multiple images/videos into 3D space by utilizing the depth value of each pixel, and projecting the pixel's value into the synthesized view. Then, the pixel values from different views are blended together to produce the final results. Note that some of the IBR approaches also utilize depth information to synthesize the view. However, IBR approaches usually estimate the depth information from input images/videos. DIBR assume that depth maps are provided. It can be captured from depth sensors by estimating from a depth estimation algorithm.

In recent years, machine learning techniques have started to be applied in view synthesis. The neural network model is used to replace the component of the traditional IBR or DIBR approaches. For instance, Kalantari et al. [49] used a Convolutional Neural Network (CNN) model to estimate depth and color information in the synthesis process. Hedman et al. [38] used CNN to determine the weight of pixel blending with the best output quality. Besides, the neural network model have also been used to convert the input to new data representations for view synthesis [26, 59, 90]. In these representations, Muti-Plane Images (MPI) is one of the most popular, which was proposed by Zhou et al. [90]. They trained a neural network model to convert input videos into MPI, which is composed of multiple RGBA layers with different depths. Once MPI is produced, it can be utilized to synthesize a range of views. Since MPI can produce high-quality synthesized results and does not need to retrain the model according to the scene, there are several works that designed more powerful view synthesis approaches based on MPI [4, 58, 82, 86]. These works focus on improving different aspects, e.g., rendering time, light reflection, practicality, and robustness.

2.4 Recent Standard Activity

Since VR is becoming increasingly popular, how to make content run on applications or devices from different manufacturers and companies has become a critical problem. MPEG started to focus on developing standards for VR in 2015. To date, they have defined several standards for various purposes [62].

The first VR standard proposed by MPEG is the Omnidirectional MediA Format (OMAF) [20, 37], which is a standard for 360° video. MPEG released the first version of OMAF in 2017, and the second version was released in 2020. For the first version, OMAF defines the coordinate system, projection, encoding, and encapsulation format. OMAF uses a right-hand coordinate system, and only includes rotation since it only considers a single 360° video. Two kinds of 360° video projection are considered in OMAF: (i) EquiRectangular Projection (ERP) and (ii) CubeMap projection (CMP). Besides the projection, OMAF also supports fisheye video. For the encoding and encapsulation format, OMAF defines its file format based on the ISO Base Media File Format (ISOBMFF) and Dynamic Adaptive Streaming over HTTP (DASH), which allows it to support viewport-dependent streaming and tile streaming to stream 360° video more efficiently. OMAF also defines three video profiles to provide different encoding settings for different streaming modes and codecs. The second version of OMAF considers multiple 360° videos. It not only defines the rotation of the coordinate system, but also defines position coordinates for several viewpoints. In this way, OMAF can support user switching between various 360° videos to see what they are interested in.

Besides OMAF, MPEG has also developed other standards for 3D data representations in order to support the immersive application for 6DoF interaction. These standards mainly focus on two data representations: 3D point cloud, and RGBD videos. For point cloud, MPEG defines Video-based Point Cloud Compression (V-PCC) and Geometry-based Point Cloud Compression (G-PCC) [36]. V-PCC and G-PCC utilize different approaches to compress a point cloud. V-PCC employs the projection technique to project a 3D point cloud to 2D video, and compresses 2D video by using a conventional video codec, e.g., H.264 and H.265. On the other hand, G-PCC compresses the point cloud by using a geometric manner, such as voxelization and octree coding, to encode the geometric information. It also uses several hierarchical prediction algorithms to encode the attributes of the point cloud. For RGBD videos, MPEG defined the MPEG Immersive Video (MIV) standard to compress multi-view RGB-D videos [9]. MIV removes the inter-view redundancy in RGB-D videos, and compresses the remaining area by employing a conventional video codec.

Chapter 3

Related Work

We first introduce representative immersive video streaming systems. This is followed by several NN algorithms for optimizing video streaming and coding.

3.1 Immersive Video Streaming

Prior studies on 3DoF+ and 6DoF VR can be classified into two groups: (i) discrete source cameras, where users can only jump to the predefined camera positions and (ii) continuous target (synthesized) cameras, where users can freely move within the virtual worlds. The former group is also referred to as *Multi-ViewPoint (MVP) 360° videos* in the literature. For example, Corbillon et al. [18] proposed an MVP 360° video streaming system, and conducted some experiments to compare different design choices and streaming strategies. Pang et al. [69] studied MVP 360° interactive video systems. They employed multi-modal learning and DRL to make streaming decisions for high visual quality and low response time.

The latter group is also referred to as 3DoF+/6DoF VR in research communities and standardization bodies. Huang et al. [44] proposed to support 6DoF VR using a single 360° camera through image-based warping. Hosseini et al. [40] designed a streaming framework to maximize the overall quality of streamed videos under limited bandwidth in 3D tele-immersion systems. They consider a human visual system when designing the adaptive streaming mechanisms to prioritize video streams. Ghosh et al. [34] designed a rate adaptation algorithm for 360° tiled video streaming. They formulated the problem as the QoE optimization problem under a given bandwidth limitation. Besides, MPEG has been developing the TMIV reference codec [73, 74, 75] as a 3DoF+ video codec. There are some immersive video streaming papers have adopted TMIV in their experiments. Jeong et al. [46] proposed a tiled-based method to optimize immersive video streaming. They employed TMIV as the 6DoF immersive video codec, and HEVC as the 2D codec to evaluate their tile selection algorithm. In our work, we solve the configuration optimization problem in the TMIV, which was not well studied previously.

Different view synthesis approaches using multiple source views have been presented in the literature [5, 13, 49, 67, 81, 91]. View synthesis is the crux to support continuous target cameras based on the positions and orientations of users. Our configuration optimization algorithms are also applicable to view synthesis approaches other than TMIV. Another key component affecting the view synthesis quality is the view selector, which selects a subset of source views for synthesizing target views. To the best of our knowledge, the existing view selection approaches are heuristic. For example, Dziembowski et al. [23] solved the problem based on source and target camera parameters. In contrast, our proposed algorithms consider the content of source views for optimal synthesized video quality.

3.2 Machine Learning Algorithms for Optimizing Video Streaming

Machine learning models have been applied to: (i) optimize video streaming over the Internet and (ii) determine video coding parameters. For video streaming optimization, the studies can be further classified into two groups. First, optimizing real-time interactive streaming over RTP and UDP has been studied [15, 45]. For instance, Huang et al. [45] proposed a rate control algorithm based on DRL. Their evaluation results revealed that the proposed algorithm achieves high video quality, low sending bitrate, and low latency. Second, optimizing on-demand video streaming over DASH (Dynamic Adaptive Streaming over HTTP) has also been considered [3, 16, 31, 39, 56]. For example, Chiariotti et al. [16] proposed an NN based Adaptive BitRate (ABR) algorithm [7] for DASH clients. They formulated the problem using the Markov Decision Process (MDP) and employed a Reinforcement Learning (RL) algorithm to solve it. Along the same lines, Gadaleta et al. [31] also proposed an RL-based DASH ABR algorithm using deep Q-learning.

A few recent works have extended the above studies to support 360° video streaming [30, 47, 69, 89], where videos are divided into tiles [27]. Pang et al. [69] employed multimodal learning that capitalizes on different feature types. They trained their model by using the video quality as the reward and the latency as the penalty. Fu et al. [30] also proposed an RL-based ABR algorithm for 360° tiled video streaming. Their evaluation results demonstrated that their approach improved the overall QoE. These NN algorithms focus on the adaptation to the available bandwidth, while our proposed algorithms compute the TMIV configurations.

3.3 Machine Learning Algorithms for Optimizing Video Coding

For selecting video coding parameters, NN algorithms have also been adopted. Costero et al. [19] and Hu et al. [43] took NN approaches to determine the quantization parameters of H.265/HEVC [77]. Particularly, Hu et al. [43] proposed an RL algorithm to determine the quantization parameters. They trained the agent using the Q-learning algorithm, while considering a wide spectrum of features. Costero et al. [19] applied an RL algorithm to adjust the quantization parameters in multiple-user scenarios. These studies [19, 42, 43] are similar to our configuration optimization problem, as they all strive to find the optimal coding parameters. However, none of them consider immersive video codecs, which often have a huge configuration space. To the best of our knowledge, this is the first study that focuses on the codec optimization problem for the immersive video codecs, particularly TMIV. Hence, we cannot compare our algorithms against others in the literature.

Chapter 4

Test Model of Immersive Video

In this chapter, we provide a high-level overview of the TMIV codec [74][1].

4.1 Components

TMIV has two main entities: the encoder and decoder, which are shown in Fig. 4.1. The TMIV encoder inputs multiple source views and outputs a bitstream composed of the compressed texture, depth, and camera parameters. The TMIV decoder inputs the bitstream and outputs a target view. The TMIV encoder contains the following components:

- **View optimizer** chooses one or multiple views from the source views as *basic views* based on the coverage of each view. All other source views are referred to as *additional views*.

- **The atlas constructor** analyzes the basic and additional views, and removes redundant (duplicated) regions from the additional views based on the basic views. It then packs the basic views and the remaining additional views into rectangular video frames, which are called *atlases*.

- **The video encoder and metadata composer** encode the atlas and the camera parameters of the source views into a bitstream for streaming. The camera parameters include the position and orientation of a given camera.

The TMIV decoder contains the following components:

- **The video decoder and metadata parser** receive the bitstream and then decode the atlas and source camera parameters.

[1]The version of TMIV we used in this thesis is TMIV v3.0.

- **The occupancy map generator** generates the *occupancy map* for each atlas. The occupancy map indicates the pixels that carry useful information. Then, the map is fed to the synthesizer.

- **The view selector** reconstructs the source views from the atlas and chooses views for synthesis according to geometric metadata described by camera parameters. Particularly, the selector gives the views that are closer to the target view higher priority. The source view selection follows a manually-composed configuration file.

- **The synthesizer** generates the target views using the source views selected by the view selector. Specifically, it generates multiple synthesized views in multiple *passes* (one view per pass) and merges the synthesized views at the end for better video quality.

- **The inpainter** fills up the pixels that have no information. It uses the neighboring pixels to interpolate/extrapolate the missing pixels.

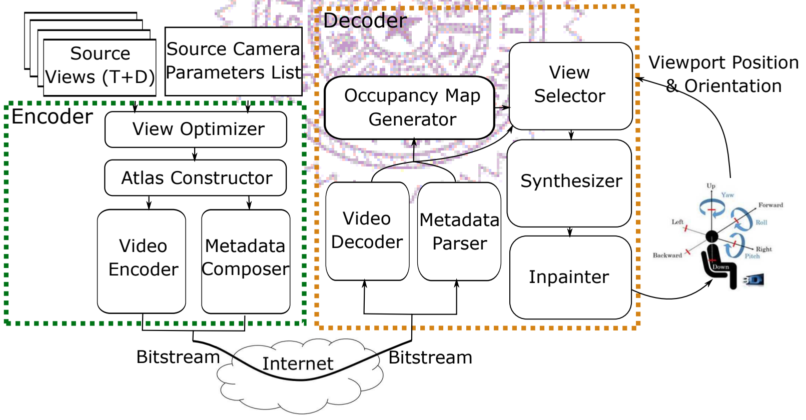
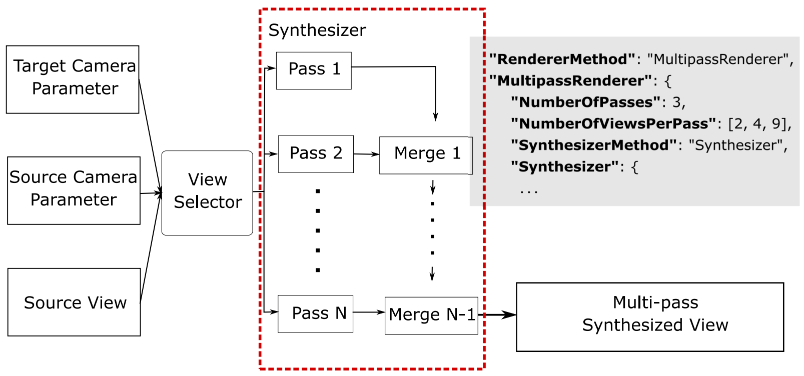


Figure 4.1: Overview of the TMIV codec.



```
"RendererMethod": "MultipassRenderer",
"MultipassRenderer": {
    "NumberOfPasses": 3,
    "NumberOfViewsPerPass": [2, 4, 9],
    "SynthesizerMethod": "Synthesizer",
    "Synthesizer": {
     ...
```

Figure 4.2: Operations of the view selector in TMIV [74].

We note that the above lists are by no means exhaustive. The TMIV codec contains other components for various functionalities, e.g., multiplexing multiple views into a single bitstream, which are not directly related to the codec configurations. These components are not discussed for brevity; interested readers are referred to the standard document [66, 74] for more details.

4.2 Workflow

TMIV can be invoked in two modes: (i) *MIV* and (ii) *MIV view*. The MIV mode runs all components in both the encoder and decoder. The MIV view mode only executes the components in the decoder, i.e., the atlas constructor does not remove the redundancy across source views before packing them. The MIV mode is more suitable when selecting the *codec configurations* or evaluating the performance of both the encoder and decoder. In contrast, the MIV view mode is more suitable when only evaluating the decoder. To evaluate the TMIV codec under realistic usage scenarios, we consider the MIV mode in this thesis. Our methodology can be applied to the MIV view mode as well.

Fig. 4.2 presents the operations of the view selector and the view synthesizer in the TMIV decoder, along with a sample configuration file. The view selector takes the parameters of the source and target cameras as inputs to calculate the geometric difference between any pair of source and target views. It then selects the views closest to the target views. After selecting a few source views for the current *pass*, the synthesizer synthesizes the target view from the selected views. This process is repeated for N passes, where N is specified by the *NumberOfPasses* value in the configuration file. The *NumberOfViewsPerPass* of pass n ($n = 1, 2, \dots, N$) is denoted by m_n, which is a positive integer. We note that the TMIV codec recommends having more views in the later passes, i.e., $r_{n'} > m_n$, $\forall n' > n$. In this way, the pixels with no information in earlier passes *may* be filled up by some later passes [71]. After finishing all passes, the view synthesizer merges the synthesized views to obtain the final target view.

4.3 Limitations

We conducted several pilot tests and identified a few key limitations of the TMIV codec [74, 74, 75]. We adopted Museum [48] as the test sequence, which consists of 24 source views at 2048×2048 resolution. We selected seven source views to synthesize another view following the default TMIV parameters. Particularly, we synthesized a random video frame using different configurations. The reconstructed video frame was compared against the ground truth to get the video quality in WS-PSNR (Weighted-to-Spherically-Uniform

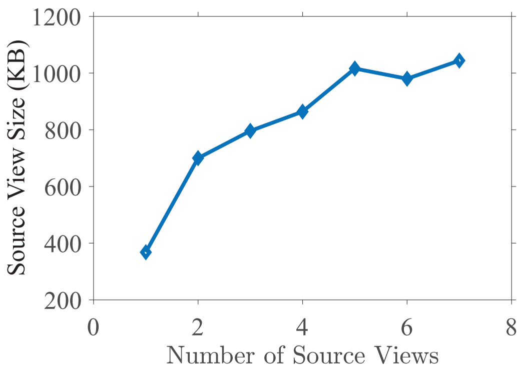
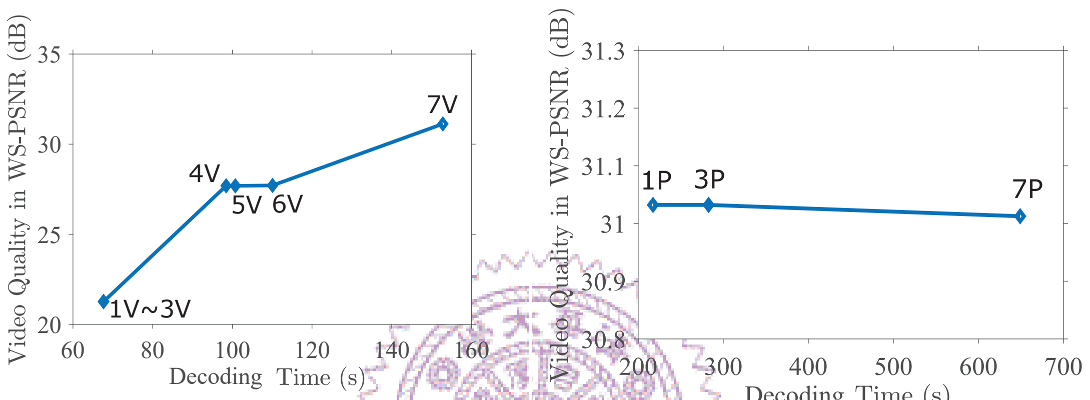


Figure 4.3: The numbers of source views and their corresponding encoding size.



(a) (b)

Figure 4.4: The decoding time and the video quality for synthesizing one target view with different configurations: (a) different numbers of source views and (b) different numbers of passes.

PSNR) [78][2], which quantifies the distortion of the reconstructed video in the spherical domain. We also recorded the decoding time[3] on an Intel i9 workstation at 3.5 GHz and the per-frame source view size, which is the sum of the size of all selected source views. We report the sample results from view #21.

We conducted two experiments. First, we studied the implications of different numbers of source views. We set seven one-pass configuration files, which included $1, 2, \ldots,$ and 7 source views, respectively. Fig. 4.3 reveals the per-frame source view size encoded by the H.265/HEVC codec [77] with the default coding parameters. This result shows that the source view size grows along with the number of source views[4]. Therefore, the

[2]We note that alternative quality metrics can be found in the literature, such as MPEG Comment Test Condition [48], which can also be adopted by our methodology. That is, researchers and engineers who prefer to use another quality metric or a weighted sum of multiple quality metrics are free to do so, because our purpose algorithms are general and can work with different quality metrics.

[3]The decoding time reported in this thesis is the CPU time.

[4]When the number of views is increased from 5 to 6, one more basic view with higher coverage of pixels is selected. Hence, more redundant portions of source views are removed, which leads to a smaller source view size. This, however, does not affect the overall trend.

network bandwidth consumption increases when selecting more source views. In fact, *excessive* bandwidth consumption is observed when streaming 7 source views: a total size of 1.1 MB per frame, or a bitrate of 264 Mbps at 30 FPS (frame per second) based on paper-and-pencil calculations. We also plot the relation between video quality and decoding time with 1–7 source views (V) in Fig. 4.4(a). We observe two trends: (i) the video quality grows with more source views, and (ii) the video quality dramatically jumps from 3 to 4 views, and then from 6 to 7 views. The two jumps are due to the two basic views that are selected by the view selector. In particular, once the view selector selects more basic views, the video quality increases because the basic views cover more pixels and are transmitted in their entirety. In summary, these experiments reveal *the importance of carefully choosing the number of source views.*

Next, we study the implications of different numbers of *passes* (P). Particularly, we use seven source views to synthesize the target view and vary the number of passes among 1, 3 and 7. In the three-pass test, the first pass contains a source view and the second and third passes contain four and seven source views, respectively. In the seven-pass test, the first pass contains a source view and every pass adds one more source view. Fig. 4.4(b) reports the relationship between the video quality and decoding time. As the figure shows, the decoding time and the number of passes are positively correlated because the decoding time increases as the involved source views increase. However, increasing the number of passes without adding more source views does not significantly increase the video quality. These experiments reveal *the importance of carefully choosing the number of passes.*

We conclude that TMIV configuration files need to be intelligently prepared and dynamically adapted to strive for the best tradeoff among:

- **The video quality** of the synthesized target views, which highly depends on the number of the selected source views.
- **The decoding time**, which increases when more source views are selected for view synthesis.
- **The network bandwidth consumption**, which highly depends on the number of selected source views that are streamed to the decoder side.

We aimed to solve the above *configuration optimization* problem by introducing a new component, a *configuration optimizer*, to find the optimal configuration for the TMIV codec, as presented in the rest of this thesis.

Chapter 5

The Configuration Optimization Problem

In this chapter, we first define the configuration optimization problem. We then place the configuration optimizer algorithms in the big picture.

5.1 Problem Statement

Problem (Configuration Optimization). *We are given: (i) V source views, where each view v ($v = 1, 2, \ldots, V$) consists of texture (T_v) and depth (D_v) and (ii) V corresponding source camera parameters, where each camera of view v is described by position (P_v) and orientation (O_v). We consider users with diverse and changing target views described by a time-series of target camera parameters (P_T and O_T). Find the optimal f^* from all F possible TMIV configurations to maximize a user-defined utility function $U(\cdot)$. That is: $f^* = \arg\max_{f \in F} U(f)$, where the utility function $U(\cdot)$ may depend on several factors, including but not limited to video quality, decoding time, and bandwidth consumption.*

We strive to make the above problem as general as possible. First, we adopt a *configuration template* to specify the search space of all F possible configurations. The configuration template defines the adjustable parameters with feasible ranges. For a concrete discussion, we focus on two essential parameters of TMIV: (i) the number of passes N and (ii) the number of views per pass m_n, $n = 1, 2, \ldots, N$; other parameters can also be added to the configuration templates if needed. Second, the utility function $U(\cdot)$ can be defined for different usage scenarios. For instance, video quality may dominate the viewing experience in high-end systems[1], while longer decoding time could negatively affect

[1]High-end systems, such as virtual tours in museums, aim to maximize the video quality to provide a truly immersive experience for users.

the frame rate on consumer-grade systems[2]. Therefore, we decided to support arbitrary utility functions without assuming any mathematical properties. For concrete discussion, we consider a utility function that is a decreasing function of the decoding time and an increasing function of the video quality. Concretely, we write it as:

$$U(\cdot) = \frac{Q_T - Q_J}{Y_T}, \tag{5.1}$$

where Y_T is the decoding time, Q_T is the video quality of the synthesized target view, and Q_J is the requested video quality. In our experiments, we set Q_J to 20 dB in the video quality (WS-PSNR or PSNR) if not otherwise specified. Q_J is essentially the threshold, which can be derived by server approaches, such as Just-Noticeable-Difference (JND) quality [32] or some QoE methods [51]. We note that the decoding time is correlated with the number of source views, where more source views lead to more network traffic. Hence, our proposed utility function indirectly considers the transmission time and bandwidth consumption. Last, we emphasize that Eq. (5.1) is merely a sample utility function to facilitate our discussions. Other utility functions can be adopted for different usage scenarios, as our proposed algorithms do not rely on any specific properties of the utility function.

5.2 The Configuration Optimizer In the Immersive Video Codec

We illustrate the role of the configuration optimizer in the immersive video codec in Fig. 5.1. When the user's head rotations and movements are captured by the HMD sensors, the target camera parameters are estimated and sent to the server side. The proposed configuration optimizer dynamically computes the configurations based on the received target camera parameters and sends the optimal configurations to the (local) TMIV encoder and the (remote) TMIV decoder. Then, the TMIV encoder selectively sends the required source views to the TMIV decoder following the configurations, which conserve the network bandwidth compared to sending all source views (① in the figure). Based on the configuration, target camera parameters, and the selected source views, the TMIV decoder synthesizes the target view (②). Finally, the target view is rendered to the user (③) based on the optimal configuration. By doing so, the decoding time and the resulting target view are optimized.

Several system-level decisions are crucial in real usage scenarios, although they are

[2]Consumer-grade systems, such as commodity HMDs, aim to deliver sufficient frame rates for a basic viewing experience to users.

beyond the scope of this thesis. For example, the execution frequency of the configuration optimizer may affect the system performance. The choice of the frequency depends on the degrees of user movement and target usage scenarios. For example, a lower frequency may be adopted when the user's positions and orientations change slowly; otherwise, a higher frequency is preferred. We list some system-level design challenges in Chapter 11 as future tasks.

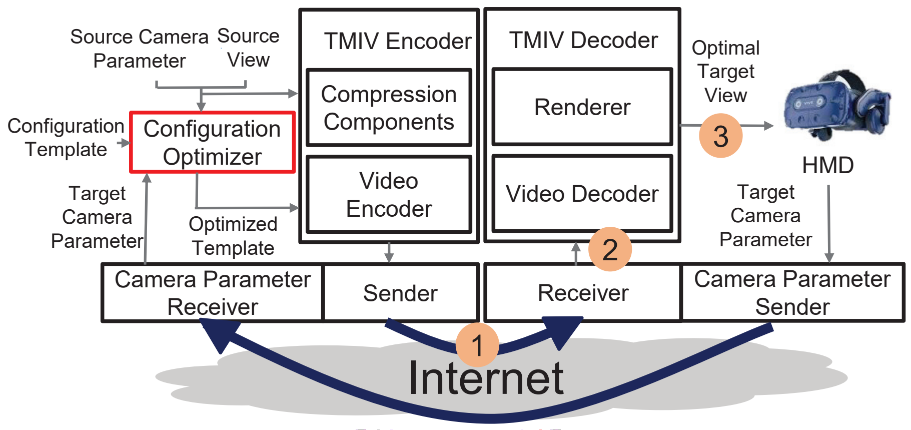


Figure 5.1: The high-level architecture of immersive video streaming systems.

Chapter 6

Machine-Learning-Based Configuration Optimizers

The goal of the configuration optimizer is to find the optimal configuration f^* to synthesize the target view with the optimal utility (e.g., Eq. (5.1)). However, solving the configuration optimization problem is not an easy task because of the potentially huge configuration space F which depends on the textures, depths, and camera parameters of the source views (T_V, D_V, and C_V, respectively) and the camera parameters of the target view (C_T).

In this chapter, we treat the configuration optimization problem as: (i) a regression problem and (ii) a decision making problem, and solve them using two representative NN algorithms: the CNN and DRL algorithms for predicting a configuration f approximating the optimal configuration f^*. Rather than directly predicting (or deciding) how many views are required in each pass, the proposed algorithms output the number of extra views required in each pass to ensure that the number of source views increases monotonically across passes.

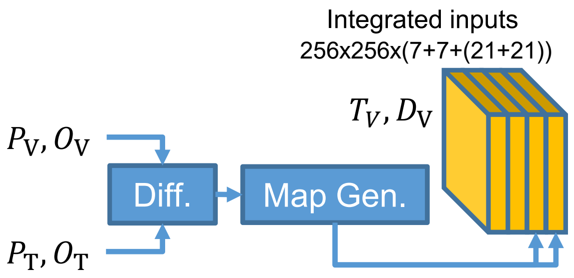


Figure 6.1: The preprocessing procedure for generating the inputs of the CNN algorithm.

6.1 The input Preprocessing Procedure

The input data of both NN algorithms are composed of T_V, D_V, C_V, and C_T. Because the source views' texture and depth resolutions (T_V and D_V) vary dramatically, we resized both T_V and D_V to the same resolution of 256×256 for consistency. Although the images might be slightly distorted because the ordinary resolution of the images is not square, the relative spatial information is still preserved. Moreover, we converted T_V into grayscale to reduce the input dimensions. We also tried to convert the depth value in D_V to real distance according to the MPEG document [22], but the performance of the resulting models was no better than the original depth values. We embed the camera parameters, C_V and C_T, in the inputs. We note that the camera parameters include camera positions P and orientations O. Firstly, we subtract C_T from C_V. Then, the results are fed into a map generator. The map generator duplicates the results for 256×256 times to form a two-dimensional map with a size of $256 \times 256 \times (21 + 21)$. Finally, we integrate the generated map with the textures and depths of the source views. Fig. 6.1 summarizes the preprocessing procedure for generating inputs. The resulting inputs contain: (i) two maps with $256 \times 256 \times 7$ resolution for the texture and depth, respectively; and (ii) two maps with $256 \times 256 \times 21$ resolution for the camera positions and orientations, respectively (3 dimensions for each source view).

6.2 Output Post-Processing Procedure

Assume the output of the model is a vector E, where $E = (e_1, e_2, ..., e_N)$ and N represents the number of passes. The required number of views for each pass $m_n (n = 1, ..., N)$ is calculated by

$$m_n = \begin{cases} 0, & e_n = 0; \\ 0, & r_j = 0 \text{ and } j < n; \\ \sum_{i=1}^{n} e_i, & \text{otherwise.} \end{cases} \tag{6.1}$$

We set $N = 3$ in our experiments if not otherwise specified, because optimal configuration f^* tends to use fewer than three passes in our pilot tests.

6.3 The Convolutional Neural Network (CNN) Algorithm

As the configuration f can be seen as a vector composed of positive integers, we can treat the configuration optimization problem as a regression problem. We then design a CNN regression algorithm to extract the features and predict the optimal configuration f^*.

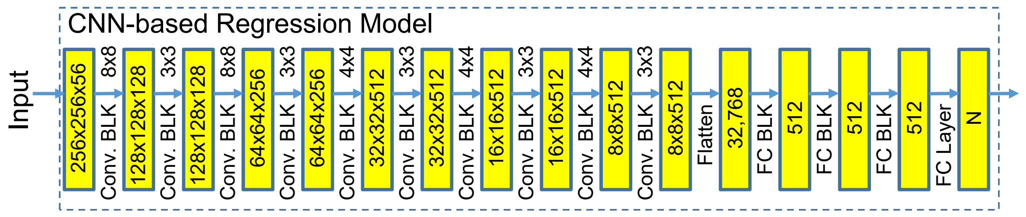


Figure 6.2: The network architecture of the CNN algorithm. The architecture comprises a convolutional and a fully-connected part.

Network Architecture. A CNN regression model is adopted to predict how many extra views should be added for each pass. The detailed network architecture is illustrated in Fig. 6.2. A convolutional network is used to extract features from the inputs, and a fully-connected network is used to infer the results from the extracted features. In the figure, the *Conv. BKL* represents a convolutional block that comprises a convolutional layer and a ReLU activation layer sequentially; a *FC BKL* represents a fully-connected block that is similar to the Conv. BKL, but the convolutional layer is replaced with a fully-connected layer. Every rectangle represents a hidden tensor, and the tensor size is given in the rectangle. We set the stride of the convolutional layers to 2 when the kernel size is 8×8 or 4×4 to integrate features without losing information; otherwise, we set the stride to 1. A sigmoid layer is added at the end of the model to ensure the model output is bounded. The number of parameters of our CNN model is 38,186,755. After rounding the output, the results are the required extra views in each pass, represented as an N dimensional vector.

We adopt Mean Squared Error (MSE) as the loss function to train the CNN regression model, which can be written as:

$$loss_{CNN} = \sum_{n=1}^{N} (e_n - e_n^*)^2, \tag{6.2}$$

where e_n^* is the number of extra views in the pass n, which can be obtained from f^*.

6.4 The Deep Reinforcement Learning (DRL) Algorithm

To treat the configuration optimization problem as a decision making problem, we start with the definition of the configuration space.

Configuration Space. The configuration space comprises configuration states and actions. Fig. 6.3 shows the transition diagram of the configuration space. In the figure, a circle represents a certain configuration state $s = (N, m_n, T_V, D_V, C_V, C_T)$, which corresponds to f in the configuration optimization problem. We omit T_V, D_V, C_V, and C_T

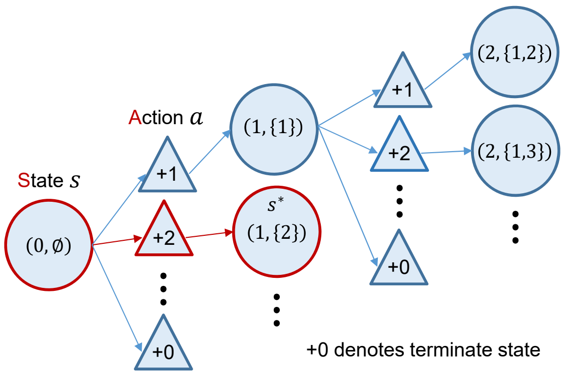


Figure 6.3: The transition diagram of the configuration space. The circles and the triangles represent states and actions, respectively.

in the figure for brevity. The first and second elements of the pair in each circle represent $N = |m_n|$ and m_n, respectively. The action of the configuration space is defined by the number of extra views to reach the next state, which is represented by a triangle in the figure.

Based on the configuration space, we can reformulate the configuration optimization problem into a search problem for the optimal state. We note that the optimal state can be any state in the space except the initial state $s_0 = (0, \emptyset)$. Once the utility value of each state is estimated, the optimal configuration for synthesizing the target view can readily be obtained. Unfortunately, calculating the utility value for every state is time-consuming, and thus is not practical for an immersive video streaming system. To solve this problem, we adopt DRL to efficiently approximate the utility value for every state. Specifically, a DRL agent starts at the initial state s_0 and chooses the most valuable action according to its observations to move towards the optimal state s^*.

Network Architecture. Fig. 6.4(a) illustrates our proposed DRL agent. The *observation* of the DRL agent comprises m_n, T_V, D_V, C_V, and C_T. Similar to the preprocessing procedure described in Sec. 6.1, we additionally extend m_n by the map generator and embed all maps by concatenation. The observations are then fed into the agent network. The agent network selects the best action based on the gained knowledge from the historical and current observations. Via the design of the states, the agent either continues to the next state or stops at the current state (choosing the +0 action). Our experiments end once the agent stops at a state and the corresponding vector m_n is the decision made by the DRL agent. The detailed architecture of the agent network is illustrated in Fig. 6.4(b). We intentionally keep the network architecture of the DRL agent similar to the NN algorithm to make a fair comparison.

Training the DRL Agent. We adopt Deep Q-learning (a.k.a. Deep Q-Network, DQN) to train the DRL agent so that the trained network can take the most valuable

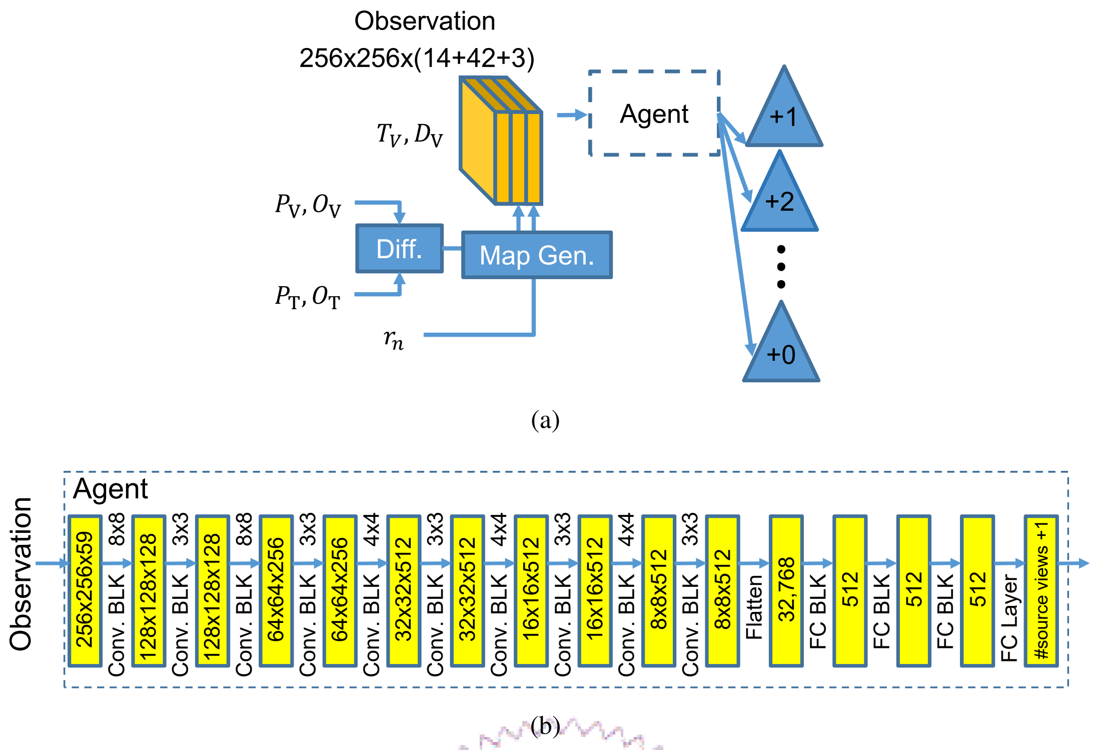


Figure 6.4: The proposed DRL algorithm: (a) the agent observes the response of the environment, integrates the observation, and takes an action based on the observation and (b) the network architecture of the agent.

action according to the observations. The DQN algorithm can be written as:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)], \tag{6.3}$$

where S_t, A_t are the sets of possible states and actions at step t, R_{t+1} is the immediate reward when performing action a, α is the learning rate, and γ is the discount factor. In our experiments, the immediate reward is set to be the difference of the utility values between a state and its immediate followed state, and the discount factor is set to one. Following the DQN approach, we employ two networks, namely the *prediction* and *target* networks to approximate the Q function. We adopt the following loss function:

$$loss_{DRL} = (r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i))^2, \tag{6.4}$$

to update the weights of the prediction and target networks. In this equation, $Q(s, a; w_i)$ is inferred by the prediction network and $Q(s', a'; w_i^-)$ is inferred by the target network. The target network is updated using the weights of the prediction network every m training iterations.

Table 6.1: Selected MPEG Video Sequences

Sequence	Projection	Resolution	Cameras	No. Frames
Classroom	ERP	4096x2048	24	300
Hijack	ERP	4096x4096	10	300
Museum	ERP	2048x2048	15	120
Kitchen	PTP	1920x1080	25	97
Painter	PTP	2048x1088	16	300
Frog	PTP	1920x1080	13	300
Fencing	PTP	1920x1080	10	250
Street	PTP	1920x1088	9	250
Carpark	PTP	1920x1088	9	250
Hall	PTP	1920x1088	9	500

Table 6.2: Sample Frames and Training/Testing Sets of Selected ERP Video Sequences

Sequence	Frame #	No. Views for Train.	No. Views for Test.
Classroom	50, 100, 150, 200, 250	3	5
Hijack	20, 40, 60, 80, 100	8	5
Museum	50, 100, 150, 200, 250	17	5

6.5 Training and Testing Datasets

We selected ten video sequences from MPEG [48] for the experiments, which are summarized in Table 6.1. Three of them are in Equirectangular Projection (ERP) [88] and the other seven are in PerspecTive Projection (PTP). Fig. 6.5 illustrates several sample video frames from the sequences.

To produce the training and testing sets, the following data processing procedure was applied. First, we randomly selected seven cameras from each video sequence to serve as the *source views*. Then, the TMIV encoder encoded the selected source views and the corresponding metadata into multiple altases. After that, we adopted the H.265/HEVC codec to compress/decompress the altases. For the compression process, we set QP to 30. Finally, the TMIV decoder decoded and rendered the target view with the position and orientation of a specific target view. We set the number of passes up to three and calculated the video quality of the target view, decoding time, and utility function values for all m_n combinations to form our training and testing sets. Note that we only took five sample frames at an equal frame interval from each video sequence for the experiments, because running all video frames through the TMIV reference software takes a prohibitively long time. Since we only had three ERP video sequences, we could not further divide them into the training and testing sets. Nevertheless, we employed different user positions and orientations in the training and testing sets for fair comparisons. In particular, we selected all the non-source-view cameras as the target views in the training set, and selected random camera parameters from MPEG [48] in the testing set. Table 6.2 summarizes the sampled frames and training/testing sets of the three ERP video sequences. We note that the orientations of the testing set in the Classroom and Hijack video sequences were set to zero to match the sample distributions of the camera orientations in the training dataset.

For the PTP video sequences, we used different video sequences, camera positions, and camera orientations in the training and testing sets. That is, we tested the performance of our algorithms on totally new, i.e., untrained, video sequences. We selected all the non-source-view cameras as the target view to generate data. To evaluate the ability of the proposed model for handling the untrained data, we adopted leave-one-out strategy to train and evaluate the proposed model. Specifically, we trained seven individual models. For each model, we assigned six video sequences as the training set and the remaining one as the testing set.

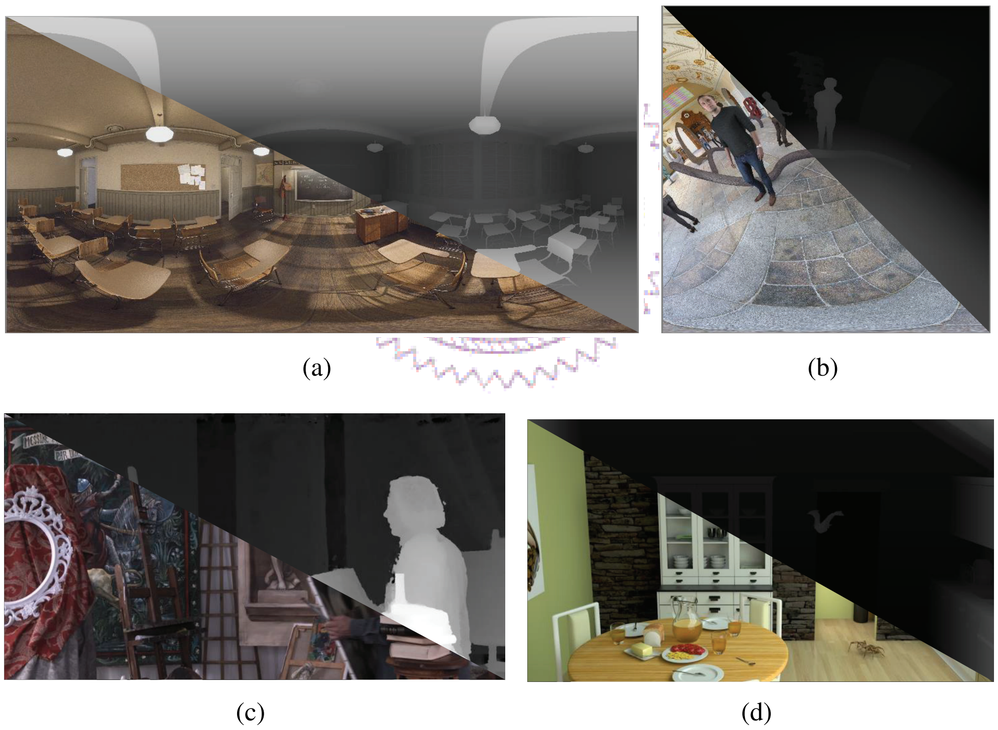


Figure 6.5: Sample video frames from: (a) Classroom (ERP), (b) Museum (ERP), (c) Painter (PTP), and (d) Kitchen (PTP).

6.6 Training Procedure

We trained the CNN regression model on a workstation with an Intel Xeon E5-2680 v3 CPU, 188 GB RAM, and four NVIDIA GeForce GTX 1080 Ti GPUs. The neural network was implemented in Python 3.7.6 and TensorFlow 1.14.0. We employed the Adam

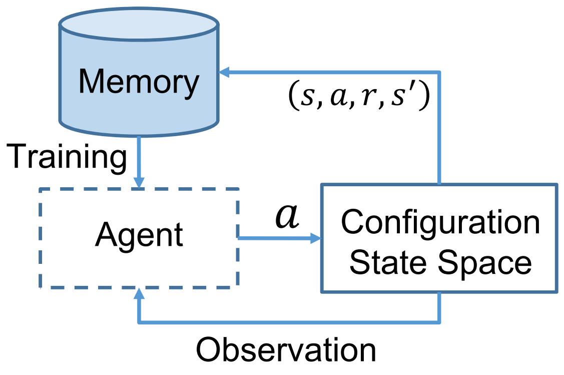


Figure 6.6: The training procedure of the DRL algorithm. We sampled the historical trajectories from memory to train the DRL agent.

optimizer [52] and set the learning rate at 10^{-5}. The CNN model took about three hours to converge.

We trained the DRL agent on a workstation with an Intel Core i7-6850K CPU, 64 GB RAM, and an NVIDIA GeForce GTX 1080 GPU. The networks were implemented in Python 3.5.2 and TensorFlow 1.4.0. Adam optimizer [52] was adopted to update the network, and the learning rate was set to 10^{-4}. Fig. 6.6 illustrates the training procedure to update the prediction and the target networks. The weights of both networks were randomly initialized. The agent uses the ϵ-greedy policy to balance the exploitation and exploration degrees. That is, in each step, the agent chooses an action a: (i) according to its observation or (ii) randomly. Then, the agent takes action a to reach the next state s'. The trajectory—consisting of the current state s, the selected action a, the immediate reward R, and the next state s'—of the agent is stored in memory for training the prediction network until $300,000$ steps (that is, $300,000$ trajectories) are stored. The immediate reward R is set to the utility difference between the state s and the state s'. In our experiments, the memory is a First-In-First-Out (FIFO) queue, and we set the queue size to 1,000. The ϵ of the ϵ-greedy policy was set to 0.5 for the first $100,000$ steps, 0.7 between $100,000$ and $200,000$ steps, and 0.9 at $200,000+$ steps to force the agent to explore the configuration space more aggressively when the training just started. We trained the prediction network once every ten steps after 1,000 steps. For each training round, a data batch with 32 trajectories was randomly sampled from the memory. The target network was updated once again when the prediction network was trained 500 times. The DRL model took about two days to converge.

Chapter 7

Objective Evaluations

In this chapter, we conduct objective experiments to evaluate the proposed configuration optimization algorithms.

7.1 Experiment Setup

We ran the experiments on a workstation with an Intel Core i9 CPU at 3.5 GHz and 32 GB RAM. We install the TMIV v3.0 [74] and H.265/HEVC [60] reference software on it. We implemented our proposed algorithms in Python to generate configurations for TMIV. We introduce two baselines for comparison:

- **Default (DEF).** The default configurations in TMIV [74] represent the *current practice*. If the default configuration requires more than seven views, we set the requested view to seven. Namely, we set $m_n = [2, 4, 7]$ in DEF for consistency of the experiments.

- **Optimal (OPT).** We found the optimal configuration (the highest utility) using exhaustive search as the *performance upper bound*.

We measured the following performance metrics:

- **Number of required views** to be streamed for synthesizing the target views, which directly affects the network bandwidth consumption.

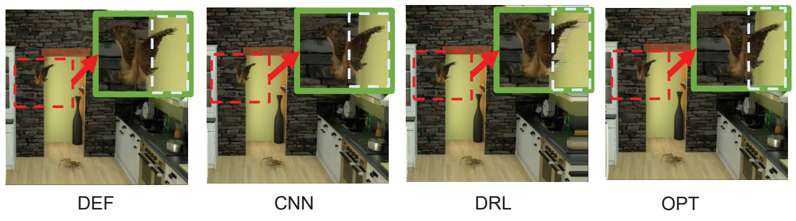
- **Video quality** in WS-PSNR [78] and PSNR for ERP and PTP videos, respectively. To calculate the WS-PSNR (or PSNR), for target views that fall on cameras, we took the source views from those cameras as the full-quality *references*. Otherwise, we synthesized the full-quality references using all cameras (as high as 24 cameras) as the input and added only one more camera in each pass (as high as 24 passes). Intuitively, this is the most time-consuming configuration that generates the synthesized target view with the best video quality.

- **Decoding time** of the TMIV decoder. The TMIV encoder may be executed in advance for on-demand service or on powerful servers for live services. Hence, we focus on the decoding time of the TMIV decoder, which imposes direct impacts on the latency and the frame rate for users. The TMIV reference software does not synthesize multiple passes in parallel, although the design permits that. To better understand the decoding time in properly-optimized TMIV software, such as Fleureau et al. [29], we assume all TMIV passes are *concurrently* executed and report the *maximal* decoding time across all passes as the decoding time.
- **Utility** of the resulting configuration. Eq. (5.1) indicates the balance between the video quality and the decoding time. In addition to absolute utility function values, we also define **optimal score** as the ratio of the utility achieved by an algorithm to that of OPT. The utilities of algorithms and OPT are shifted by a value which is calculated by the smaller value between the minimal utility in each experiment setup and zero.
- **Inference time** is the running time for generating a configuration based on proposed algorithms.

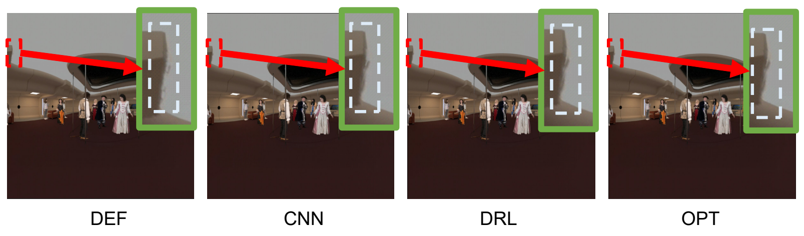
In each experiment, we compared four algorithms: DEF, CNN, DRL, and OPT. To get statistically meaningful results, we retrained the NN models in the CNN and DRL algorithms four times. We present the average results from these independently trained models with 95% confidence intervals whenever applicable. Moreover, to show the robustness of our algorithms, we conducted experiments to evaluate the performance under different Q_J values in the utility function.

7.2 Qualitative Evaluations

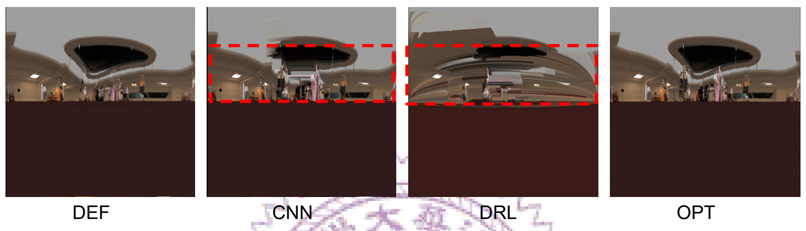
Fig. 7.1 shows several samples of synthesized target views from different algorithms. As shown in Figs. 7.1(a) and 7.1(b), most of the synthesized target views are visually similar to one another. There is only a tiny distortion in the result of the CNN and DRL algorithms. We highlight the distortion part with white rectangles in figures, which are still not obvious. The CNN and DRL algorithms sometimes generate noticeable distortion because the number of source views for synthesizing the target view is insufficient. We show some examples in the ERP dataset in Fig. 7.1(c) and 7.1(d). As the ERP video sequences cover a larger space than the PTP videos, their optimal configurations are more challenging to choose.



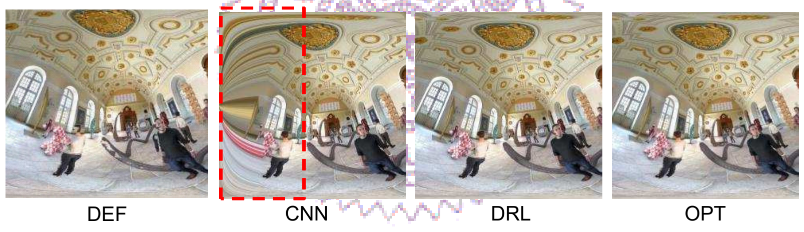
(a)



(b)



(c)



(d)

Figure 7.1: The synthesized target views with the configurations generated by different algorithms: (a) Kitchen, (b) and (c) Hijack, and (d) Museum.

7.3 Quantitative Evaluations

The ERP video sequences. We report the results from the ERP video sequences in Figs. 7.2 and 7.3. The results from the training set are shown in Fig. 7.2. In particular, Fig. 7.2(a) gives the average number of required views. The DEF algorithm requires more views to synthesize the target view, because the DEF algorithm opts for more source views to be conservative. Specifically, the default configurations require seven, five, and seven views for Classroom, Hijack, and Museum, respectively. The DRL algorithm requires fewer views than the DEF algorithm, because the agent intelligently stops at configuration states with the most valuable states. The CNN algorithm needs the least number of views, which however leads to some quality degradation, as shown in Fig. 7.2(b). The OPT algorithm requires slightly more views than the CNN algorithm. Nonetheless, OPT

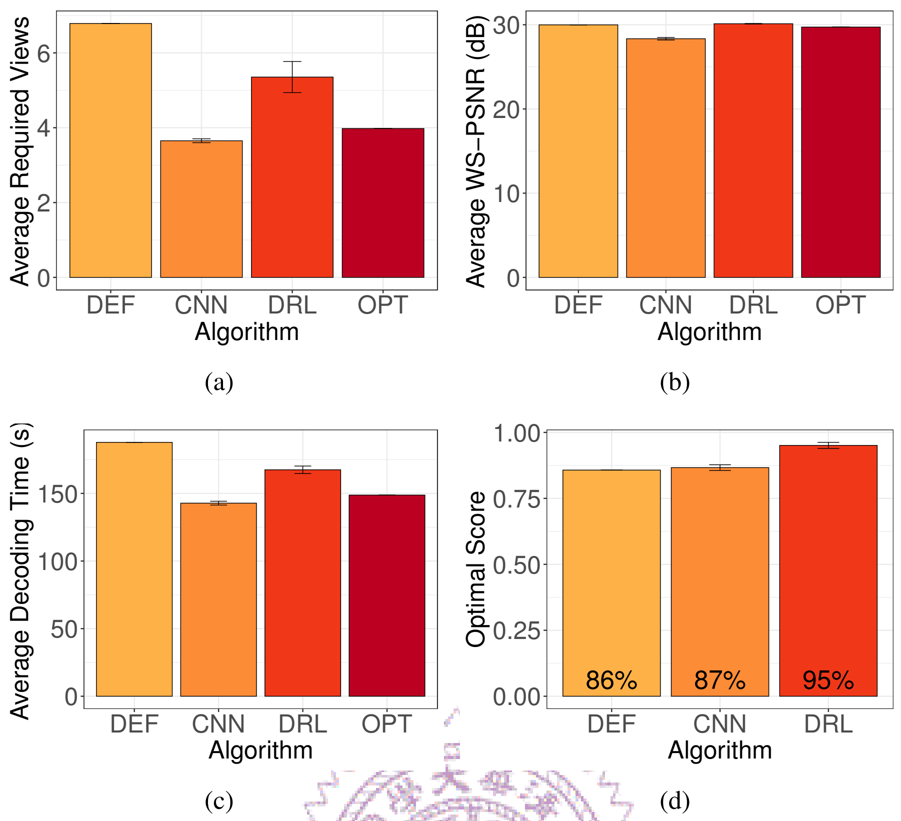


Figure 7.2: The objective results from the training set with ERP video sequences: (a) number of views, (b) video quality, (c) decoding time, and (d) optimal score.

checks the whole search space, which takes a prohibitively long time. Fig. 7.2(b) reports WS-PSNR values of the synthesized target views achieved by individual algorithms. This figure shows that all algorithms, except CNN, achieved similar synthesized video quality. Fig. 7.2(c) shows the decoding time for synthesizing the target views. We notice that the absolute values of the decoding time of the unoptimized TMIV reference software are less important. Therefore, we compare the relative difference among algorithms. We observe that the decoding time and the number of streamed views are in direct proportion because more source views means more computations. Next, we present the utility function in Fig. 7.2(d). The DRL algorithm achieves the highest optimal score, while the CNN algorithm also outperforms the DEF algorithm. *Overall, our NN algorithms significantly outperform the DEF algorithm in the training set and the DRL algorithm performs the best among all algorithms.*

Fig. 7.3 reports the results from the testing set. Fig. 7.3(a) shows the number of required views, which shows a similar trend as in the training set. Specifically, the DEF and CNN algorithms streamed the most and the least numbers of views, respectively. Our CNN algorithm led to small variance, as indicated by its confidence interval. However, the number of views required by the DRL algorithms were quite diverse. This may be

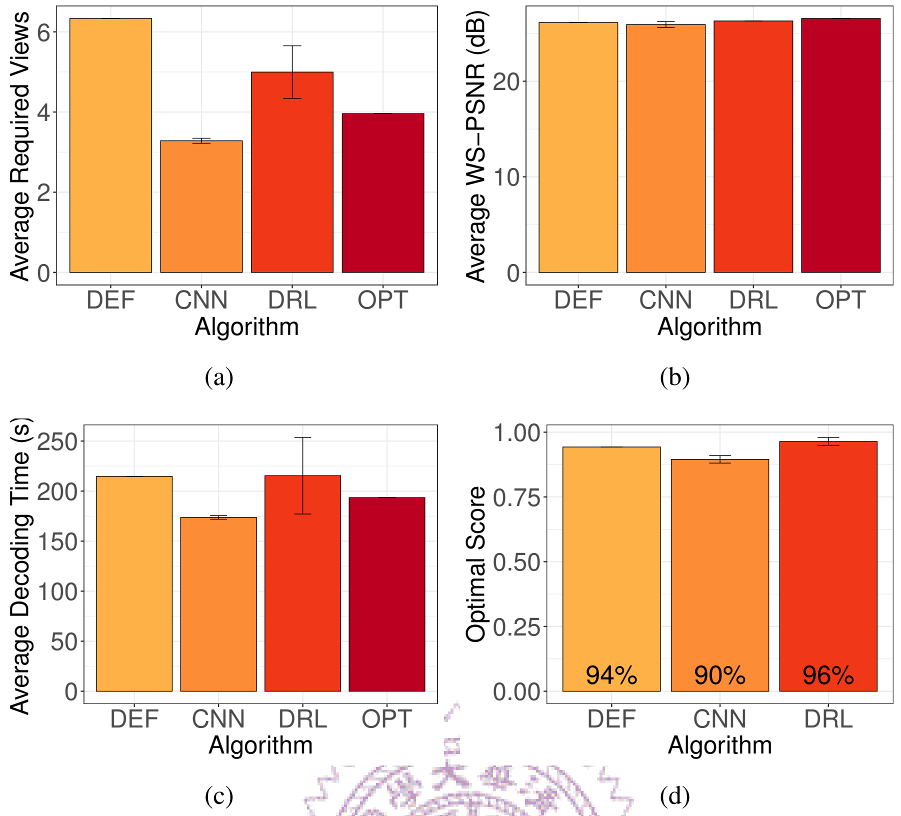
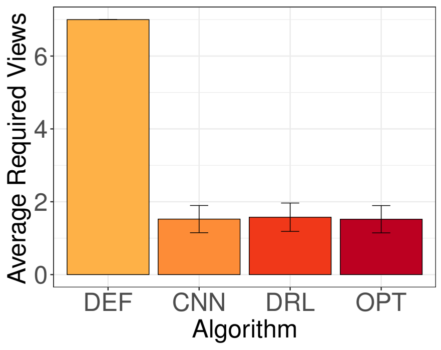


Figure 7.3: The objective results from the testing set with ERP video sequences: (a) number of views, (b) video quality, (c) decoding time, and (d) optimal score.

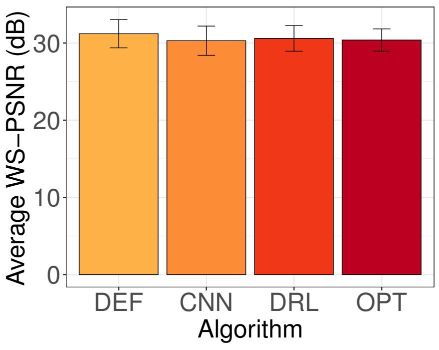
attributed to the fact that the DRL agent was more conservative while selecting the views. Fig. 7.3(b) gives the WS-PSNR values, which reveal that all the considered algorithms resulted in similar WS-PSNR. Compared to the training set in Fig. 7.2(b), the WS-PSNR values from the testing set were a few (3.3) dB lower on average, due to the dynamic camera parameters. We give the decoding time of view synthesizer in Fig. 7.3(c), which shows the same trend as in Fig. 7.2(c). Fig. 7.3(d) reports the optimal scores. The DRL algorithm still significantly outperformed the DEF and CNN algorithms. The CNN algorithm suffers from unstable video quality; therefore, its optimal score is slightly lower than that of the DEF algorithm. *Overall, the difference in video quality among the considered algorithms is insignificant. Our proposed CNN and DRL algorithms require fewer views than the DEF algorithm. The DRL algorithm has better performance than other algorithms.*

Summarizing the results, with the ERP video sequences, our DRL algorithm performed well for various user positions and orientations. On the other hand, the CNN algorithm resulted in inferior performance when facing new user positions and orientations.

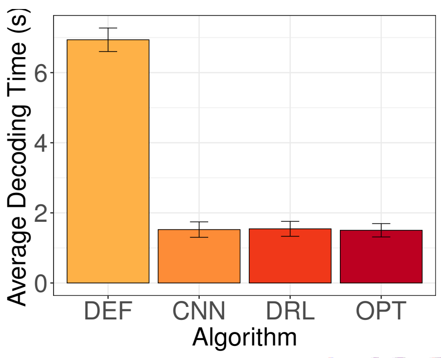
PTP video sequences. We report the results from the PTP video sequences in Figs. 7.4



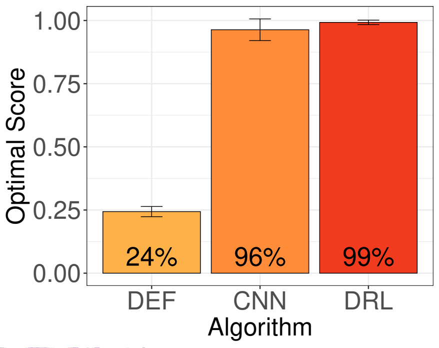
(a)



(b)



(c)



(d)

Figure 7.4: The objective results from the training set with PTP video sequences: (a) number of views, (b) video quality, (c) decoding time, and (d) optimal score.

and 7.5. The results from the training set are shown in Fig. 7.4. In particular, Fig. 7.4(a) shows the number of required views, where the DRL and CNN algorithms require fewer views than the DEF algorithm. They also performed well in video quality as shown in Fig. 7.4(b), comparable to that of the baselines. In Fig. 7.4(c), the DRL and CNN algorithms took less decoding time than the DEF algorithm. Last, Fig. 7.4(d) shows that the DRL and CNN algorithms achieved optimal scores of 99% and 96%, respectively. *Overall, we observe that the performance of the DRL and the CNN algorithm is very close to OPT.* Moreover, they significantly outperformed the DEF algorithm in the training set.

Fig. 7.5 reports the results from the testing set. Similar to Fig. 7.4, the DRL and CNN algorithms outperformed the DEF algorithm in all aspects. We note that the DRL algorithm led to a lower optimal score than the CNN algorithm, as shown in Fig. 7.5(d). In addition, the performance of the DRL algorithm had higher variance compared to the training set. We suspect that this is because the DRL agent cannot infer the state values well due to the insufficient explorations. As we know, the performance of RL algorithms is closely related to the exploration of the state space. Due to the diversity of video content and the limited number of video sequences, the DRL algorithm had difficulty predicting the accurate values of unseen states. Nevertheless, once the states were well

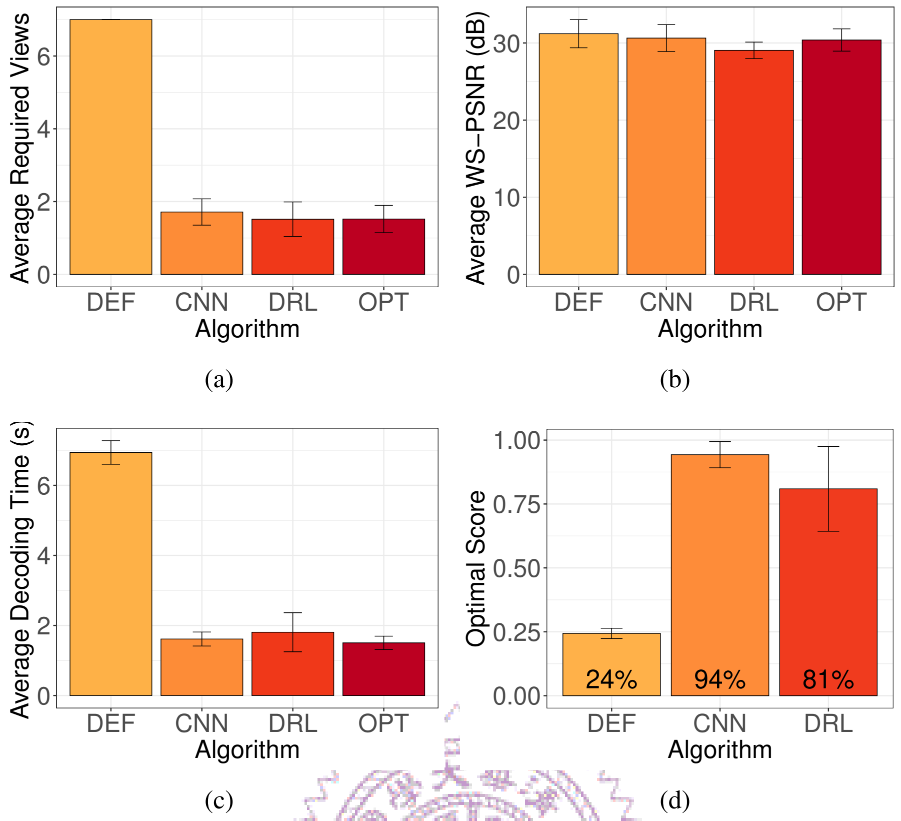


Figure 7.5: The objective results from the testing set with PTP video sequences: (a) number of views, (b) video quality, (c) decoding time, and (d) optimal score.

explored, the DRL algorithm could generate near-optimal configurations as we showed in the training datasets. *Overall, the CNN and DRL algorithms significantly outperformed the DEF algorithm, and the performance of DRL shows higher variance in the testing set.*

Summarizing the results, with the PTP video sequences, the CNN and DRL algorithms performed well for various video sequences and user positions and orientations. The CNN algorithm led to more stable performance with new video sequences.

Inference Time. For the CNN algorithm, the average and standard deviation of the inference time are 0.037 and 0.002 seconds, respectively. For the DRL algorithm, we recorded the time of taking actions to reach the optimal state. We computed the average and standard deviation of the inference time, which are 0.051 and 0.002 seconds, respectively. The difference is because the CNN algorithm directly generates the TMIV configurations, while the DRL algorithm takes at least two actions to obtain them. Nevertheless, the time to obtain the optimal configurations by our proposed algorithms is negligible, which showed their practicality.

7.4 Robustness Evaluation Results

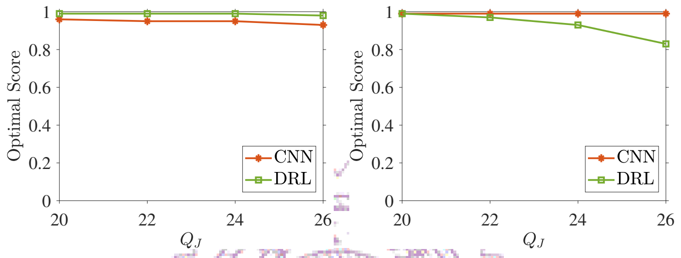
Robustness. To understand the robustness of our proposed algorithms, we first consider the performance under various Q_J values without retraining the models[1]. Specifically, we vary $Q_J \in \{20, 22, 24, 26\}$, while the original model was trained with $Q_J = 20$. We report the overall results from all three ERP video sequences and sample the PTP results from the PoznanStreet video sequence in the following. of the ERP and PTP video sequences, without retraining, are shown in Table 7.1 as well as Figs. 7.6(a) and 7.6(b), respectively. The upper half of the table and the figures reveal that our algorithms achieve similar optimal scores (always above 0.8) under different Q_J values in the training and testing sets. We next retrained our models with different Q_J values. The results from the retrained models are presented in Table 7.1 as well as Figs. 7.6(c) and 7.6(d). The bottom half of the table and the figures show that the performance of our algorithms is even better after being retrained with specific Q_J values. This is not surprising though. Summarizing the results, our algorithms work well with different parameter values even without retraining the model. The algorithms perform even better once the additional retraining process is done.

Table 7.1: The Optimal Score from ERP Video Sequences for Various Q_J Values

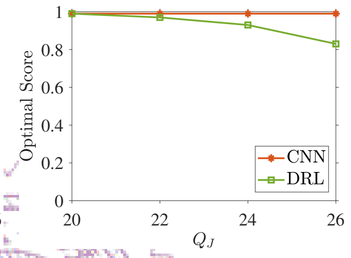
Q_J	**Without Retraining**			
	CNN		**DRL**	
	Training	**Testing**	**Training**	**Testing**
20	0.86	0.88	0.96	0.97
22	0.85	0.86	0.98	0.98
24	0.83	0.85	0.98	0.98
26	0.82	0.84	0.99	0.99

$\mathbf{Q_J}$	**With Retraining**			
	CNN		**DRL**	
	Training	**Testing**	**Training**	**Testing**
20	0.86	0.88	0.96	0.97
22	0.85	0.86	0.85	0.85
24	0.83	0.85	0.97	0.92
26	0.82	0.88	0.98	0.81

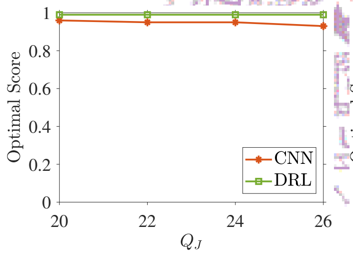
[1]Different from earlier experiments, where we report the average performance across four training sessions (see Sec 7.1), we report the results from the best training session here for meaningful comparison against the results without retraining.



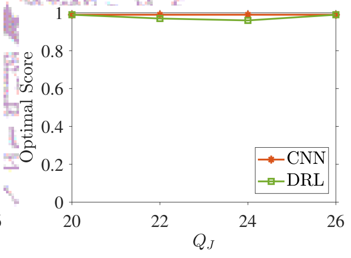
(a)



(b)



(c)



(d)

Figure 7.6: The optimal scores from PTP video sequences for various Q_J values: (a) training set without retraining, (b) testing set without retraining, (c) training set with retraining, and (d) testing set with retraining.

Chapter 8

Subjective Evaluations

In this chapter, we conduct subjective experiments to evaluate the synthesized target views resulting from individual algorithms.

8.1 Experiment Setup

Fig. 8.1 shows the environment setup for the subjective evaluations with the ERP video sequences. We asked each subject to sit on a swivel chair in her/his most comfortable sitting posture. The subject was allowed to freely move her/his upper-body, including her/his head, during the experiments. The synthesized target views were rendered to the subject via an HTC VIVE [41] tethered to an Intel i7 workstation running the Unity engine. We randomly selected six target camera parameters from each video sequence. Among them, three and three camera parameters were selected from the training and testing sets, respectively. Each experiment session included 18 experiment rounds, and the order of the experiment rounds was random. In each experiment round, the synthesized target views generated from the four algorithms (two NN-based algorithms and two baselines) were shown to the subjects in a random order. For the PTP video sequences, we asked each subject to observe synthesized target views on a 27” 2D monitor. We randomly chose four camera parameters from the testing set. Each experiment session included 28 experiment rounds. The PTP video sequences were the same and the order of the experiment rounds and target views were also random.

Subjects were asked to carefully observe the shown target views and rank the video quality of the views. They could switch among the four views until they were confident with their decisions on the ranking. Once the ranking among the four views was provided by each subject, it could not be changed. To avoid fatigue, subjects were allowed to take a short break after each round. All the events in the experiments were recorded. To

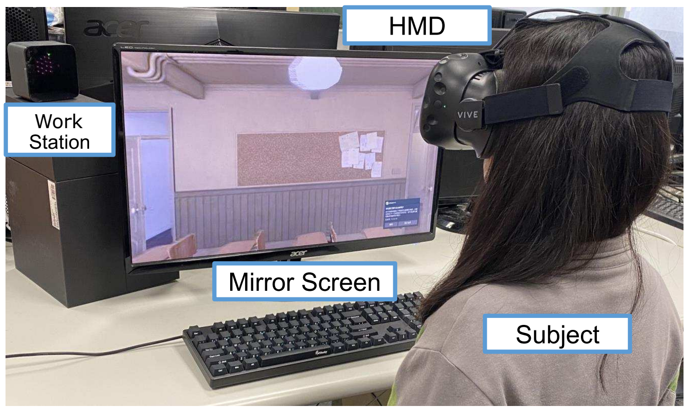


Figure 8.1: The environment setup for subjective evaluations.

Table 8.1: Basic Statistics of The Subjective Evaluations with ERP Video Sequences

	DEF	CNN	DRL	OPT
No. Observ. Round	2.7 (1.7)	3.1 (1.9)	3.1 (2.0)	3.3 (2.2)
No. Time Observ.	5.2 (3.0)	4.8 (3.1)	5.0 (3.0)	6.3 (5.5)

Table 8.2: The Results from Bradley-Terry (Top) and Plackett-Luce (Bottom) Models with ERP Video Sequences

	DEF	CNN	DRL	OPT
Coefficient	0.00	-0.10	-0.07	0.08
p-value	N/A	0.16	0.30	0.24
Coefficient	0.00	-0.07	-0.05	0.09
p-value	N/A	0.45	0.58	0.31

ensure reliability, if the average observation time of each target view was shorter than two seconds, that subject was considered as an outlier. In our evaluations, we found no outliers.

8.2 Results and Analysis

ERP video sequences. We recruited 23 subjects from college and graduate students, and 14 of whom were males. Only 11 subjects had previous HMD experience. All subjects had 20/20 corrected vision or no myopia. The average and standard deviation of their ages are 21.7 and 2.5 years, respectively. Table 8.1 presents the basic statistics. The numbers in this table denote the average values, and the numbers in the parentheses denote the standard deviations, respectively. On average, the subjects observe all target views about three times before ranking them. Each observation takes more than five seconds. Furthermore, subjects spend a similar amount of time on target views from different algorithms.

For detailed analysis, we first performed rank-breaking to separate a ranking into six pairwise comparisons. That is, if a subject ranked views as A>B>C, the ranking was separated into three pairwise comparisons, A>B, B>C, and A>C. Fig. 8.2 reports the pairwise comparison matrix. The values at row A and column B denote the proportion of the subjects that ranked the target view generated by algorithm A as having a higher

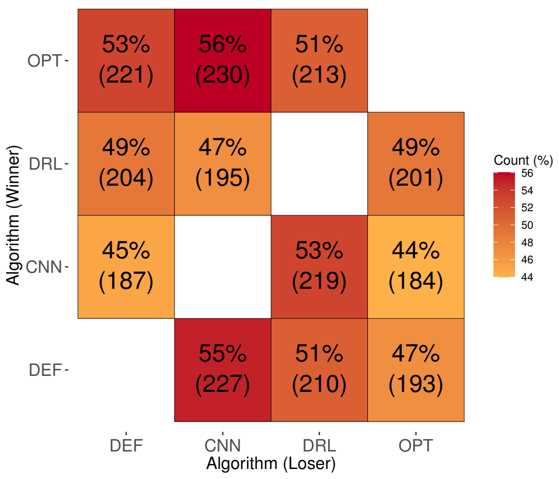


Figure 8.2: The pairwise comparison matrix of the ERP video sequences.

QoE than the view generated by algorithm B. Across all pairs, the proportions ranged between 44% and 56%. The pairwise comparisons show no obvious superior algorithm. Furthermore, we interviewed the subjects after they finished the questionnaires. They reported that the QoE of the four target views was hard to rank. The results validate that our proposed CNN and DRL algorithms achieve similar perceived video quality, while conserving network bandwidth and computational power as reported in Chapter 7.

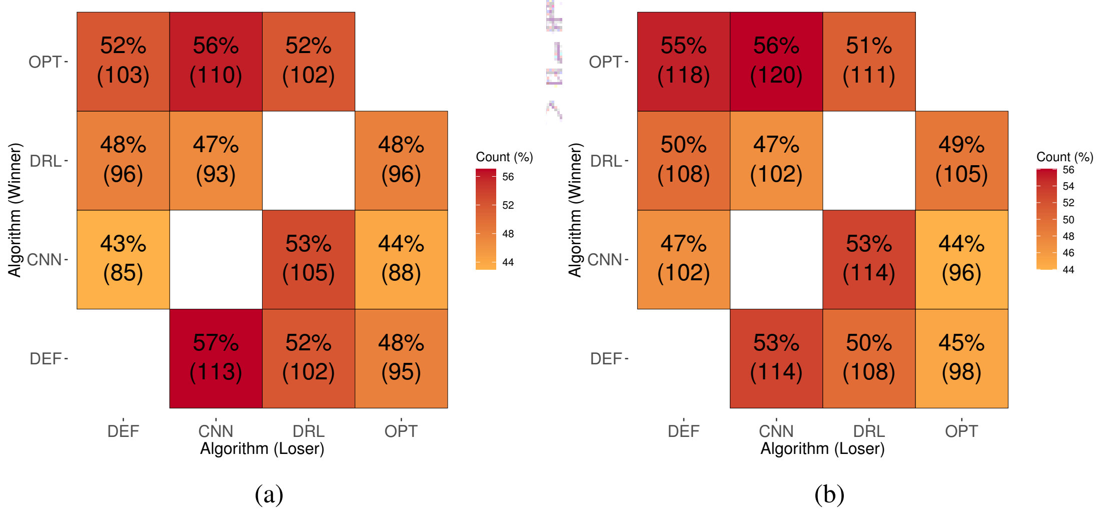


Figure 8.3: The pairwise comparison matrices of: (a) experienced and (b) non-experienced subjects.

Next, we modeled the pairwise comparisons and the ranking results using the Bradley-Terry [10] and Plackett-Luce [70] models. According to the coefficients of the modeled results, we explored the QoE among the investigated algorithms. Table 8.2 gives the modeled results of the Bradley-Terry and Plackett-Luce models. The estimated coefficients of both models ranged between -0.1 to 0.09, and the corresponding p-values of the coefficients were greater than 0.15. That is, there was no significant superior relationship among the investigated algorithms. We further explored the probability that each algorithm was ranked first by the Plackett-Luce model. The probability was 0.25, 0.24, 0.24, and 0.27 for the DEF, the CNN, the DRL, and the OPT algorithms, respectively. The results confirm that all the algorithms delivered similar QoE to subjects.

Last, we separated the subjects into two groups: experienced and non-experienced subjects. The pairwise comparison matrices are shown in Fig. 8.3. We also ran the results through the Bradley-Terry and Plackett-Luce models, and observed no significance among the investigated algorithms in any group.

Table 8.3: Basic Statistics of the Subjective Evaluations with PTP Video Sequences

	DEF	**CNN**	**DRL**	**OPT**
No. Observ. Round	11.8 (9.8)	13.9 (12.5)	12.5 (11.9)	14.5 (13.6)
No. Time Observ.	0.88 (0.44)	0.85 (0.47)	0.89 (0.55)	1.22 (1.37)

Table 8.4: The Results from Bradley-Terry (Top) and Plackett-Luce (Bottom) Models for the PTP Video Sequences

	DEF	**CNN**	**DRL**	**OPT**
Coefficient	0.00	-0.003	0.04	0.10
p-value	N/A	0.96	0.52	0.06
Coefficient	0.00	0.07	0.02	0.16
p-value	N/A	0.34	0.76	0.02

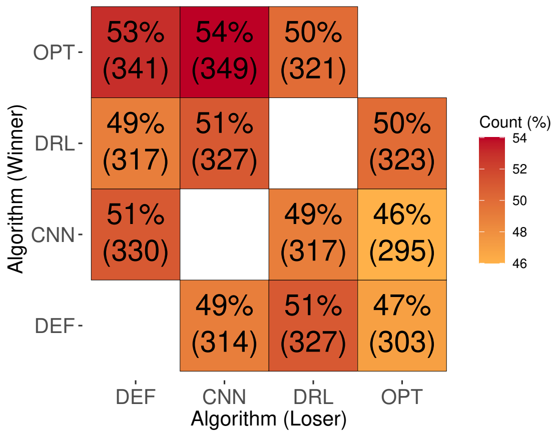


Figure 8.4: The pairwise comparison matrix of the PTP video sequences.

PTP video sequences. We recruited 23 subjects from college and graduate students, and 15 of whom were male. All subjects had 20/20 corrected vision or no myopia. The average and standard deviation of the subjects' ages are 22.13 and 2.12 years, respectively. Table 8.3 presents the basic statistics. On average, the subjects observed all target views more than ten times before ranking them. Each observation took about 1 second. This indicates that difference between the synthesized target views was not obvious, as subjects needed to switch among them for ranking. The same is true for the ERP video sequences; the subjects did not spend significant time on target views from any specific algorithm. Note that the observation and switch times for the PTP video sequences are higher than those for the ERP video sequences. This may be attributed to the fact that subjects could switch the video frames on the 2D display without rotating their heads.

Fig. 8.4 reports the pairwise comparison matrix. Across all pairs, the proportions ranged between 46% and 54%. The pairwise comparisons showed no obvious superior algorithm. The results for the PTP video sequences are the same, most subjects reflected that it was hard to rank the quality of the four target views because they were too similar.

Table 8.4 gives the modeled results of the Bradley-Terry and Plackett-Luce models. The estimated coefficient of both models ranges between -0.003 to 0.16, and the difference between the DEF, the CNN, and the DRL algorithms are insignificant. Although the OPT algorithm is statistically superior to the DEF algorithm, the difference in the quality of these two algorithms is minor and can be neglected. We further explored the probability that each algorithm was ranked first by the Plackett-Luce model. The probability is 0.23, 0.25, 0.24, and 0.28 for the DEF, the CNN, the DRL, and the OPT algorithms, respectively. The results confirm that all the algorithms delivered similar QoE to subjects.

Chapter 9

Summary of Our Findings

Table 9.1: Summary of Recommendation

Type	Dataset	Rec.	Compared to DEF			
			No. of Views	Video Quality	Decoding Time	Utility Value
ERP	Training (Seen Camera Parameters)	**DRL**	79%	100%	89%	9%
	Testing (New Camera Parameters)	**DRL**	79%	101%	100%	2%
PTP	Training (Seen Video Sequences and Camera Parameters)	**DRL**	23%	98%	22%	75%
	Testing (New Video Sequences and Camera Parameters)	**CNN**	24%	98%	23%	70%

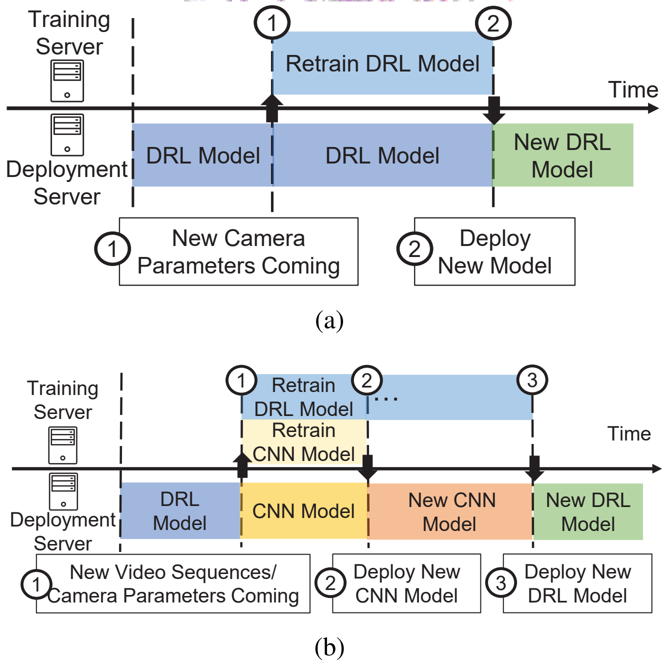


Figure 9.1: Usage scenarios of our proposed algorithms for: (a) ERP video and (b) PTP video sequences.

Our evaluations reveal that our algorithms require fewer source views and fewer computational resources to deliver comparable video quality compared to the baseline algorithms. Table 9.1 depicts our recommendation and performance improvement in different settings. We chose the recommended algorithms according to their utility values. For ERP video sequences, we suggest using DRL on both the training and testing sets. It achieves 9% and 2% improvement on the utility values compared with the DEF algorithm, respectively. For PTP video sequences, we suggest using DRL on the training set, and using CNN on the testing set. The DRL algorithm achieves 75% improvement on utility value, and the CNN algorithm achieves 70% improvement on utility value. Although our algorithms may lead to a minor drop in video quality, it can reduce the number of views and decoding time significantly.

Our proposed algorithms can be adopted in larger immersive video streaming services deployed in the future. Fig. 9.1 shows a sample scenario: an Over-The-Top (OTT) service for immersive videos. For ERP video sequences in Fig. 9.1(a), the service provider can train and deploy the DRL algorithm to reduce the bandwidth and computational resource consumption. For PTP video sequences in Fig. 9.1(b), the DRL algorithm can be used once all video sequences have been considered in the previous training phase. When new immersive video sequences arrive, the CNN model can be applied to achieve better performance on these new video sequences and camera parameters. At the same time, the CNN and DRL models start to be retrained on a training server. The new CNN and DRL models are deployed whenever they are ready. Note that the DRL model takes longer to be trained.

Chapter 10

Use Case: Real Estate Virtual Tour

Besides conducting experiments on datasets from MPEG, we also evaluated the performance of our algorithms in real applications. In this chapter, we apply our algorithms in the application of real estate virtual tours, and compare the performance of our algorithms and the baseline.

10.1 Usage Scenario

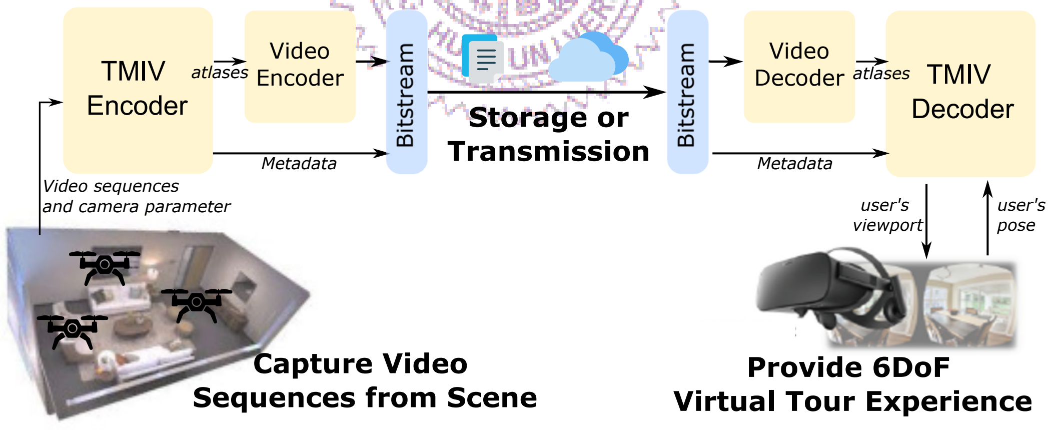


Figure 10.1: The usage scenario of a real-estate virtual tour.

We adopt real-estate as a use case, where HMD viewers can remotely walk around a house on the market; they can also visit the house at different times, e.g., to check if a room may suffer from excessive sun exposure in the late afternoon. Fig. 10.1 illustrates the usage scenario. First, several cameras capture video from various positions and rotations in a real-estate scene. The video sequence is compressed by TMIV encoder and video encoder, and is sent to a remote client or stored in the storage device. After that, the compressed video sequence is decoded by the video decoder and TMIV decoder. The view synthesizer in the TMIV decoder renders the user's viewport according to the user's position and orientation to provide a 6DoF virtual tour experience.

10.2 System Overview

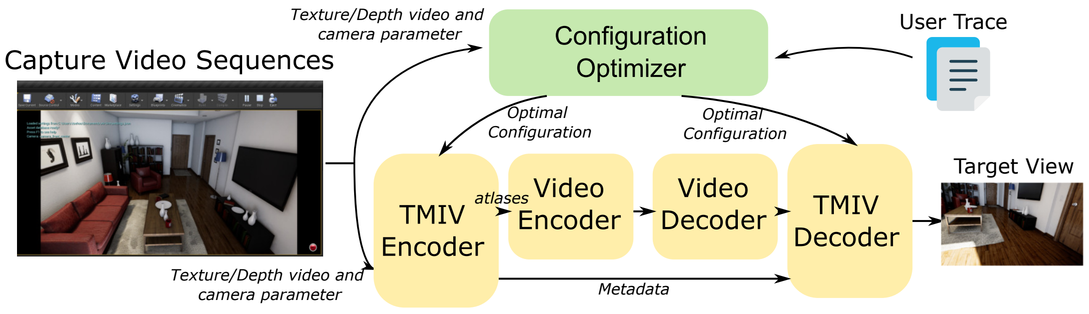


Figure 10.2: System overview of our implementation.

Fig. 10.2 shows the overview of our implementation for experiments. For capturing video sequences, we implemented a system based on Airsim [76] to capture video sequences from real-estate scenes. Airsim is a drone simulator based on Unreal Engine [24], which allows the user to create a virtual drone or vehicle with cameras in virtual scenes. After capturing video sequences, our configuration optimizer takes video sequences and camera parameters of source views and the target view as the inputs, and outputs the optimal configuration to TMIV codec. Finally, we ran TMIV and video codec to compress/de-compress the video sequences with configuration generated by the optimizer, and generated the target view according to the user trace.

10.3 Data Collection from the Photo-Realistic Simulator

Capturing high-quality video sequences and ground truth is not an easy task. In a real scene, multiple cameras have to be placed in various positions of the scene to capture RGB-D information, which is expensive and time-consuming. On the other hand, capturing video sequences from a virtual scene is relatively easier. It requires placing multiple virtual cameras by setting the camera parameters of each camera.

To capture video sequences from the virtual scene, 3D creation software and a game engine are utilized. Most 3D creation software already has an entire 3D pipeline (e.g., modeling, animation, and rendering), which can save much effort for implementing basic functions. For instance, some video sequences provided by MPEG [48] utilize blender [8], which is an open-source 3D creation software. Other similar software can also be utilized to support capturing. In our implementation, we chose Airsim in our system because of the following advantages: (i) Airsim provides Python API and documentation to help us build the system more easily. (ii) Airsim is a drone simulator based on Unreal Engine, which can simulate real environments physically. It is helpful when we want to conduct experiments considered real-world situations. (iii) For future experiments, how to stream sensor data from the capturing device to the remote server/client is a critical problem. Our implementation of Airsim can be applied to AirsimN [80], which is a network simulator for Airsim. With AirsimN, we can conduct experiments considered the network-related problem.

In our system, we use the computer vision mode of Airsim, capturing RGB and depth images by setting the camera parameters of the virtual cameras. We set the camera position and rotation, image resolution, and Field-of-View (FoV) according to the experiment setup, and set other parameters as the default value. After capturing data from Airsim, we carried out the following procedure to ensure that the data could be utilized by the MIV codec: (i) We calculated the focal length (x_F, y_F) and principal point (x_P, y_P) for the MIV codec from resolution and FoV, where $x_F = y_F = width/(2 * tan(FoV/2)$ and $x_P = width/2, y_P = height/2$. (ii) Because the coordinate systems of Airsim and MIV are different[1], we converted the coordinate of the data from Airsim to the MIV coordinate system by inverting y, z, $pitch$, and yaw. (iii) The format of the depth value required by the MIV codec is normalized disparity format [72]. To meet the requirement of the MIV codec, we captured the depth value by using the DepthPlanar format in Airsim, and converted the depth value by using the following equation to normalize the disparity format:

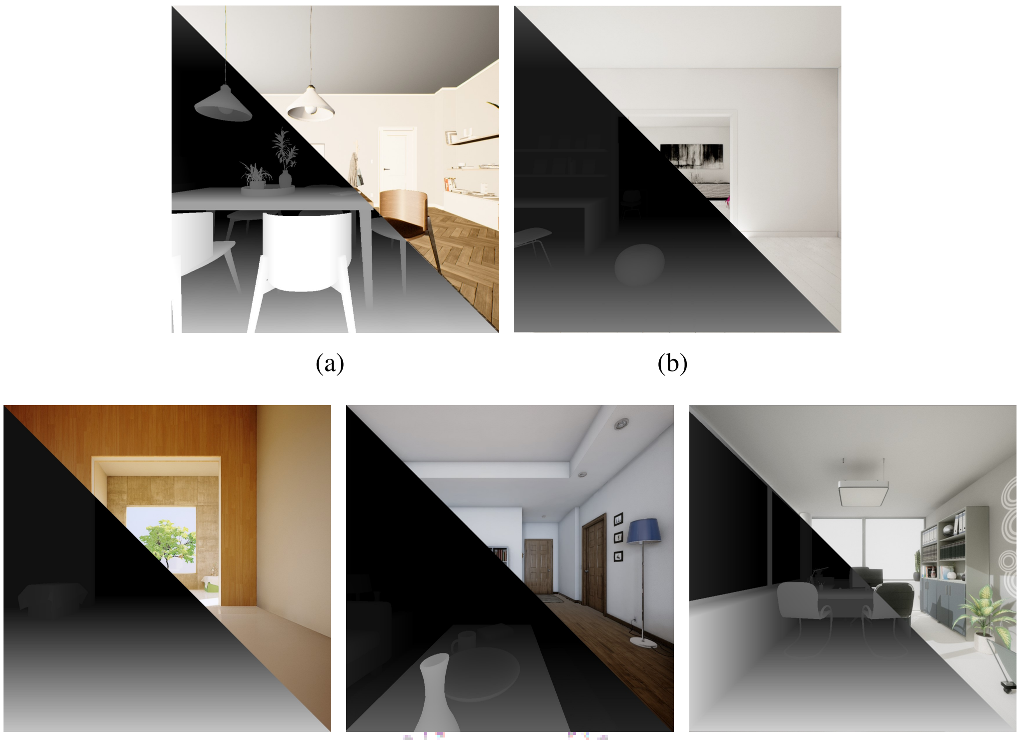
$$v = \frac{\frac{1}{Z} - \frac{1}{Z_{far}}}{\frac{1}{Z_{near}} - \frac{1}{Z_{far}}} \times v_{max} \tag{10.1}$$

where v is the depth value in the normalized disparity format, Z is the depth value captured from Airsim, Z_{far} and Z_{near} are the maximal and minimal depth values captured from Airsim, and v_{max} is the maximal binary value (65535 in case of 16-bits depth maps). Fig 10.3 shows the sample frame of capturing results. Our system can produce video sequences by capturing photo-realistic textures and accurate depth and can generate metadata needed by the MIV codec.

10.4 Experiment Setup

We captured source views from five real-estate scenes from Unreal Engine marketplace [25]. Fig. 10.3 shows sample screen-shots from the scenes. For each scene, we manually chose

[1]Airsim adopts the NED coordinate system, where +X is North, +Y is East and +Z is Down. MIV adopts the OMAF coordinate system, where +X is forward, +Y is left, +Z is up.


(a) (b)

(c) (d) (e)

Figure 10.3: Sample screen-shots from: (a) ArchVizInterior, (b) XoioBerlinFlat, (c) LightroomInteriorDayLight, (d) RealisticRendering, and (e) office.

a relatively representative direction which is supposed to have diverse features as our shooting target, and we placed 7 cameras with fixed camera placement, as shown in Fig 10.4. The resolution and FoV of the camera is 1024x1024 and 90°, respectively. After capturing video sequences, we ran TMIV to synthesize the target views for each scene to produce training and testing data for the configuration optimizers. We ran TMIV with the procedure similar to that shown in Sec. 6.5. The difference is the position and orientation of synthesized target views. We sampled 200 positions of target views for each view from the area surrounding the cameras. The sampling area is shown in Fig. 10.5. For the configuration optimizer, we used the model structure and parameters in Chapter 6, and the leave-one-out strategy was also adopted in training. Specifically, we trained five individual models. For each model, we assigned four video sequences as the training set and the remaining one as the testing set.

We evaluated the experiment results by using the baselines and metrics described in Sec. 7.1. However, in this chapter, we calculate the video quality metric by using video captured from Airsim as the ground truth rather than using the synthesized results from TMIV.

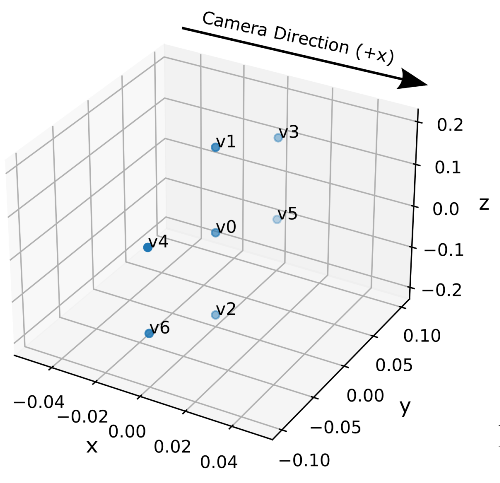


Figure 10.4: The camera placement.

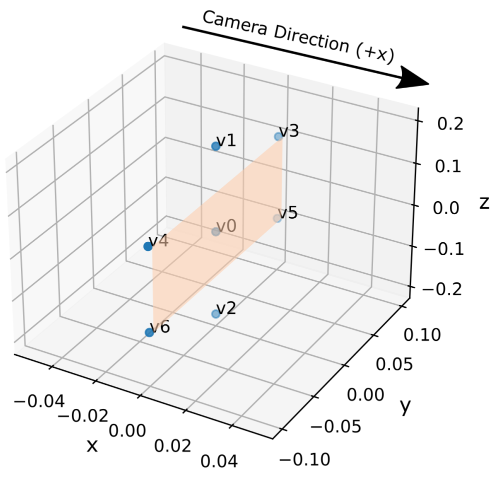
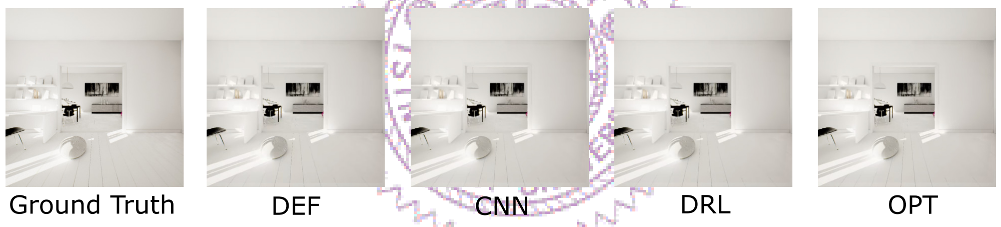
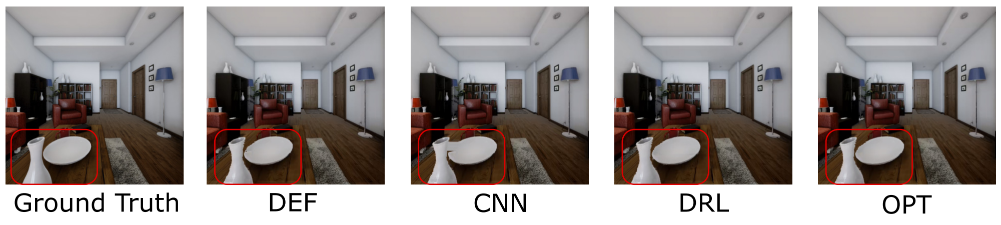


Figure 10.5: The sampling area of target views. The positions of target views are sampled from this area. The orientation of the target views is the same as the source view s camera.

10.5 Results



(a)



(b)

Figure 10.6: The ground truth and synthesized target views with the configurations generated by different algorithms

Fig 10.6 shows the ground truth and samples of the synthesized target views with the configurations generated by different algorithms. For most of the target views, there is no significant difference between each algorithm. However, the synthesized results may have minor distortion on the specific area of target views. For instance, in Fig. 10.6(b), there is some distortion in the red rectangular area. The reason for this distortion is that the algorithm may choose too few source views to cover the entire area of the target view.

We report the results from the training set in Fig. 10.7. In Fig 10.7(a), CNN and DRL require fewer source views compared to DEF. CNN use fewer views than DRL. At the same time, the CNN and DRL algorithms achieve higher PSNR than DEF in Fig 10.7(b), but the difference of PSNR between algorithms is not obvious. For decoding time, CNN and DRL algorithms use less decoding time than DEF in Fig. 10.7(c). The CNN algorithm achieves lower decoding time than the DRL algorithm. Finally, in Fig. 10.7(d), the CNN and DRL algorithms achieve higher optimal scores than DEF. Overall, the results in the training set show that our CNN and DRL algorithms use less computing and space resources to achieve similar viewing quality.

We report the results from the testing set in Fig. 10.8. The results for the testing set are similar to the results for the training set. The difference is the variance in Fig. 10.8(b) and 10.8(c) which is higher than the results for training set. Nonetheless, our algorithms still achieve higher optimal scores and fewer required views. Overall, the results for the testing set show the same conclusion as the results for the training set.

In our experiments, the results show that our algorithms achieve performance improvement similar to the previous results on the MPEG dataset. It shows that our algorithms are also applicable in real applications and scenarios.

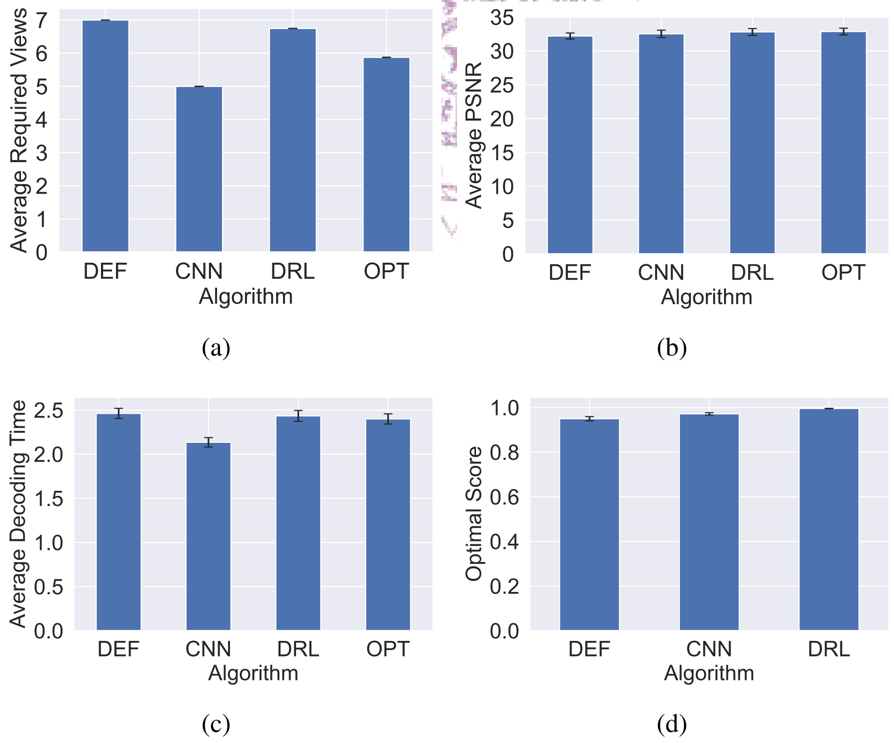
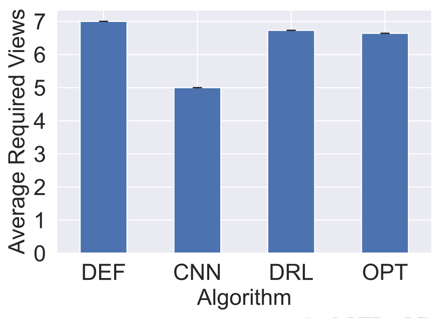
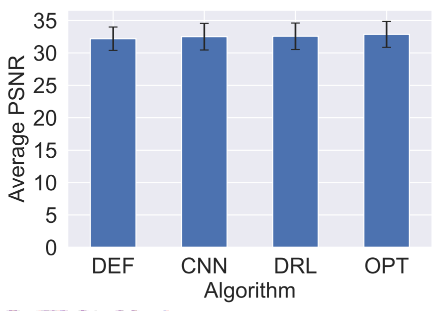


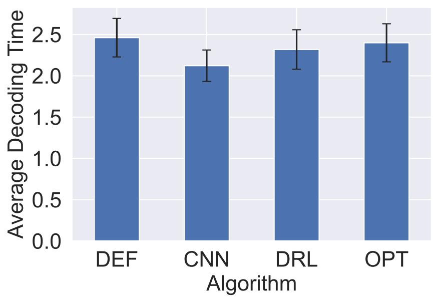
Figure 10.7: The results from the training set with five real-estate scenes: (a) number of views, (b) video quality, (c) decoding time, and (d) optimal score.



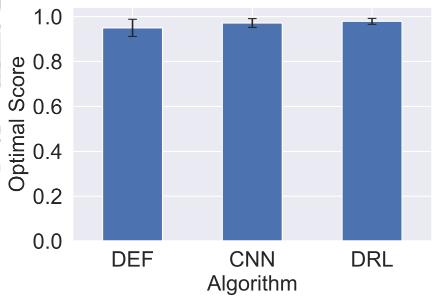
(a)



(b)



(c)



(d)

Figure 10.8: The results from the training set with five real-estate scenes: (a) number of views, (b) video quality, (c) decoding time, and (d) optimal score.

Chapter 11

Conclusion and Future Work

11.1 Conclusion

While 3DoF video streaming systems are becoming increasingly popular, enhancing the 3DoF systems into immersive streaming systems to support 3DoF+/6DoF remains an open issue. The MPEG-I group has proposed the TMIV codec for the emerging immersive video streaming systems, which can synthesize target views in arbitrary positions and orientations. Unfortunately, the TMIV codec dictates that the users manually specify configurations for view synthesis, which may lead to suboptimal results in terms of video quality, decoding time, and bandwidth consumption.

In this thesis, we propose two NN configuration optimization algorithms: the CNN and DRL algorithms, to solve the configuration optimization problem from two different perspectives. Our proposed algorithms work with a user-specified utility function and compute the best configurations for the TMIV codec. We adopt real video sequences from MPEG and evaluate our algorithms with both objective and subjective experiments. Our algorithms significantly reduce resource consumption and deliver similar video quality, compared to the default TMIV configurations. We also give recommendations for using NN algorithms in different scenarios. For ERP video sequences, we suggest using the DRL algorithm on both learned and new camera parameters. For PTP video sequences, we suggest using the DRL algorithm on learned video sequences and camera parameters, and use the CNN algorithm on the new ones.

11.2 Future Work

Our work is among the pioneering studies trying to optimize the TMIV configurations. We believe it can stimulate future research in immersive video streaming. Nevertheless, before our algorithms can be applied in end-to-end streaming systems, several system-level challenges need to be addressed. We list sample challenges below:

- **Inference frequency**: The invocation frequency of the configuration optimizer can be manually determined by users. In dynamic environments, an adaptation algorithm for the inference frequency is needed for higher and more stable user experience.

- **Generalization**: We have demonstrated that our NN-based algorithms can handle different video sequences. A practical system, however, needs to operate smoothly under diverse and dynamic network and system conditions. Therefore, the performance of such a system is affected by the available bandwidth, computational capability, and background application workload. These network and system statistics can also be fed into future (enhanced) NN models for better prediction performance.

- **Scalability**: High scalability is needed for a commercially-viable immersive video streaming service. Traditional approaches, such as hierarchical and distributed architectures, media-aware traffic and computational scheduling, load balance and migration, are all possible approaches to optimize such end-to-end systems.

Besides, the prediction performance of our algorithms can be improved by collecting more datasets with various video sequences and camera parameters or by employing future model structure and training approaches. The hard-constrained optimization can also be considered to support specific usage scenarios, e.g., video streaming with low bandwidth constrains.

We will also conduct extra experiments to explore the models' performance under different training designs. The experiment results are shown in the following paragraphs.
Limiting the Prediction Space of the CNN Algorithm. In our original design, our CNN algorithm predicted a vector of integers that represent a specific combination of the number of passes and the number of views per pass. Since there is no limitation for the value of the output vector's elements, the CNN algorithm may predict a configuration that is invalid for the TMIV codec, i.e., it may choose a number of views bigger than the total number of views. To overcome this possible drawback, in this section, we limit the number of prediction options by redesigning the output layer of the CNN algorithms. Specifically, we map each valid combination of the number of passes and the number of views per pass to an integer, and convert the integer to binary format. Our CNN algorithm predicts the binary of the integer to generate the optimal configuration. In this way, our CNN algorithm can only choose the valid configuration for TMIV.

The performance of redesigning the CNN algorithm[1] is shown in Table 11.1. After we limit the prediction space of the CNN algorithm, our algorithm can achieve lower re-

[1]We used datasets collected from chapter 10 to conduct the experiments. For the training set, we used

quired views, lower decoding time, and comparable video quality compared to the default configuration of TMIV. However, the optimal score of the CNN algorithm is lower than the default configuration. Overall, the experiment results show that our redesigned CNN algorithm can avoid invalid prediction results, but the performance is not as good as that of the original design.

Table 11.1: The performance comparison of the default configuration and the redesigned CNN algorithm.

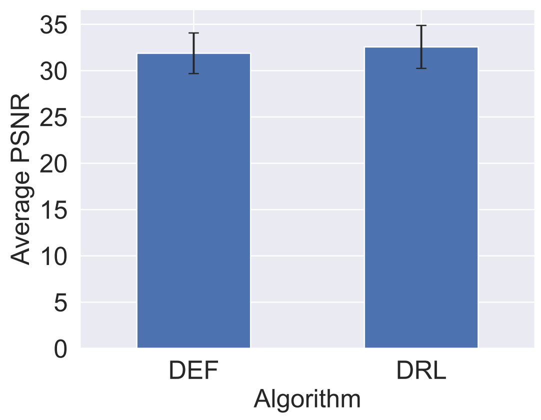
	Training		Testing	
	DEF	CNN	DEF	CNN
Required Views	7 (0)	3.25 (0.44)	7 (0)	2.99 (0.141)
Video Quality	31.87 (2.19)	31.58 (2.84)	33.46 (1.05)	32.76 (1.71)
Decoding Time	2.55 (0.19)	1.96 (0.35)	2.09 (0.05)	1.73 (0.14)
Optimal Score	4.63 (0.67)	4.51 (0.93)	6.45 (0.54)	6.11 (0.83)

Training the DRL Algorithm by Using Different Reward Functions. In our original design of the DRL algorithm, the reward function was a user-defined utility function. The user of our algorithm can design their own utility function according to the usage scenarios. In this section, we use video quality as our utility function to retrain our DRL agent. We conducted the experiments because: (i) We wanted to know whether our DRL algorithm would work when we changed the utility function, and (ii) Since the optimal video quality usually happens when the number of passes is big, we chose video quality as the utility function to evaluate the DRL algorithm under the long path of action.

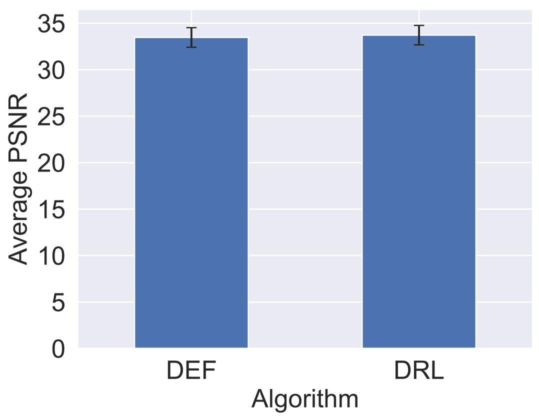
Fig. 11.1 shows the performance of the redesigning DRL algorithm[2]. The DRL algorithm achieves better video quality in the both training and testing sets. The improvement is small since the quality difference between different configurations is not big. The results show that our algorithm can still work under different utility functions.

the following video sequences: ArchVizInterior, XoioBerlinFlat, LightroomInteriorDayLight, and RealisticRendering. For the testing set, we used the following video sequence: office.

[2]We used datasets collected from chapter 10 to conduct the experiments. For the training set, we used the following video sequences: ArchVizInterior, XoioBerlinFlat, LightroomInteriorDayLight, and RealisticRendering. For the testing set, we used the following video sequence: office.



(a)



(b)

Figure 11.1: The performance comparison of the default configuration and the redesigned DRL algorithm: (a) training set and (b) testing set.

Bibliography

[1] D. Abe, L. Marc, and D. Fredo. Unstructured Light Fields. *Computer Graphics Forum*, 31:305–314, May 2012.

[2] E. Adelson and J. Bergen. *The Plenoptic Function and the Elements of Early Vision*, volume 2, chapter Models for Concurrency, pages 3–20. Computational Models of Visual Processing. Cambridge, MA: MIT Press (1991), 1991.

[3] S. Altamimi and S. Shirmohammadi. QoE-Fair DASH Video Streaming Using Server-Side Reinforcement Learning. *ACM Transacrions on Multimedia Computing, Communications, and Applications*, 16(2s):68:1–68:21, 2020.

[4] B. Attal, S. Ling, A. Gokaslan, C. Richardt, and J. Tompkin. Matryodshka: Real-Time 6DoF Video View Synthesis Using Multi-Sphere Images. In *Proc. of European Conference on Computer Vision*, pages 441–459, 2020.

[5] S. Avidan and A. Shashua. Novel View Synthesis in Tensor Space. In *Proc. of IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'97)*, pages 1034–1040, 1997.

[6] W. Bennett, J. Neel, V. Vaibhav, T. Eino-Ville, A. Emilio, B. Adam, A. Andrew, H. Mark, and L. Marc. High Performance Imaging Using Large Camera Arrays. In *Proc. of ACM Special Interest Group on Computer Graphics and Interactive Techniques Conference (SIGGRAPH'05)*, page 765–776, 2005.

[7] A. Bentaleb, B. Taani, A. Begen, C. Timmerer, and R. Zimmermann. A Survey on Bitrate Adaptation Schemes for Streaming Media over HTTP. *IEEE Communications Surveys Tutorials*, 21(1):562–585, 2019.

[8] blender. blender. 2021. Retrieved September 30, 2021 from `https://www.blender.org/`.

[9] J. Boyce, R. Doré, A. Dziembowski, J. Fleureau, J. Jung, B. Kroon, B. Salahieh, V. K. M. Vadakital, and L. Yu. MPEG Immersive Video Coding Standard. *IEEE Proceedings of the IEEE*, 109(9):1521–1536, 2021.

[10] Y. Chang, K. Chen, C. Wu, C. Ho, and C. Lei. Online Game QoE Evaluation Using Paired Comparisons. In *Proc. of IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR'10)*, pages 1–6, 2010.

[11] G. Chaurasia, S. Duchene, O. Sorkine-Hornung, and G. Drettakis. Depth Synthesis and Local Warps for Plausible Image-Based Navigation. *ACM Transactions on Graphics (TOG)*, 32(3):1–12, 2013.

[12] S. Chen. Quicktime VR: An Image-Based Approach to Virtual Environment Navigation. In *Proc. of conference on Computer graphics and interactive techniques*, pages 29–38, 1995.

[13] S. Chen and L. Williams. View Interpolation for Image Synthesis. In *Proc. of ACM Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'93)*, pages 279–288, 1993.

[14] W. Chen, Y. Chang, S. Lin, L. Ding, and L. Chen. Efficient Depth Image Based Rendering with Edge Dependent Depth Filter and Interpolation. In *Proc. of IEEE International Conference on Multimedia and Expo*, pages 1314–1317, 2005.

[15] B. Cheng, J. Yang, S. Wang, and J. Chen. Adaptive Video Transmission Control System Based on Reinforcement Learning Approach over Heterogeneous Networks. *IEEE Transactions on Automation Science and Engineering*, 12(3):1104–1113, 2015.

[16] F. Chiariotti, S. D'Aronco, L. Toni, and P. Frossard. Online Learning Adaptation Strategy for DASH Clients. In *Proc. of ACM International Conference on Multimedia Systems (MMSys'16)*, pages 8:1–8:12, 2016.

[17] C. Conti, L. Soares, and P. Nunes. Dense Light Field Coding: A Survey. *IEEE Access*, 8:49244–49284, March 2020.

[18] X. Corbillon, F. Simone, G. Simon, and P. Frossard. Dynamic Adaptive Streaming for Multi-viewpoint Omnidirectional Videos. In *Proc. of ACM International Conference on Multimedia Systems Conference (MMSys'18)*, pages 237–249, 2018.

[19] L. Costero, A. Iranfar, M. Zapater, F. Igual, K. Olcoz, and D. Atienza. MAMUT: Multi-Agent Reinforcement Learning for Efficient Real-Time Multi-User Video Transcoding. In *Proc. of IEEE Design, Automation Test in Europe Conference Exhibition (DATE'19)*, pages 558–563, 2019.

[20] I. Curcio, K. Kammachi-Sreedhar, and S. Mate. Multi-Viewpoint and Overlays in the MPEG OMAF Standard. *ITU Journal: ICT Discoveries*, 3(1):17–24, 2020.

[21] P. Debevec, C. Taylor, and J. Malik. Modeling and Rendering Architecture from Photographs: A Hybrid Geometry-and Image-Based Approach. In *Proc. of conference on Computer graphics and interactive techniques*, pages 11–20, 1996.

[22] R. Doré and G. Lafruit. Updated Call for Test Materials for 3DoF+ Visual. International Organization for Standardization Meeting Document ISO/IEC JTC1/SC29/WG11 MPEG2018/N17617, 2018.

[23] A. Dziembowski, J. Samelak, and M. Domański. View Selection for Virtual View Synthesis in Free Navigation Systems. In *Proc. of IEEE International Conference on Signals and Electronic Systems (ICSES'18)*, pages 83–87, 2018.

[24] EPIC Games. Unreal Engine. 2021. Retrieved August 29, 2021 from `https://www.unrealengine.com/en-US/`.

[25] EPIC Games. Unreal Engine Marketplace. 2021. Retrieved August 29, 2021 from `https://www.unrealengine.com/marketplace/en-US/store`.

[26] A. Eslami, J. Rezende, F. Besse, F. Viola, A. Morcos, M. Garnelo, A. Ruderman, A. Rusu, I. Danihelka, K. Gregor, et al. Neural Scene Representation and Rendering. *Science*, 360(6394):1204–1210, 2018.

[27] C. Fan, W. Lo, Y. Pai, and C. Hsu. A Survey on 360° Video Streaming: Acquisition, Transmission, and Display. *ACM Computing Surveys*, 52(4):71:1–71:36, 2019.

[28] C. Fehn. Depth-Image-Based Rendering (DIBR), Compression, and Transmission for a New Approach on 3D-TV. In *Proc. SPIE 5291, Stereoscopic Displays and Virtual Reality Systems XI*, pages 93–104, 2004.

[29] J. Fleureau, B. Chupeau, F. Thudor, G. Briand, T. Tapie, and R. Doré. An Immersive Video Experience with Real-Time View Synthesis Leveraging the Upcoming MIV Distribution Standard. In *Proc. of IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, pages 1–2, 2020.

[30] J. Fu, X. Chen, Z. Zhang, S. Wu, and Z. Chen. 360SRL: A Sequential Reinforcement Learning Approach for ABR Tile-Based 360 Video Streaming. In *Proc. of IEEE International Conference on Multimedia and Expo (ICME'19)*, pages 290–295, 2019.

[31] M. Gadaleta, F. Chiariotti, M. Rossi, and A. Zanella. D-DASH: A Deep Q-Learning Framework for DASH Video Streaming. *IEEE Transactions on Cognitive Communications and Networking*, 3(4):703–718, 2017.

[32] G. Gescheider. *Psychophysics: the Fundamentals*. Psychology Press, 2013.

[33] R. Ghaznavi-Youvalari and A. Aminlou. Geometry-Based Motion Vector Scaling for Omnidirectional Video Coding. In *Proc. of IEEE International Symposium on Multimedia (ISM)*, pages 127–130, 2018.

[34] A. Ghosh, V. Aggarwal, and F. Qian. A rate adaptation algorithm for tile-Based 360-degree video streaming. *arXiv preprint arXiv:1704.08215*, 2017.

[35] S. Gortler, R. Grzeszczuk, R. Szeliski, and M. Cohen. The Lumigraph. In *Proc. of conference on Computer graphics and interactive techniques*, pages 43–54, 1996.

[36] D. Graziosi, O. Nakagami, S. Kuma, A. Zaghetto, T. Suzuki, and A. Tabatabai. An Overview of Ongoing Point Cloud Compression Standardization Activities: Video-Based (V-PCC) and Geometry-Based (G-PCC). *APSIPA Transactions on Signal and Information Processing*, 9(0):e13, 2020.

[37] M. Hannuksela, Y. Wang, and A. Hourunranta. An Overview of the OMAF Standard for 360 Video. In *Proc. of IEEE Data compression conference (DCC)*, pages 418–427, 2019.

[38] P. Hedman, J. Philip, T. Price, J. Frahm, G. Drettakis, and G. Brostow. Deep Blending for Free-Viewpoint Image-Based Rendering. *ACM Transactions on Graphics (TOG)*, 37(6):1–15, 2018.

[39] J. Hooft, S. Petrangeli, M. Claeys, J. Famaey, and F. Turck. A learning-Based algorithm for improved bandwidth-awareness of adaptive streaming clients. In *Proc. of IFIP/IEEE International Symposium on Integrated Network Management (IM'15)*, pages 131–138, 2015.

[40] M. Hosseini, G. Kurillo, S. Etesami, and J. Yu. Towards coordinated bandwidth adaptations for hundred-scale 3D tele-immersive systems. *Springer Multimedia Systems*, 23(4):421–434, 2017.

[41] HTC VIVE. HTC VIVE. 2019. Retrieved April 21, 2020 from `https://www.vive.com/tw/product/vive`.

[42] J. Hu, W. Peng, and C. Chung. HEVC/H.265 Coding Unit Split Decision Using Deep Reinforcement Learning. In *Proc. of IEEE International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS'17)*, pages 570–575, 2017.

[43] J. Hu, W. Peng, and C. Chung. Reinforcement Learning for HEVC/H.265 Intra-Frame Rate Control. In *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS'18)*, pages 1–5, 2018.

[44] J. Huang, Z. Chen, D. Ceylan, and H. Jin. 6-DOF VR Videos with A Single 360-Camera. In *Proc. of IEEE Virtual Reality Conference (VR'17)*, pages 37–44, 2017.

[45] T. Huang, R. Zhang, C. Zhou, and L. Sun. QARC: Video Quality Aware Rate Control for Real-Time Video Streaming Based on Deep Reinforcement Learning. In *Proc. of ACM International Conference on Multimedia (MM'18)*, pages 1208–1216, 2018.

[46] J. Jeong, S. Lee, I. Ryu, T. Le, and E. Ryu. Towards Viewport-Dependent 6DoF 360 Video Tiled Streaming for Virtual Reality Systems. In *Proc. of ACM International Conference on Multimedia (MM'20)*, page 3687–3695, 2020.

[47] X. Jiang, Y. Chiang, Y. Zhao, and Y. Ji. Plato: Learning-Based Adaptive Streaming of 360-Degree Videos. In *Proc. of IEEE Conference on Local Computer Networks (LCN'18)*, pages 393–400, 2018.

[48] J. Jung, B. Kroon, and J. Boyce. Common Test Conditions for Immersive Video. International Organization for Standardization Meeting Document ISO/IEC JTC1/SC29/WG11 MPEG/N18563, 2019.

[49] N. Kalantari, T. Wang, and R. Ramamoorthi. Learning-Based view synthesis for light field cameras. *ACM Transactions on Graphics (TOG)*, 35(6):1–10, 2016.

[50] T. Kanade, P. Rander, and P. Narayanan. Virtualized Reality: Constructing Virtual Worlds from Real Scenes. *IEEE multimedia*, 4(1):34–47, 1997.

[51] L. Kapov, M. Varela, T. Hoßfeld, and K. Chen. A Survey of Emerging Concepts and Challenges for QoE Management of Multimedia Services. *ACM Transacrions on Multimedia Computing, Communications,and Application*, 14(2s):29:1–29:29, 2018.

[52] D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *Proc. of International Conference on Learning Representationsnce Track (poster)*, 2015.

[53] M. Levoy and P. Hanrahan. Light Field Rendering. In *Proc. of conference on Computer graphics and interactive techniques*, pages 31–42, 1996.

[54] L. Li, Z. Li, X. Ma, H. Yang, and H. Li. Advanced Spherical Motion Model and Local Padding for 360° Video Compression. *IEEE Transactions on Image Processing*, 28(5):2342–2356, 2019.

[55] L. Marc and H. Pat. Light Field Rendering. In *Proc. of ACM Special Interest Group on Computer Graphics and Interactive Techniques Conference (SIGGRAPH'96)*, pages 31–42, 1996.

[56] C. Maxim, L. Steven, F. Jeroen, and D. Filip. Design and Evaluation of a Self-Learning HTTP Adaptive Video Streaming Client. *IEEE Communications Letters*, 18(4):716–719, 2014.

[57] L. McMillan and G. Bishop. Plenoptic Modeling: An Image-Based Rendering System. In *Proc. of conference on Computer graphics and interactive techniques*, pages 39–46, 1995.

[58] B. Mildenhall, P. Srinivasan, R. Ortiz-Cayon, N. Kalantari, R. Ramamoorthi, R. Ng, and A. Kar. Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines. *ACM Transactions on Graphics (TOG)*, 38(4):1–14, 2019.

[59] B. Mildenhall, P. Srinivasan, M. Tancik, J. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing Scenes as Neural Radiance Fields for View Synthesis. In *Proc. of European conference on computer vision*, pages 405–421, 2020.

[60] MPEG. HM 16.16. 2019. Retrieved April 21, 2020 from `https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-16.16/`.

[61] MPEG. Activity Report on Dense Light Fields. International Organization for Standardization Meeting Document ISO/IEC JTC1/SC29/WG11MPEG2020/N19493, 2020.

[62] MPEG. MPEG roadmap. International Organization for Standardization Meeting Document ISO/IEC JTC1/SC29/WG11 MPEG/w19514, 2020.

[63] MPEG. Text of ISO/IEC 23090-9 DIS Geometry-Based PCC. International Organization for Standardization Meeting Document ISO/IEC JTC1/SC29/WG11 MPEG2020/N19088, 2020.

[64] MPEG. Text of ISO/IEC DIS 23090-5 Video-Based Point Cloud Compression. International Organization for Standardization Meeting Document ISO/IEC JTC1/SC29/WG11 MPEG2020/N18670, 2020.

[65] MPEG. Text of ISO/IEC FDIS 23090-5 Visual Volumetric Video-Based Coding and Video-Based Point Cloud Compression. International Organization for Standardization Meeting Document ISO/IEC JTC1/SC29/WG11 MPEG/w19579, 2020.

[66] MPEG. Text of ISO/IEC DIS 23090-12 MPEG Immersive Video. International Organization for Standardization Meeting Document ISO/IEC JTC1/SC29/WG11 MPEG/w20003, 2021.

[67] K. Mueller, A. Smolic, K. Dix, P. Merkle, P. Kauff, and T. Wiegand. View Synthesis for Advanced 3D Video Systems. *Springer EURASIP Journal on image and video processing*, 2008(0):1–11, 2009.

[68] P. Ndjiki-Nya, M. Koppel, D. Doshkov, H. Lakshman, P. Merkle, K. Muller, and T. Wiegand. Depth Image-Based Rendering with Advanced Texture Synthesis for 3-D Video. *IEEE Transactions on Multimedia*, 13(3):453–465, 2011.

[69] H. Pang, C. Zhang, F. Wang, J. Liu, and L. Sun. Towards Low Latency Multi-viewpoint 360° Interactive Video: A Multimodal Deep Reinforcement Learning Approach. In *Proc. of IEEE Conference on Computer Communications (INFO-COM'19)*, pages 991–999, 2019.

[70] R. Placket. The Analysis of Permutations. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 24(2):193–202, 1975.

[71] B. Salahieh, S. Bhatia, and J. Boyce. Multi-Pass Add-on Tool for Coherent and Complete View Synthesis (US Patent 2019/0320164 A1), 2019.

[72] B. Salahieh, B. Kroon, J. Jung, and M. Domański. Test Model for Immersive Video. International Organization for Standardization Meeting Document ISO/IEC JTC1/SC29/WG11 MPEG2017/N16730, 2017.

[73] B. Salahieh, B. Kroon, J. Jung, and M. Domański. Test Model 2 for Immersive Video. International Organization for Standardization Meeting Document ISO/IEC JTC1/SC29/WG11 MPEG/N18577, 2019.

[74] B. Salahieh, B. Kroon, J. Jung, and M. Domański. Test Model 3 for Immersive Video. International Organization for Standardization Meeting Document ISO/IEC JTC1/SC29/WG11 MPEG/N18795, 2019.

[75] B. Salahieh, B. Kroon, J. Jung, and M. Domański. Test Model for Immersive Video. International Organization for Standardization Meeting Document ISO/IEC JTC1/SC29/WG11 MPEG/N18470, 2019.

[76] S. Shah, D. Dey, C. Lovett, and A. Kapoor. Airsim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *Proc. of Field and Service Robotics*, pages 621–635, 2018.

[77] G. Sullivan, J. Ohm, W. Han, and T. Wiegand. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, 2012.

[78] Y. Sun, A. Lu, and L. Yu. Weighted-to-Spherically-Uniform Quality Evaluation for Omnidirectional Video. *IEEE Signal Processing Letters*, 24(9):1408–1412, 2017.

[79] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction.* A Bradford Book, 2 edition, 2018.

[80] S. Tang, C. Hsu, Z. Tian, and X. Su. An Aerodynamic, Computer Vision, and Network Simulator for Networked Drone Applications. In *Proc. of ACM International Conference on Mobile Computing and Networking (MobiCom'21) Poster Session*, pages 0–0, 2022.

[81] D. Tian, P. Lai, P. Lopez, and C. Gomila. View Synthesis Techniques for 3D Video. In *Proc. of SPIE Conference on Applications of Digital Image Processing (ADIP'09)*, pages 74430T:1–74430T:11, 2009.

[82] R. Tucker and N. Snavely. Single-View View Synthesis with Multiplane Images. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 551–560, 2020.

[83] B. Vishwanath, T. Nanjundaswamy, and K. Rose. Rotational motion model for temporal prediction in 360 video coding. In *Proc. of IEEE International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–6, 2017.

[84] Y. Wang, D. Liu, S. Ma, F. Wu, and W. Gao. Spherical Coordinates Transform-Based Motion Model for Panoramic Video Coding. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(1):98–109, 2019.

[85] M. Wien, J. Boyce, T. Stockhammer, and W. Peng. Standardization Status of Immersive Video Coding. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(1):5–17, 2019.

[86] S. Wizadwongsa, P. Phongthawee, J. Yenphraphai, and S. Suwajanakorn. Nex: Real-Time View Synthesis with Neural Basis Expansion. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8534–8543, 2021.

[87] M. Xu, C. Li, S. Zhang, and P. Le Callet. State-of-the-art in 360 Video/Image Processing: Perception, Assessment and Compression. *IEEE Journal of Selected Topics in Signal Processing*, 14(1):5–26, 2020.

[88] A. Yaqoob, T. Bi, and G.-M. Muntean. A Survey on Adaptive 360° Video Streaming: Solutions, Challenges and Opportunities. *IEEE Communications Surveys Tutorials*, 22(4):2801–2838, 2020.

[89] Y. Zhang, P. Zhao, K. Bian, Y. Liu, L. Song, and X. Li. DRL360: 360-degree Video Streaming with Deep Reinforcement Learning. In *Proc. of IEEE Conference on Computer Communications (INFOCOM'19)*, pages 1252–1260, 2019.

[90] T. Zhou, R. Tucker, J. Flynn, G. Fyffe, and N. Snavely. Stereo Magnification: Learning View Synthesis Using Multiplane Images. *arXiv preprint arXiv:1805.09817*, 2018.

[91] T. Zhou, R. Tucker, J. Flynn, G. Fyffe, and N. Snavely. Stereo Magnification: Learning View Synthesis Using Multiplane Images. *ACM Transactions on Graphics (TOG)*, 37(4), 2018.

[92] S. Zinger, L. Do, and P. de With. Free-Viewpoint Depth Image Based Rendering. *Elsevier Journal of visual communication and image representation*, 21(5-6):533–541, 2010.

[93] ZION Market Research. Virtual Reality (VR) Market by Hardware and Software for (Consumer, Commercial, Enterprise, Medical, Aerospace and Defense, Automotive, Energy and Others): Global Industry Perspective, Comprehensive Analysis and Forecast, 2016–2022. 2018. Retrieved April 21, 2020 from `https://www.zionmarketresearch.com/report/virtual-reality-market`.

[94] L. Zitnick, S. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High-Quality Video View Interpolation Using a Layered Representation. *ACM Transactions on Graphics (TOG)*, 23(3):600–608, 2004.