國立清華大學電機資訊學院資訊工程研究所
碩士論文
Department of Computer Science
College of Electrical Engineering and Computer Science
National Tsing Hua University
Master Thesis

頭戴式虛擬實境中之光場技術應用

Capitalizing Light-Field Technology in Head-Mounted Virtual Reality

賴宥銘
Yu-Ming Lai

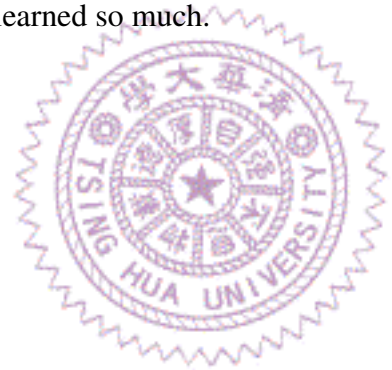
學號：106062552
Student ID:106062552

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

中華民國 108 年 9 月
September, 2019

Acknowledgements

I would like to express my gratitude toward all the people who helped me in the past two years. I would not be able to finish my thesis were it not without your help along the way. I want to thank my parents specifically, for it is they who provided me with the firm support and stood behind my decisions. I would also like to thank my labmates in Networking and Multimedia Systems Laboratory, especially Ching-Ling Fan, Hua-Jun Hong, who helped me a great deal in the course of my research. Lastly, I would like to express my gratitude toward my adviser: Prof Cheng-Hsin Hsu. Without the guidance and the suggestion I received from him, I would not be able to accomplish what I have done and learned so much.



致謝

在此我要感謝在過去兩年中所有幫助過我的人， 如果沒有你們的幫助我一定沒有辦法順利完成我的論文。 在此我要特別感謝我的父母，他們提供我堅定不移的支持， 同時也支持我所作的每一個決定。我也要感謝網路與多媒體系統實驗室的同學們，特別是樊慶玲、洪華駿、及其他實驗室同仁在過去兩年中， 不管是研究或人生課題都幫助我非常的多。 最後，我要感謝我的指導教授：徐正炘教授。如果沒有他的給予我的指導以及建議， 在過去的兩年內我一定沒辦法完成如此多的事情以及學到如此多的東西。

Abstract

Augmented and Virtual Reality (AR/VR) has become more popular over the years. It delivers a more immersive experience than using the traditional planar monitor with the head-mounted display (HMD). Still, to increase the Quality of Experience (QoE), researchers dedicate to building a better environment with more captured space information. With the capability to retrieve all light information in the space, the light field technology (LF) has excellent potential for the future development of AR/VR technology. In this paper, we study and research two possible directions of LF applications in AR/VR. In the microlens camera system, we design and implement a head-mounted VR system that enables the auto scene refocusing based on the user's eye gaze. To optimize the latency of the refocusing process, we design two optimization methods that significantly reduce the execution time. In the camera array system, we develop a 3DoF+ VR environment and create a novel view selection algorithm that exploits the 3D space information (view scene coverage, object occlusion) of the scene to save both the bandwidth and the computation of view synthesis process. Finally, we hold experiments in both objective and subjective perspectives to evaluate the performance of the systems. The results show that, for the auto-refocus VR system, our optimization methods reduce the refocusing time by up to 319 times and increase the subjective Mean Opinion Score (MOS) by 19% compared to the baseline system. As for the view selection algorithm, our proposed algorithm leads to 99.67% of average synthesis result coverage, which is only 0.1% lower than the optimal solution. However, at the same time, our execution time is about 18 times faster than the optimal solution.

中文摘要

擴增和虛擬實境（AR / VR）在近年來蔚為風行，而隨著頭戴顯示器的普及使用，它提供了使用者比傳統平面顯示器更加身臨其境的體驗。 儘管如此，為了提供更高品質的觀看體驗，研究人員致力於建立一個能夠捕獲更多空間信息的環境。 其中，光場技術能夠收集空間中的所有信息，具有很大的發展潛力。 在本論文中，我們研究了AR / VR中光場技術應用的兩個發展方向。 首先，在微透鏡相機系統中，我們設計了一套自動聚焦VR系統，能夠根據使用者的注視位置自動對場景重新聚焦， 並在優化層面設計了2種方法來大幅減少重新聚焦的計算時間。 而在相機陣列系統中，我們開發了一套3DoF+ VR系統、並同時設計了一套新的視圖選擇算法， 該算法能有效利用場景中的資訊（包刮視圖覆蓋區域、物體遮擋等）來節省視圖合成需要的頻寬及運算量。 最後，我們收集了以客觀和主觀角度執行的實驗結果，以評估系統效能。 結果表明，對於自動聚焦VR系統，我們的優化將重新聚焦的時間縮短了近319倍， 並且與基準系統相比，我們系統的主觀平均意見分數（MOS）高出了19％。 而對於視圖選擇算法，我們提出的算法可以得到高達99.67％的平均覆蓋率，只比最優解低了0.1％， 而同時我們的計算時間比最優解快了近18倍。

Contents

List of Figures

List of Tables

Chapter 1

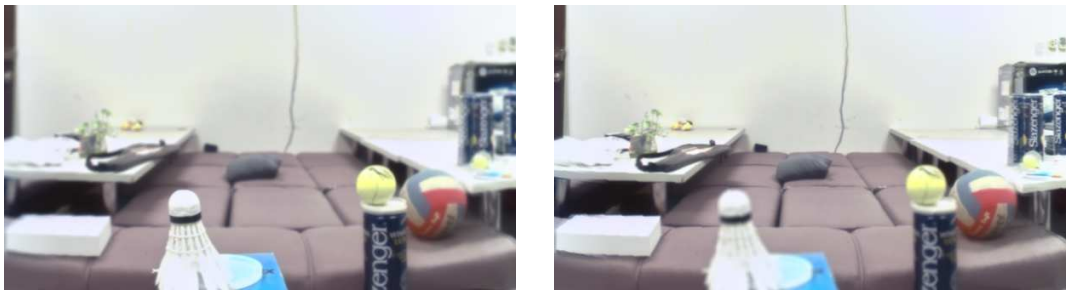
Introduction

1.1 Motivation

Augmented and virtual reality (AR/VR) have been thriving in recent years, and have drawn much attention from both industries and academia. For its capability of providing an omnidirectional experience of virtual worlds and captured scenes, With recent advances in hardware, network, computation power, storage, and other adjunct areas, different novel AR/VR applications are widely promoted in various fields. For instance, Facebook spaces [4] allow users to socialize with their friends from different places in a shared virtual room. For retailers, customers may buy clothes online with virtual fitting rooms to save commute time and expense. In education, HMDs may capture more students' attention at lower costs for the more intuitive classroom experience. Without a doubt, AR/VR may bring a brand new way of working, communication, and entertainment shortly.

According to the reports, it is estimated that the global VR market size will reach 26.89 billion USD by 2022, with an annual growth rate of 54% from 2017 to 2022 [54]. Also, IDC reports [55] indicate that more than a billion people will be using and accessing AR/VR apps, content, and data by 2021. There's no doubt that AR/VR is one of the most trending technologies in the world. Moreover, for the more immersive viewing experience, the Head-Mounted Displays (HMDs) is widely researched and used. Compared to the traditional planar monitors, HMDs enables the dynamic viewport changes in real-time, which makes the VR viewing experience more immersive and intuitive. Over the past years, AR/VR products, such as HMDs, have become easily accessible. Many companies develop and release their HMD products, such as Oculus Rift DK2 [8], HTC Vive [13], FOVE VR [10], Google Cardboard [12], Sony PlayStation VR [21], and so on. These products offer viewers wider Field-of-Views (FoVs) than traditional monitors and dedicate to provide better user experience, some of them can even detect user body move-

ment and eye movement [14, 10], which makes more advanced usage scenarios possible. Furthermore, lots of 360° cameras, which take pictures in 360° panorama format, also come to the market, for instance, Ricoh Theta S [19], Luna 360 VR [15], and Samsung Gear 360 [20].



(a) (b)

Figure 1.1: Same scene with different focal length: (a) apply short focal length and focus on the badminton, (b) apply long focal length and focus on the ball cans.

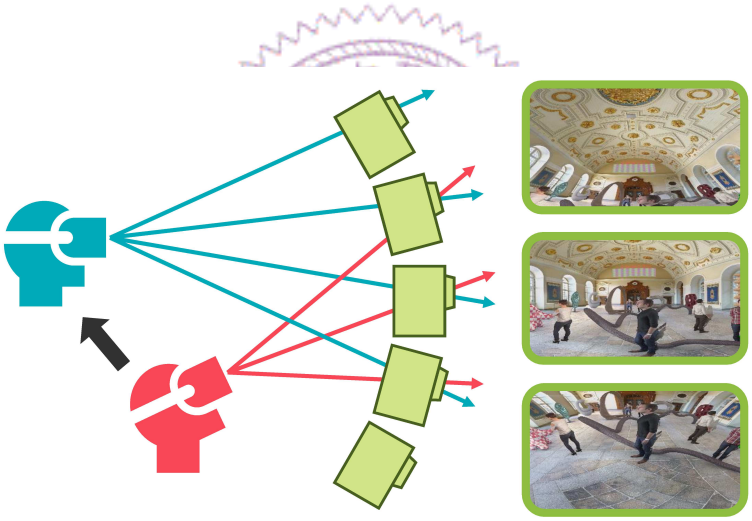


Figure 1.2: The schematic diagram of multi-view video system.

As the development of the 360° video continues to thrive, there are still several limitations that keep the technology from the more immersive experience, including *fixed picture focal length* and *fixed viewpoint*. These limitations not only keep users from further exploring the scene but also may result in diverse types of discomfort, including dizziness, imbalance, and vomiting, which lead to significant drops of the QoE. The following are some observations on these two limitations:

- **Fixed video focal length.** Conventionally, images/videos are taken with a *default* focal length, which leads to the fixed focused and blurred area in the videos/images and we can *not* adjust the focus afterward. Since there are no focus/defocus effect in

the 360° images/videos, the objects' proper distances from the eye gaze cannot be indicated when users watch the scene in HMDs. The phenomenon violates human visual adaptation and hence leads to confusion of our brains, it's not hard to imagine the discomfort due to the *mismatch* between the DoF and eye gaze [49, 29, 52]. To resolve the mentioned problem, we use the dynamic focal length adaptation mechanic, in which the focal length of the scene should automatically change based on the depth value of whatever objects the users are gazing. As shown in Fig. 1.1, when users gaze on the badminton, a short focal length is applied, and the scene is then refocusing on the badminton; and when users gaze on the ball cans on the table, a longer focal length is applied, and the scene should be focusing on the cans.

- **Fixed viewpoint.** Usually, photographers take images/videos from a single viewpoint instead of multiple viewpoints. The single viewpoint means that users cannot move their perspectives in the video even if they change their positions (head, body) in real life when viewing 360° videos with HMDs, resulting in the conflicts between what information our bodies and our eyes send to our brains and may cause discomfort as well [49]. There are several possible solutions to deal with the problem, one of the most intuitive ways is by increasing the number of viewpoints, that is, to capture the scene from multiple different positions and rotations in advance, and generate the view of the user viewpoint during playout time by interpolation or other blending methods. Fig. 1.2 demonstrate the concept of multi-viewpoint media, Since the light information is collected from multiple different viewpoints, users can move their positions with the corresponding scenes in HMDs available (as long as moving within the range of captured scene). For this kind of HMD VR, MPEG has categorized it as the *3DoF+ VR*, which is a standardization of the immersive media. More details of 3DoF+ VR will be elaborated in Sec. 2.3.

As the observations indicate, these limitations are the main factors why people feel the discomfort after a long-time wearing HMD during VR exploration. Thus, the main goal of this thesis is to propose systems that can eliminate the discomfort caused by the limitations and hopefully increase the user's Quality of Experience (QoE).

1.2 Research Problems

To better understand the limitations, we first study the state-of-the-art 3D video technology [56, 30, 33, 51] and survey its possible applications in 360° video [32, 53, 71]. To better handle the problems, we use the **light field (LF)** technology [46, 50, 66], which is a special format of the multi-view video in a more efficient way. With the characteristic

of the LF image, we can perform image refocusing effect with an arbitrary depth value within the focal range as well as view synthesis with any viewpoint within image capturing range. We introduce the background knowledge of the LF technology in Chapter. 2.

However, solely with the methods is not enough for a real system for there are still several challenges that need to be solved, such as heavy computation, storage/memory exceeding, bandwidth consumption. Throughout this thesis, we strive to solve these challenges and establish two real VR systems to deal with the limitations mentioned above and maximize user experience. For each system, we pick one most important challenge as our research problems:

- **Auto-refocus VR system.** The main focus on this system is the computation of the refocusing process, which is the foundation of the depth adaptation. The computation of the process is extremely heavy due to the enormous size of the LF data, and this profoundly impacts the performance of the system. To build up a real and workable system, we must optimize the refocusing process and make it work in real-time. Thus, how to optimize the refocusing process as well as the whole system is the leading research problem here.

- **3DoF+ VR system.** The most important problem of this system is to reduce the number of views for view synthesis, which is the core of the 3DoF+ VR. Since the computation of view synthesis is heavily reliant on the number of reference views as well as the pixel number it needs to process, we want the used reference view to be as relevant as the target view, not only in geometry relationship (position, rotation) but also in 3D space coverage. To be specific, the used reference views must lead to a high-quality synthesis result as possible while in the meantime, limit the number of used reference view to save the computation, and ultimately leads to the view selection problem of the reference views. Therefore, to find an effective and efficient view selection algorithm is the leading research problem of this system.

1.3 Contributions

In this thesis, we study the aforementioned limitations of 360° video and present ways to solve it using LF technology. Also, we develop real systems to tackle each limitation by proposing the optimization methods as well as the novel algorithm. Eventually, in this thesis, we made the following contributions:

- We propose and implement an HMD VR system with real-time depth adaptation supports based on the user eye movement. To our best knowledge, this has never

been done in the literature. Two practical optimization techniques are proposed to reduce the computation latency of the refocusing process. The details of the HMD VR system are described in Chapter 3. Also, the work of the system is highlighted in ACM AltMM'18 workshop [45].

- We propose and implement another 3DoF+ VR system that allows users to move their heads around with HMDs. We also offer a view selection algorithm that can exploit the 3D space information, like scene coverage and object occlusion information, of the reference view. To ensure the efficiency of the selection, we map the problem to Maximum Coverage Problem (MCP), which is an NP-hard question, and approximate it with a polynomial-time solution. The details of the system are described in Chapter 4.

- We evaluate the performance of both systems in objective and subjective perspectives. In the auto-refocus VR system, we measured the system performance and analyzed the effectiveness of the optimization methods. We also held a user study to learn the impact of depth adaptation to the user's QoE level. The results show the efficiency of our system with auto-refocus supports, about 319 times faster than the baseline system, and show that providing Depth of Field is indeed critical for good user experience with a 19% higher Mean Opinion Score (MOS). In the 3DoF+ VR system, we evaluate the effectiveness and efficiency of our proposed view selection algorithm by comparing to the geometry-based algorithm and the optimal set selection (brute force). We also test online vs. offline view selection and see the possibility of view set caching. The results show that our proposed algorithm holds about 6.15% higher view coverage percentage than the geometry-based algorithm on average and only approximately 0.1% lower than the optimal selection on average. Moreover, we cache 5929 views and test with 1200 views, which are an extract from real user traces, to see the performance of offline selection. The results demonstrate a somewhat competent ability to the online selection (about 1% lower on average).

1.4 Thesis Organization

The rest of this thesis is organized as followed: we give an introduction in the AR/VR research field and list down the limitation as well as the solution of current 360° video viewing experience in Chapter 1; the background knowledge of light field technology (LF) in Chapter 2; the auto-refocus VR system we proposed to solve the fixed focal length limitation, including its system architecture, optimization methods, implementation details

and evaluations in Chapter 3; another 3DoF+ VR system we proposed to tackle the fixed viewpoint limitation, including its system overview, algorithm design, implementation details and evaluations in Chapter 4; finally, conclusions of our works and discussion of the future works are listed in Chapter 5.

Chapter 2

Background

2.1 Light Field Technology

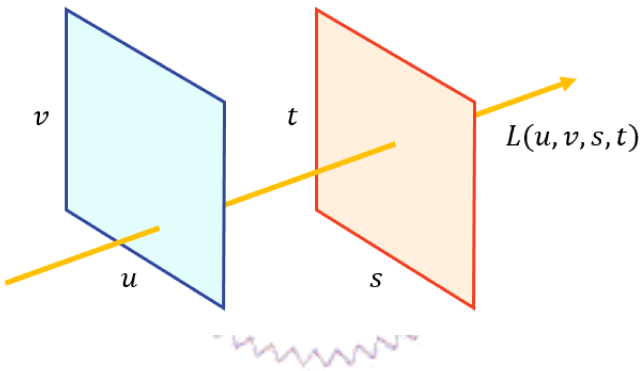


Figure 2.1: The light field in 4D data format, with UV plane as angular coordinate system and ST plane as spatial coordinate system.

Being one of the long researched topics, light field technology (LF), as the name suggests, is designed to capture all the light information in the space. The concept is becoming more and more popular these years as the immersive media becoming one of the most trending industry. It is originally formulate as a 7-dimensional plenoptic function [66]:

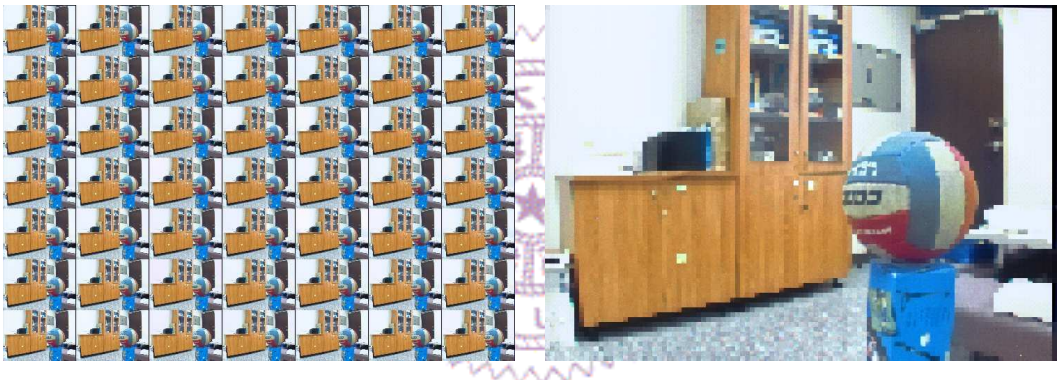
$$L(x, y, z, \theta, \phi, \gamma, t) \tag{2.1}$$

The formulation describes the amount of light flowing in every direction through every point in space, in which, (x, y, z) describe the point the light travels through in the 3D space, (θ, ϕ) indicate the direction (angle) the light is heading, γ shows the spectrum of the light, and t stands as the time stamp. While the function is quite complete, it is also burdensome and unrealistic for real-life usage. For instance, γ is unnecessary since the spectrum range we human can observe is limited; also, t is not a concern for every

image/video taken contains its time stamp, and we cannot record light information of all times. After simplification and modification, the LF function can be shown as a 4D function [46, 66]:

$$L(u, v, s, t) \tag{2.2}$$

4D LF is defined as radiance along with rays in space, which is a sense turns the light (x, y, z, θ, ϕ) into a ray shoot through two planes - UV plane and ST plane As shown in Fig. 2.1, each light information is considered as two coordinates on two parallel planes. Among them, UV plane, which is the first plane the light pass through, represents the angular coordinate system, and another one, ST plane, represents the spatial coordinate system. With this structure, lights from specific directions (the ones facing the ST plane) can be captured. In this way, while the scale of LF may be limited, its accessibility and practicality increase a lot, which leads to a bunch of useful applications, such as virtual aperture synthesis, image scene refocusing, and scene depth estimation.



(a) (b)

Figure 2.2: LF image representation. (a) present all spatial coordinates with the fixed angular coordinate; (b) present all angular coordinates with the fixed spatial coordinate.

Since an LF image contains 4D light volume information, there are two ways of inspecting the data: (i) present all spatial coordinates with a fixed angular coordinate, and (ii) present all angular coordinates with a fixed spatial coordinate. In the first format, an LF image can be represented as a 2D array (angular resolution) of 2D images (spatial resolution), as shown in Fig. 2.2(a). For each angular coordinate (u, v), it records all the light information coming from the same direction, which can then form as an image taken with the corresponding angular distance and in size of spatial resolution, where the pixels are the spatial coordinates. As for the second format, a tile is formed in the same size of angular resolution recording the information from the same spatial coordinate for each spatial coordinate (s, t), which is shown as Fig. 2.2(b). The image shown in the first format is also called sub-aperture (sub-view) image, it is a common and efficient way for the researchers to access the LF data.

The light field consists of an enormous amount of data for it collect the data from multiple points. So how do we capture such amount of data at the same time? Empirically, there are two ways of acquiring the data. One of them is the micro-lenses camera system, this method is widely used and can acquire a small scale of the light field. Another one is the camera array system, while it can be expensive in both design and establishment, it can acquire the LF data on a large scale. The features and the applications of these two acquisition methods will be further introduced in Sec. 2.2 and Sec. 2.3, respectively.

2.2 Micro-lenses Camera System

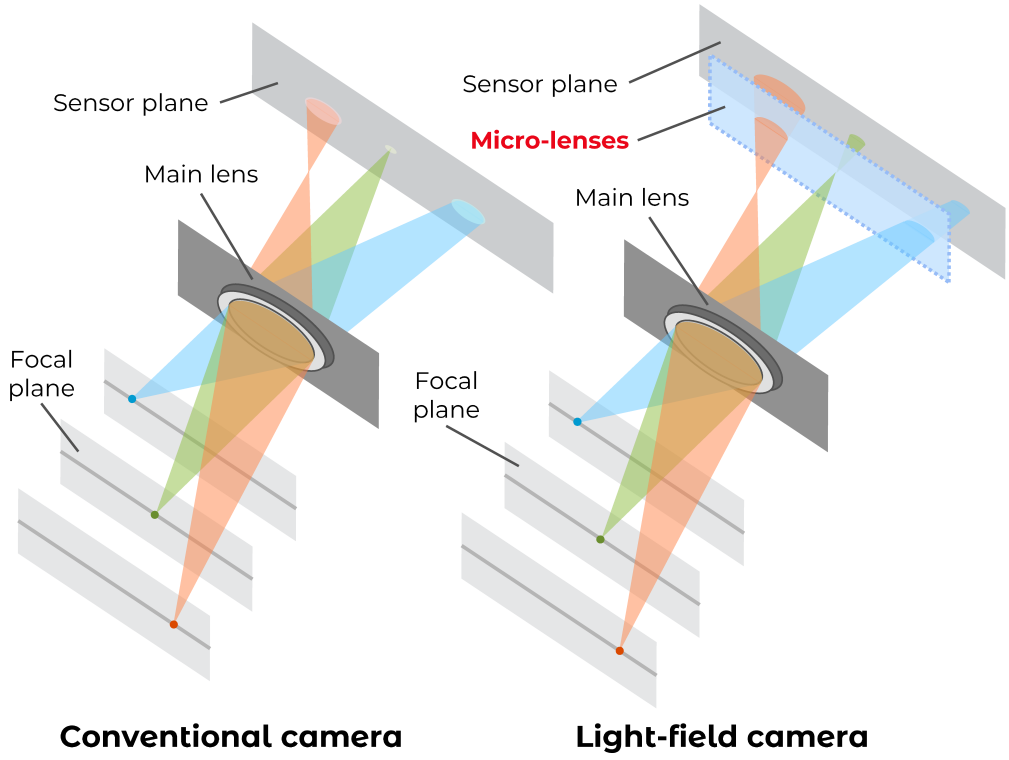


Figure 2.3: Comparison between conventional camera and light field (plenoptic) camera. The main difference is the micro-lenses placed between the main lens and the image sensor.

Microlens array system is a delicate yet straightforward way of collecting LF data; it is long researched and developed over the years. From the simple image sensor array [67, 59, 65, 61] to the commercial product [5, 18], micro-lenses system can capture the light volume in the space with a single camera lens. Precisely, in the camera, a micro-lens array is placed between the primary lens and image sensor to effectively disperse the lights so the angular information can be obtained [47, 50, 68], this kind of special camera is also

called *plenoptic camera*. Like Fig. 2.3 shows, the main difference between a conventional camera and a plenoptic camera is the lenses in front of the image sensor, it is used to split the lights into multiple perspectives and multiplex the lights onto the sensor. Practically, the UV plane represents the focal plane of the camera, and the ST plane can be viewed as the image sensor plane.

The system only uses one camera lens for data acquisition, resulting in a comparably small scale of LF. The disparity between two sub-aperture images is small, and hence the overlapping part is significant, which is suitable for the application like depth estimation, and image refocusing. However, the limited image sensor size also leads to the trade-off between angular and spatial resolution. Since LF images require 4D data, compared to the regular 2D image, the resolution of the captured scene (spatial resolution) is several times smaller. Let's say we have a 4K LF image, whose size is 4000×2000 pixels, and its angular resolution is 10×10; this results in the size of 400×200 of spatial resolution, which is relatively low in modern standard, not to mention that we may want to view it with HMD. Many researchers have tried to tackle the problem (spatial super-resolution), including applying interpolation to the EPI solpes [39, 64], using a regular DSLR camera as pixel supplement [28, 62]. These solutions are practical and have lots of potentials to expand the scale of an LF image.

(a) (b) (c)

Figure 2.4: Commercially available plenoptic cameras: (a) Original Lytro LF camera [1], (b) Lytro Illum camera [3], and (c) RayTrix R11 camera [2].

Commercial plenoptic camera manufacturers, such as Lytro Inc. and RayTrix Inc., also utilize this micro-lenses technology in their products. Fig. 2.4 shows the camera models that are commercially available now, including the original Lytro camera [1], Lytro Illum camera [3], and RayTrix R11 camera model [2]. Among them, the resolution of the Lytro Illum cameras is up to $15 \times 15 \times 434 \times 625$ pixels, which is the state-of-the-art LF camera technology. However, due to lengthy process latency, the cameras can only shoot the LF video in 3 fps.

2.2.1 Depth Estimation

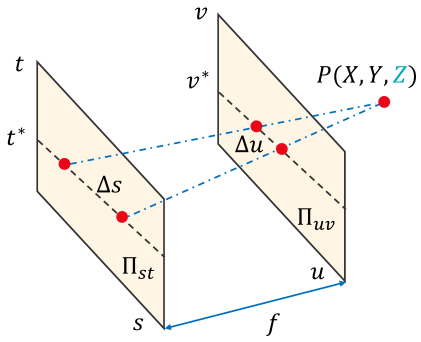


Figure 2.5: The depiction of epipolar plane extraction of LF data. By fixing (t, v) values, the epipolar slope of point P on SU epipolar plane is $\Delta s/\Delta u$.

Depth estimation is one of the main features of the micro-lenses camera system [42, 69, 57, 43]. The feature is crucial in many kinds of advance processing (such as 3D scene reconstruction) and has been researched for a while. The necessary process procedure is described as below [27].

Thanks to the slight viewpoint difference and the high overlapping part between the sub-aperture images, we're able to depict the epipolar plane, by fixing (t, v) coordinate or (s, u) coordinate. As shown in Fig. 2.5, the goal is to find the depth value Z of an arbitrary 3D point P. First, we fix the (t, v) coordinate to get SU epipolar plane. Then we extract the epipolar slopes, which are indicative of the depths of the different objects in the scene, with the edge detecting technique to the SU epipolar plane. These slopes can then induce the depth values throughout the scene with the following function:

$$\begin{aligned} s_Z &= \frac{Z}{Z-f}, \\ Z &= \frac{s_Z \times f}{s_Z - 1} \end{aligned} \tag{2.3}$$

Let s_Z be the epipolar slope of the corresponding depth value Z, and let f be the focal length of the camera. Thus, the Z value is the depth value of point P in 3D space, and it's the depth estimation result we want.

2.2.2 Image Refocusing

Image refocusing is another important application for the micro-lenses camera system, we can even say that the plenoptic cameras exist for this feature to some extent. There are several ways of achieving the refocusing effect to an LF image, some by splitting the foreground and background of the image [25, 70], others by designing the volumetric filter

applied in frequency domain [35, 40]. Among them, the easiest and the most effective one being the shift-sum algorithm proposed by Ng et al. [50], The method refocuses the LF images based on the target epipolar slope in the spatial domain and will be explained in the following paragraph.

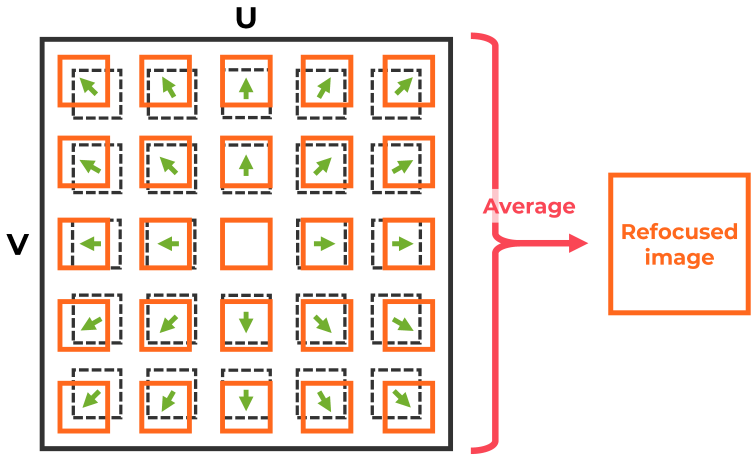


Figure 2.6: The procedure of the LF shift-sum algorithm proposed by Ng et al. [50]

As shown in Fig. 2.6, the algorithm is executed in three steps:

1. **Calculate the pixel-shifting number for each sub-aperture image in an LF image according to its angular coordinate, the number doesn't necessarily to be an integer.** The pixel-shifting number for each sub-aperture image is different, and the shift is in both U and V axis. From Eq. 2.3, we get the epipolar slope s_Z of any depth Z in the scene, and thus we calculate the pixel-shifting number (u_{shift}, v_{shift}) for the sub-aperture image located at angular coordinate (u, v) with the following function:

$$\begin{aligned} u_{shift} &= U \times s_Z \times \left(\frac{u}{U-1} - \frac{1}{2} \right), \\ v_{shift} &= V \times s_Z \times \left(\frac{v}{V-1} - \frac{1}{2} \right) \end{aligned} \tag{2.4}$$

2. **Shift all the sub-aperture images with the corresponding shift number pixel-wisely.** The process is essential for it aligns the lights in the focused area and then disperses those in the defocused area. Note that the shifting number may not be an integer; we use the interpolation technique for such condition.

3. **Get the average image of all sub-aperture images.** This process can calculate the stacked light information of the specific depth in the scene, and thus the focused area becomes apparent, and the defocused area becomes vague. Eventually, the stacked (average) image is the refocused image we want.

From the procedure, we can see that the computation of the algorithm is quite substantial: each pixel requires an interpolation process. The computation loading results in significant process latency, especially for images with ultra-high resolution.

2.3 Camera Array System

Compared to the micro-lenses camera system, the camera array system aims to capture light information on a much bigger scale, usually the whole 360° scene, this makes it more suitable for the VR environment establishment. The alignment of the cameras is different based on the user scenarios, it can be a simple straight line, a shape of a sphere, or just randomly scattered around the space. However, more cameras lead to a more expensive and bulkier setup and more data that needs to be processed. Therefore, more cameras don't necessarily mean a better quality of LF, the alignment of the cameras is all that matters. How to design a proper camera array system is an essential topic for setting up the system, we need to optimize the scene coverage of each camera and the dense (overlapping part) of the light information.

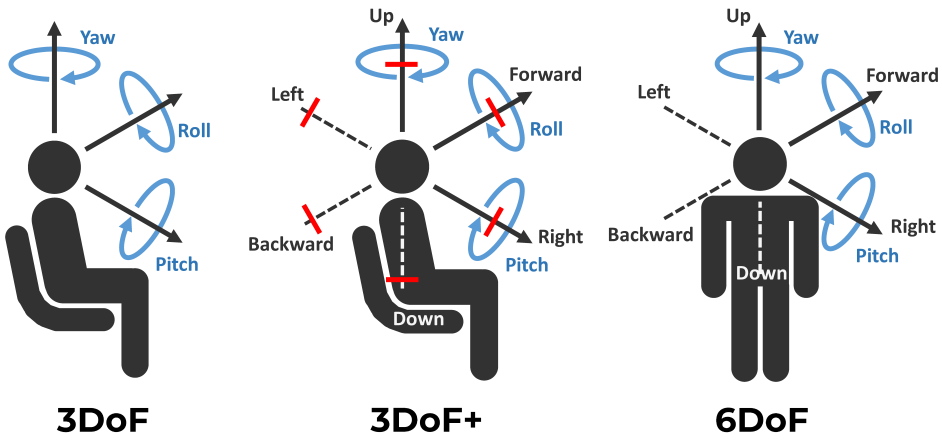
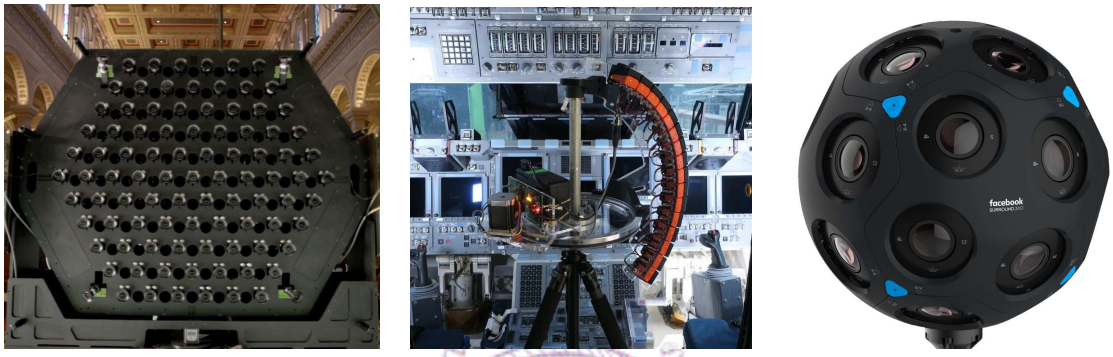


Figure 2.7: The standardization concepts of VR media proposed by MPEG-I group: 3DoF (left), 3DoF+ (center), and 6DoF (right).

Thanks to the multi-view property of camera array LF, it allows the users to move between cameras (views) and see the corresponding scene with view synthesis technique while wearing HMD devise, which is impossible for the traditional VR experience. To better standardize this new kind of VR mode as well as the future development of immersive media, MPEG-I group have proposed a scheme in 2017 [63] that divided the VR development into three phases: 3 Degree of Freedom (3DoF), 3DoF+, and 6DoF, which are shown in Fig. 2.7. Compared to the traditional 360° media, which provides only 3DoF (yaw, pitch, and roll of head rotation) experience, 3DoF+ and 6DoF enable the

view change from a different viewpoint to increase the immersion. The main difference between these two phases is the scale of the moving range. While 3DoF+ focuses on the experience when sitting still and enables a small range of head movement, 6DoF involves body moving, providing a much broader view transition scale, and allowing users to walk around in a 3D space. To achieve such a view transition, multi-view 3D videos, which consist of textures and depth maps from multiple viewpoints, are needed. Without a doubt, the camera array system LF is the perfect content provider for 3DoF+ even 6DoF VR.



(a) (b) (c)

Figure 2.8: The research product of camera array system in the market: (a) Lytro Immerge 2.0 [6], (b) Google LF VR arc system [11], and (c) Facebook Surround 360 VR camera [9].

Several state-of-the-art LF capturing structures from industries have also been released to the market as shown in Fig. 2.8. Among them, the Lytro Immerge 2.0 camera aims to shoot the cinematic level of LF; there are 95 lenses installed, resulting in 10K resolution per eye [6]. To better research 6DoF VR, Google released a camera rig to capture the light information omnidirectionally by rotating the arc structure, which is attached with 24 GoPro cameras. Also, Facebook designed and assembled a 360° camera with 24 lenses installed [11], allowing users to create their own 6DoF VR content [9].

2.3.1 View Synthesis

View synthesis is a technique for synthesizing target view (scene) based on the parameter of the reference views, such as positions, rotations, FoVs [31, 24, 48, 58]. Also, the technique is the core of 3DoF+ and 6DoF VR experience for they need to update the view in HMD based on the users' viewpoint in each frame.

Fig. 2.9 shows the process of view synthesis, in which the green cameras being the reference views, and the red one being the target view. From the figure, we see that the process includes two steps: image warping and blending. In the image warping part, the

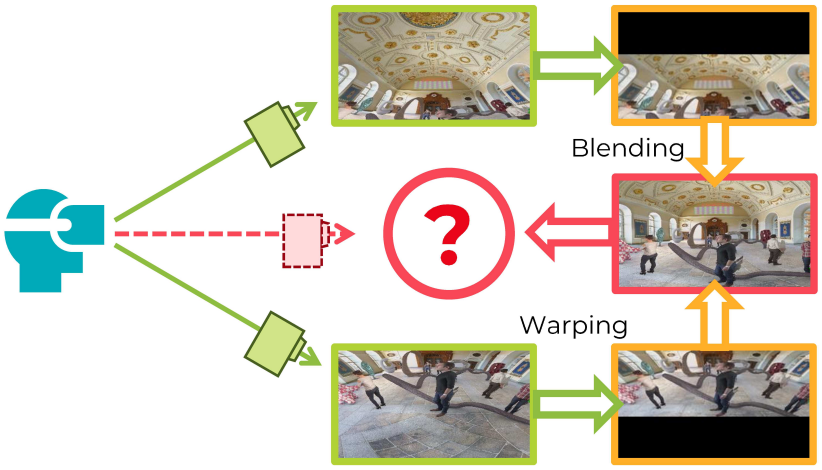


Figure 2.9: The process of view synthesis. We first warping the reference view to target view's perspectives based on the camera parameter. Then we blend the warped images together and get the synthesis result.

reference views are warped from their viewpoint to the viewpoint of the target view. The warped view is shown as the yellow images in the figure, there are some black regions in the image because they're out of the coverage of the reference view. With the warped images, we can blend them based on the geometry relationship and the color coordination of the image. The final blended image is shown as the red image in the figure, which is the result of the view synthesis. Nevertheless, there are cases that the coverage from the reference views is not complete enough, the inpainting technique is used to deal with the problem. The concept of the inpainting technique is to fill up the hole (uncovered region) in an image utilizing the pixels around the hole. There are several ways of doing it, from simple color interpolation to Generative Adversarial Network training, depends on the researcher's taste. Either way, the final inpainted image is the result of view synthesis.

Chapter 3

Auto-Refocus VR System

Our proposed system aims to render the panorama images with proper DoF on HMD based on the user's eye gaze. That is, our system refocuses the panorama image scene to match the DoF of objects we lay our eye gaze. Since the video playback is sequential (ideally $\geq$ 30fps), to achieve smooth video playback, The whole rendering process, including LF refocusing process and panorama rendering, needs to be real-time and accurate to keep the QoE level. We implement the system on the Unity engine, which is a gaming engine that supports FOVE rendering and provides comfortable usage.

3.1 System Overview

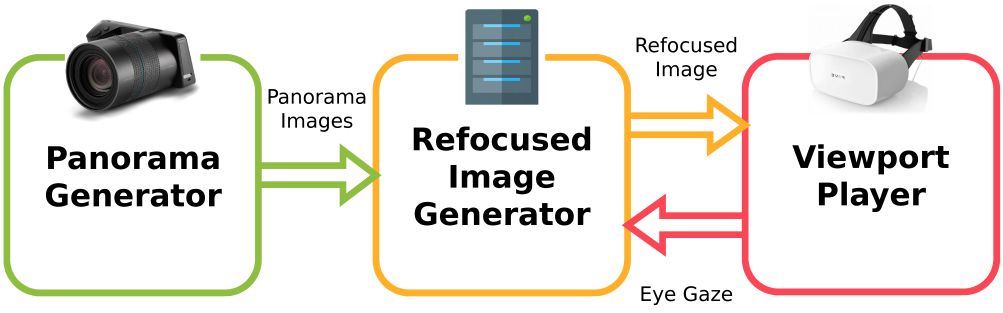


Figure 3.1: The proposed system consists of three components: panorama generator, refocused image generator, and viewport player.

As shown in Fig. 3.1, the proposed system is composed of three components: (i) panorama generator, (ii) refocused image generator, and (iii) viewport player. The panorama generator is in charge of the calibration and stitching of the LF images to form an LF panorama image. The refocused image generator manipulates LF images for refocused

panorama with the proper DoF. The viewport player connects to an eye-tracker equipped HMD to show viewport to the user. More details on these components are given below.

3.1.1 Panorama Generator

This component collects the LF images from an LF camera and stitches them into LF panoramas using stitching methods, such as the one proposed in Birklbauer et al. [26]. The resulting panorama images are sent to the refocused image generator. In addition to panorama LF images, the panorama generator also *pre-renders* some panorama images at different depth of fields for the refocused image generator. These pre-rendered images reduce the refocusing workload at the runtime, which is presented next.

3.1.2 Refocused Image Generator

This component is meant to generate refocused images in real-time. The component generates images according to the depth map and the coordinates from the eye gaze. During the process, it extracts the depth value of the eye gaze coordinates and then calculates the epipolar slope. With that slope, the image refocusing can be done using the shift sum algorithm. We note that the generation of refocused images is the most *demanding* process that must be done *on-the-fly*. Therefore, the refocused image generator is heavily optimized, as presented in Sec. 3.2.

3.1.3 Viewport Player

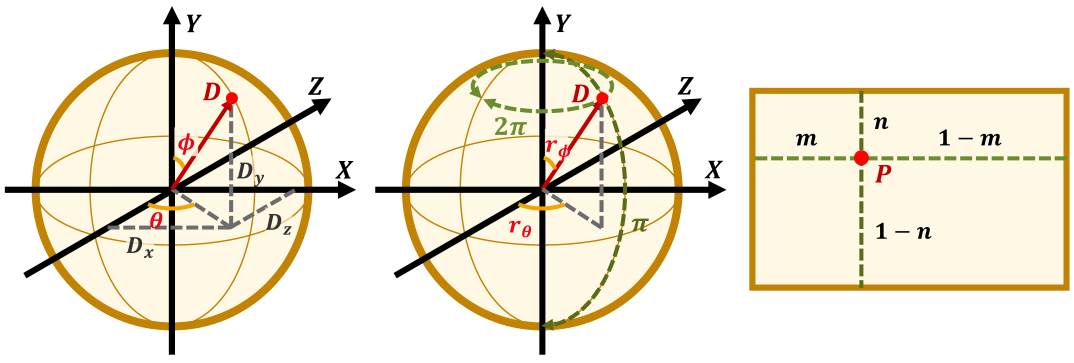


Figure 3.2: The procedure of translation from eye gaze vector, $D(x, y, z)$ in the leftmost figure, to panorama coordinate, P in the rightmost figure.

This component is primarily a video player that projects refocused panorama images to viewports for the HMD. That is, the panorama images are applied to a *sphere*'s in-

ner surface that wraps around the user, while the viewport is projected from that sphere. Moreover, this component also collects the 3D eye gaze vector $D(x, y, z)$ from the eye-tracker equipped HMD during the playback, and translates it to the 2D coordinates $P(x, y)$ on the panorama image. To do so, we first get the radius of yaw angle r_θ and pitch angle r_ϕ of the vector in the sphere with the following equations:

$$\begin{aligned} r_\theta &= atan(D_z/D_x) + \pi/2; \\ r_\phi &= acos(D_y). \end{aligned} \tag{3.1}$$

Next, we calculate the scale of P on panorama, (m, n), by dividing the radius to the range. Since the range of θ is 360°, r_θ needs to be divided by 2π, which gives:

$$m = \frac{r_\theta}{2\pi}, n = \frac{r_\phi}{\pi}. \tag{3.2}$$

Last, we get coordinates P by multiplying (m, n) with the height and width of the panorama, that is:

$$P_x = m \times W_{img}, P_y = n \times H_{img}, \tag{3.3}$$

In the formula, W_{img} and H_{img} stand for the width and height of the panorama image. The resulting coordinates P are sent back to the refocused image generator as the latest eye gaze for upcoming refocused images.

3.2 Optimization Methods

The latency of the image refocusing should be as low as possible for smoother focal transition and better user experience. We propose two optimization techniques to achieve that. Fig. 3.3 presents the detailed design of the optimized focused image generator.

3.2.1 Pre-Rendering Image Selection

As previously mentioned, we generate some pre-rendered panorama images with different DoF, so they can be quickly *retrieved* instead of *generated* from scratch. More specifically, we have the panorama generator render several images at different DoFs, so that the proper pre-rendered images can be selected at significantly reduced time complexity. The real challenge here is to choose the right DoFs so that the pre-rendered images are used more frequently.

We choose the target object depth using the depth map in our system, where the depth values are discrete. Therefore, we may pre-render images of some depths in advance and choose the closest one as the rendered result. In this way, the computing time can be

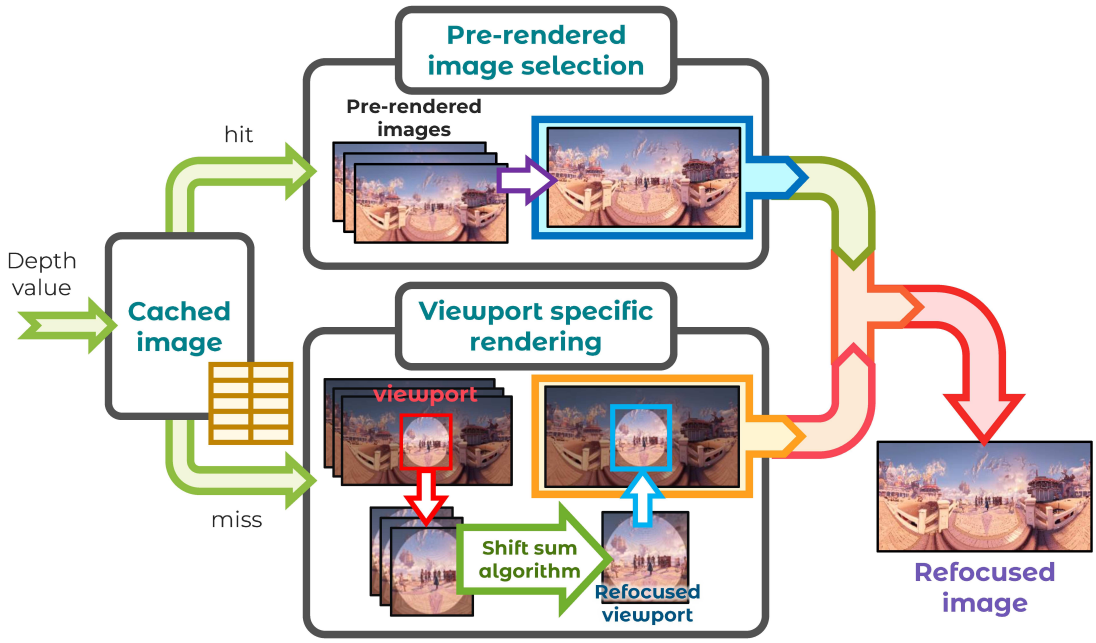


Figure 3.3: Our optimized refocused image generator.

dropped significantly, but with the cost of more memory usage. One way to mitigate this issue is to apply to cluster on the depth map. The results represent the majority depth values in the LF image, which can be used as the candidate DoFs of the pre-rendered images. Another way to choose DoFs is based on the objects in the scene for they are more attractive to the users. Two approaches can be taken to find the objects: learning-based object recognition or manually tagged objects. Both approaches give the candidate DoFs. Among all candidates, we choose N DoFs and cache N pre-rendered images, as shown in Fig. 3.3. At runtime, the input depth value is first matched against the cached DoFs. If we hit the cache, the pre-rendered image selection (top of the figure) is triggered; otherwise, the viewport specific rendering (bottom) is invoked.

3.2.2 Viewport Specific Rendering

To reduce the number of pixels from the panorama image to be rendered, we only process the pixels seen in the HMD viewport. Due to the limited scope of the viewport (about $100° \times 100°$), the HMD viewport is about 15% of the whole panorama image. As shown in Fig. 3.3, the brighter area in the panorama represents the viewport, and we crop it before rendering to reduce the time complexity. More specifically, we first determine the viewport center coordinates, which in our case are given by the HMD. We perform the refocusing process only on the viewport. We then paste the rendered viewport onto the original panorama image for the final result.

(a)

(b)

Figure 3.4: Sample results from our proposed system: (a) a full panorama LF image of our lab and (b) the corresponding depth map.

3.3 Evaluations

We carry out experiments and a user study considering both *objective* and *subjective* perspectives. In the objective evaluations, we mainly want to see the performance of the system and the improvement due to the optimization technique. As for the subjective evaluations, our user study compares the user experiences between light field panorama with refocusing and the normal panorama, in terms of MOS.

3.3.1 Implementations

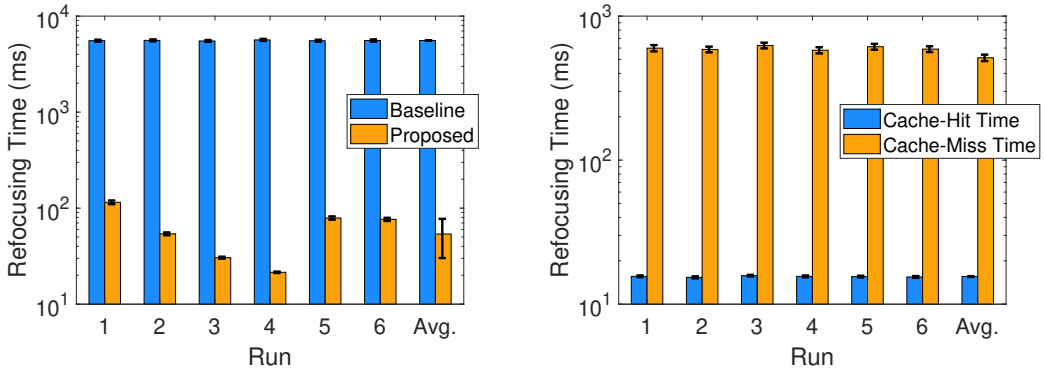
We implemented the auto-refocus HMD VR system in C# based on the Unity engine [22]. We use CSMatIO [7], a C# library to load LF images written in Matlab *mat* format. We adopt OpenCVSharp [17], which is an OpenCV [16] framework with .NET wrapping to process LF images. We prepare the LF images with a Lytro Illum camera [5] and decode the raw data with Matlab LF toolbox [34] to get 5D LF images. These 5D LF images are then stitched into a panorama image [26].

Sample stitching results from our system are shown in Fig. 3.4, where Fig. 3.4(a) being the panorama whose resolution is 3840×418, Fig. 3.4(b) being the depth map of the panorama.

The resulting system is installed on a testbed for our evaluations. The testbed is comprised of a $7 \times 7 \times 1920 \times 3840 \times 3$ LF panorama image captured in our lab. The LF image has a size of 309 MB and is shown as Fig. 3.4. We employ FOVE HMD. The system runs on a PC with an Intel i7-2600 CPU and 16 GB memory.

Table 3.1: The Average (Standard Deviation) Refocusing Time in Millisecond

Baseline	**Proposed**	**Cache Hit**	**Cache Miss**
5691	27.16	17.8	634.6



(a) (b)

Figure 3.5: The average refocusing time of the systems in 6 runs, the 7^{th} bar shows the average time of the 6 runs: (a) the refocusing time of the baseline and proposed systems, and (b) the cache-hit and cache-miss refocusing time of the proposed system.

3.3.2 Objective Measurements

Setup. We conduct the experiments using our VR HMD system. Several system parameters can be adjusted in our experiments:

- *Depth tolerance* ϵ, which determines the maximal depth difference between a refocusing request and a cached pre-rendered image. A larger ϵ value increases the cache hit ratio but reduces the QoE due to DoF mismatch.
- *Pre-rendered cache size* N, which is the number of cached pre-rendered images.

The default value of ϵ is 0.005, and that of N is 20. We run our experiments for six runs. In each run, we randomly pick 50 sample pixels in the panorama LF image and then use their depth values to generate the refocused images. The same sample pixels are fed into our proposed VR HMD system and a *baseline* system, where none of the optimization technique is adopted. For each refocused image, we consider the following metrics:

- *Hit rate*, which is the cache hit ratio of the pre-rendered image, showing the effectiveness of the cache.
- *Cache-hit time*, which is the running time of selecting the pre-rendered image from the cache. It is relevant when a cached pre-rendered image is within the depth tolerance of the depth of the refocused image.
- *Cache-miss time*, which is the running time of rendering viewport. It is relevant when no pre-rendered image is within the depth tolerance.
- *Refocusing time*, which is the time to generate a new refocused image.

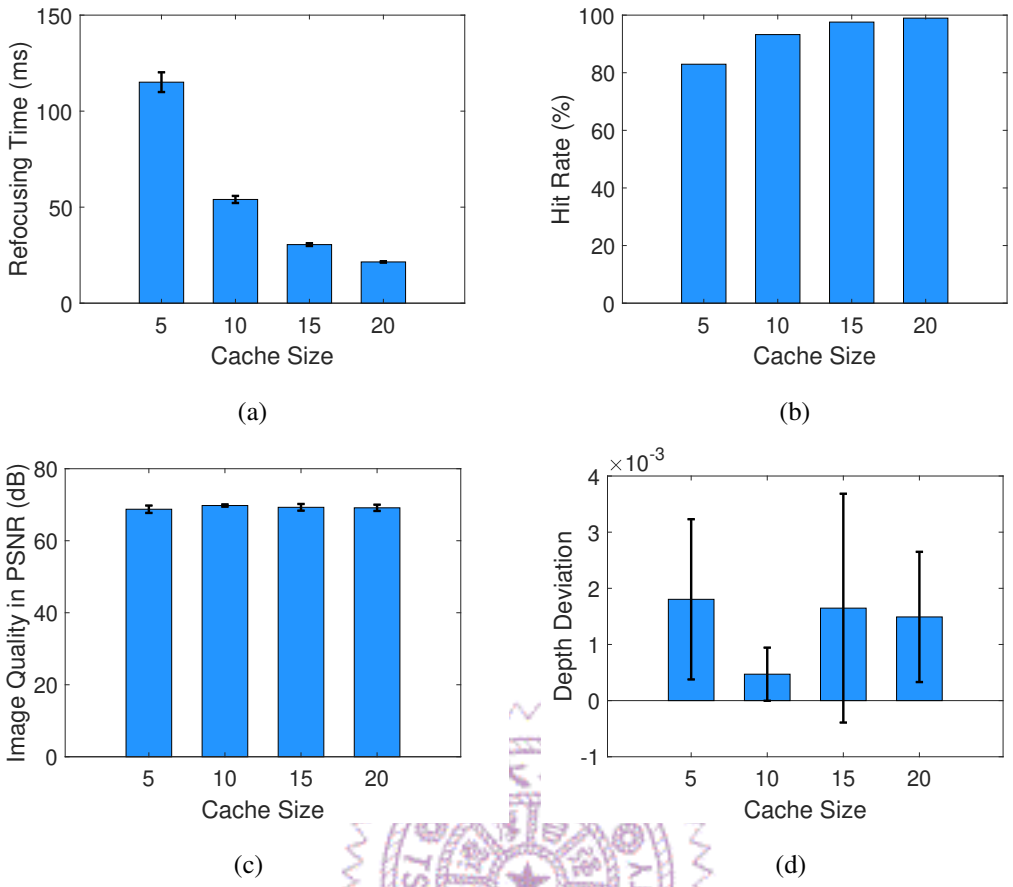


Figure 3.6: Results under different cache size N: (a) refocusing time, (b) hit ratio, (c) video quality, and (d) depth deviation.

- *Depth deviation*, which is the deviation of the depth of a refocused image from the ground truth.
- *PSNR*, which is the video quality of the refocused image compared to the ground truth.

We report the average results and give 95% confidence intervals whenever applicable. Last, we vary the system parameters $N \in \{5, 10, 15, 20\}$ and $\epsilon \in \{0.005, 0.001, 0.0005\}$, to study their impacts on the system performance.

Refocusing time. Fig. 3.5 shows the average refocusing time for 50 pixels in 6 runs, with the 7^{th} bar being the average time of the six runs. From Fig. 3.5(a), we can see that the refocusing time of the baseline system is far longer than that of the proposed system, showing that our optimization techniques significantly reduce refocusing time. Fig. 3.5(b) reveals that the refocusing time of both pre-rendered image selection and the viewport specific rendering. Between them, the cache-hit refocusing time is much lower because we only switch the images when the cache is hit. Although viewport rendering takes longer time than image switching does, it is still about 6.7 times faster than the

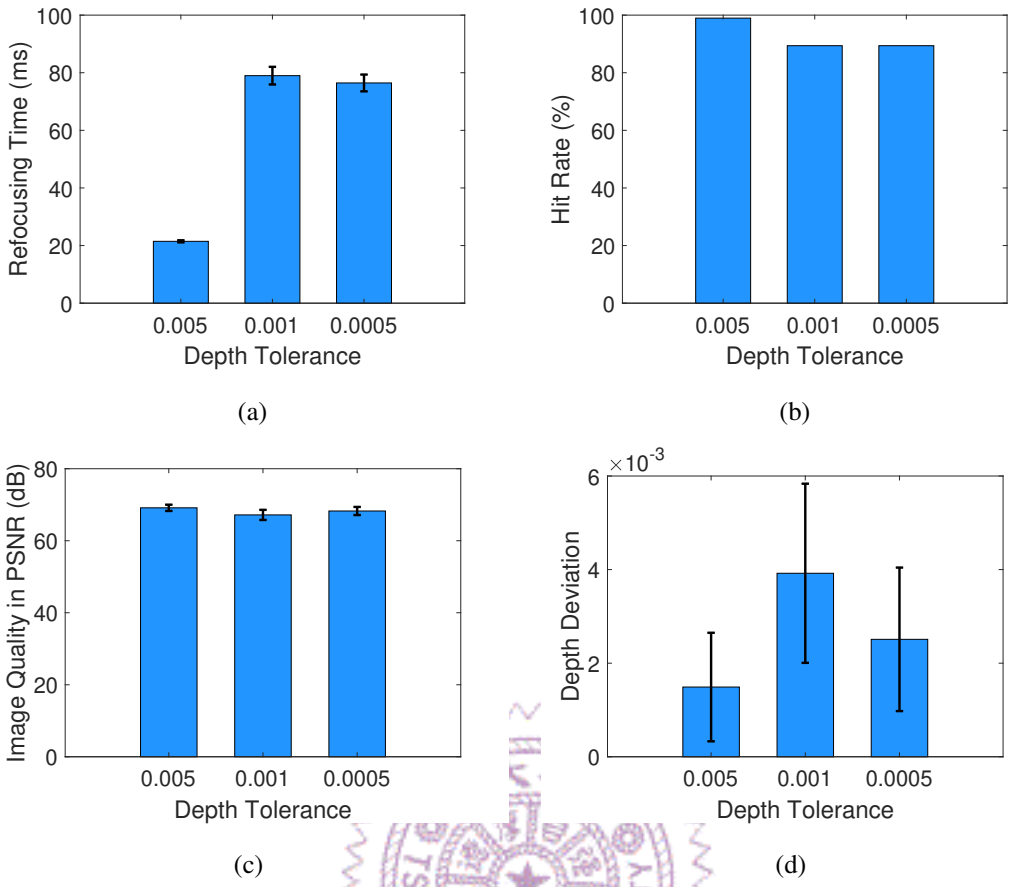


Figure 3.7: Results under different depth tolerance ϵ: (a) refocusing time, (b) hit ratio, (c) video quality, and (d) depth deviation.

baseline.

Implications of pre-rendered cache size. Fig. 3.6 reports the results under different cache sizes N. From the figures, we can see that as N becomes larger, the hit rate is higher and the refocusing time is faster because there are more candidate images for the requests. As for the image quality, it does not change much since the depth tolerance remains the same. Finally, the depth deviation is tiny, indicating the effectiveness of the depth selection method, which is k-means clustering in our implementation.

Implications of depth tolerance. Fig. 3.7 shows the results under different depth tolerance ϵ. First, we can see that the hit rate is lower as ϵ becomes lower and hence leads to the raise of the refocusing time because of the decrease of the cache hit. Moreover, small ϵ means the selected depth is closer to the target depth, leading to better image quality. Typically, the depth deviation will grow with ϵ, whereas in our case, the diversity of the depth values is small regardlessly, and hence we do not see this in the figure.

3.3.3 User Study

The user study aims to find out different user experience between the panorama playback system with refocusing mechanism and the system without it, also, to determine the impact of the DoF factor on the VR experience. We use MOS to see the general feelings of the system provided by multiple users, and the questionnaire includes two questions: (i) *do you notice the difference between two panorama images?* and (ii) *how much do you like the images?* Each user watches the same panorama image for 2-min in each system. The order of the two systems is random. For the first question, a user briefly describes the difference he/she feels during the experience. As for the second question, we use a MOS score ranging from 1 to 5, in which the higher number indicates the user has a better experience and is more fond of the video.

We hold a small user study with ten users; all of them are graduate students in their 20's. We include both male (7) and female (3) users. We have two main findings:

- All of them tell the difference between the two systems.
- The MOS scores from our proposed system are on average of 2.78; while those from the baseline system are on an average of 2.33, which is 0.45 lower.

Our user study, although preliminary, reveals the potential of refocusing supports with LF panorama images in VR HMD systems.

Chapter 4

3DoF+ VR System

4.1 System Overview

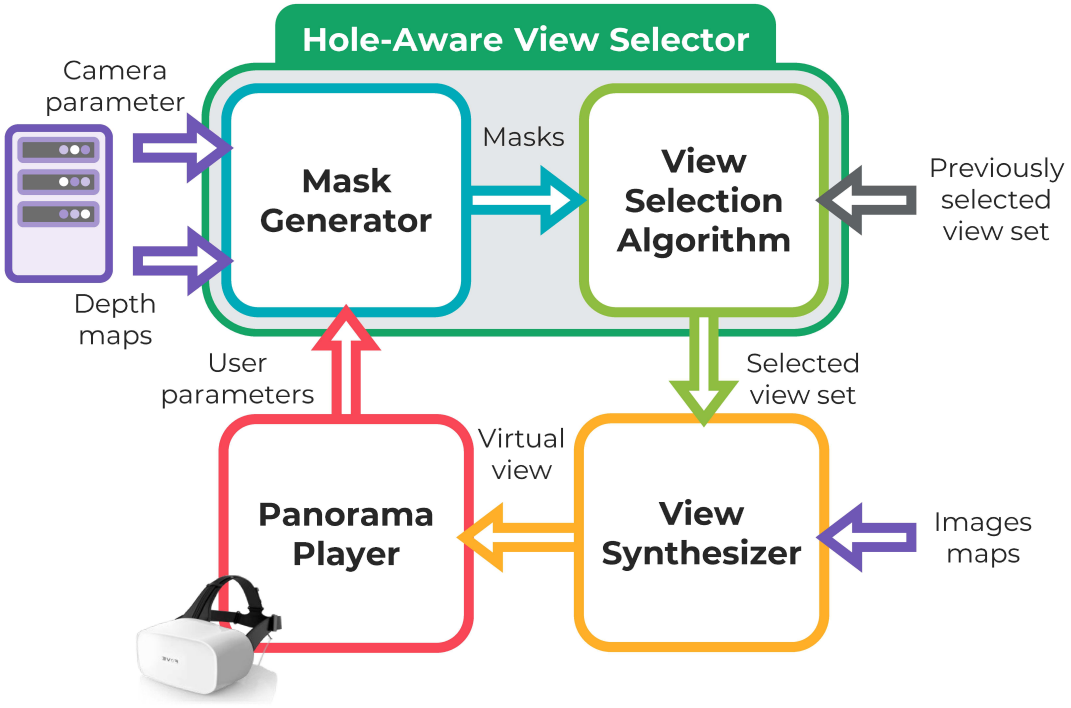


Figure 4.1: The architecture of the 3DoF+ VR system. 3 components are included: hole-aware view selector, view synthesizer, and panorama player. Among them, hole-aware view selector can be split into two components: mask generator and view selection algorithm.

This system aims to provide users a novel VR experience that allows them to move their positions with the corresponding viewpoint transition of the scene in HMD. To be

specific, the view of a virtual viewpoint (user's viewpoint) is synthesized in each frame with the reference camera parameters. One of the biggest challenges is the latency of view synthesis process, which is highly dependent on the number of reference views. Thus a good view selection algorithm is required to reduce the reference view number while maintaining the synthesis quality at the same time. To maintain the quality of synthesis result, we need to get the 3D space information of the reference views, so the issues like view coverage, object occlusion can be resolved.

The architecture of the system is shown as Fig. 4.1. As we can see, There are mainly three components in the system, including (i) hole-aware view selector, (ii) view synthesizer, and (iii) panorama player. Among them, the hole-aware view selector can be further split into two sub-components: mask generator and view selector. The former component is in charge of extracting 3D space information of each reference view, and the later one is in charge of selecting the view set based on the extracted information, View synthesizer synthesizes the image of the virtual view based on the selected reference view. Panorama player transfers the virtual view image to the HMD device and also collect the user position data for the future view selection. More details of the components are given below.

4.1.1 Hole-Aware View Selector

This component is responsible for the selection of view set that is used for view synthesis. To make a suitable selection, the camera parameter of the reference views and the target (user) view is required to get the geometry relationship between them. Besides, to extract the space information for a more accurate selection, the depth maps of the reference views are necessary as well. Moreover, to keep the consistency of the synthesized view throughout the frames, the selected view set from the last frame is also needed. Finally, the output of the component is the desired view set for view synthesizer, of course.

The desired view set should have a limited number of views, and their synthesis result should be as high quality as possible, also, the selection latency should be as low as possible. Specifically, the component ensures the effectiveness as well as the efficiency of the view selection in our system, leading to the two sub-components: mask generator and view selection algorithm. To keep the quality of synthesis (reduce the holes), mask generator extracts the space information of the reference views and generates the masks containing such information. With the masks, the view selection algorithm will then select the desired views for view synthesizer efficiently. To keep it efficient, instead of finding the optimal result with brute force computation, we utilize a greedy algorithm to approximate the optimal solution. This component is our main contribution to this system, and more details can be found in Sec. 4.2.

4.1.2 View Synthesizer

The component has only one purpose: to generate the image of the virtual (user) viewpoint with the selected reference views. The view synthesizer reads the view set from the view selector and gets the image of the corresponding reference views, based on the views, it then performs view synthesis as described in Sec. 2.3.1 to generate the target view from the user's perspective. After the synthesis, the image will be sent to the panorama player and shown to the user.

4.1.3 Panorama Player

The last component is the panorama player, which is a rather simple component that receives the image of the target view and shows it to the user in HMD. Another feature of the component is that it's in charge of collecting the user viewpoint data, including positions and orientations, and send it back to the hole-aware view selector as the next target viewpoint. There are several ways of collecting the user viewpoint data, including the depth sensor deployment, and motion capture. Among them, the easiest one is to find an HMD devise that support body movement detection, such as HTC Vive Focus [14], FOVE [10]. These HMDs can detect the relative transition as well as the orientation, the user viewpoint parameter can be derived with these data.

4.2 Hole-Aware View Selection

As previously mentioned, the proposed view selection method consists of two parts of processing: (i) mask generation and (ii) view selection. The former process produces the masks that contain the space information indicating which area in the target view is covered by the reference view. Specifically, a mask is in binary data format and the same size as the target view image, showing if the reference view covers the pixels in the target view. With these masks, the later process can select the desired view by checking which mask has the most view coverage (number of covered pixels). Since the number of the selected views is limited for reducing the computation loading, we utilize the greedy algorithm and select the currently optimal view in each stage.

Besides the greedy selection method, we also propose another selection method based on the pixel importance, which is a score of a pixel indicating the number of views covering it, the smaller the number, the more critical the pixel. The selection order is from the most critical pixels to the less important ones, also there several conditions for different conflict situations. The details of the selection methods are explained below.

4.2.1 Mask Generation

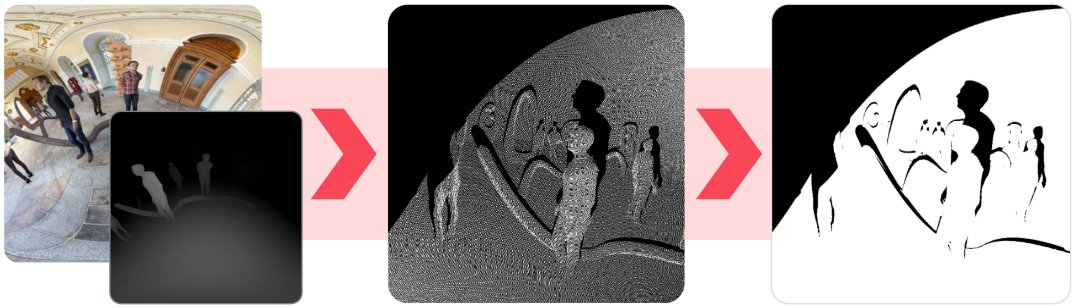


Figure 4.2: Mask generation process in the mask generator component, including two steps of process: 3D warping and hole filtering.

The goal of this process is to generate a binary mask for each reference view (or the candidate ones), and each mask shows the coverage of a reference view to the target view. Since the mask is in binary format, the value-1 pixels (white) are considered as the covered (by the reference view) pixels, and the value-0 (black) pixels are considered as the uncovered pixels. Fig. 4.2 shows the generation procedure of the masks, which includes two steps of the process: 3D warping and hole filtering. The left figure indicates the color and depth map of a reference view, which is taken with the real camera. The middle figure shows the resulting image of the 3D warping, where we transform the pixels into the target view perspective. The right figure presents the result after applying the hole filtering technique, which can determine the nature of the holes and fill up the right ones, and is also the final product of the mask generation process. In the following paragraphs, the detailed design of the process will be presented.

3D Warping

This technique is usually used for the computer graphic algorithm like view synthesis for it's capable of transforming a set of the 3D point cloud from a coordinate system to another [38]. That is, to see the scene from other viewpoints, we determine the view coverage and object occlusion with this technique. Typically, there are three steps of the process in the 3D warping technique:

1. **Unprojection.** This step aims to unproject the pixels of the 2D image to the 3D space to form the corresponding point cloud. The unprojection process is pretty straight forward, we first calculate the spherical coordinate (θ, ϕ) for each pixel based on the horizontal and vertical FoV, then we translate the spherical coordinate

to the corresponding unit vector (x_n, y_n, z_n). With the vector, we multiply it with the depth value from the depth map and can finally get a 3D coordinate (X, Y, Z). After all pixels in the image are unprojected, we can get a 3D point cloud of the scene, whose density is determined by the image resolution.

2. **Affine transform.** The second step is for the transform of the point cloud, from the reference view coordinate system to the target view coordinate system. For a classic affine transform, we have two affine spaces X and Y, and to do the transformation $f : X \to Y$, we use the function $y = ax + b$, where x and y are the vectors in two affine spaces, respectively, a is a linear transform term on space X, and is b is a vector in space Y. In our case, x represents the 3D point cloud from the pixel unprojection, y represents the transformed point cloud in the target view coordinate system, a represents a rotation matrix, and b represents a translation vector. The goal here is to calculate the transform term a and b with the equation below so we can do the transform.

$$\begin{aligned} a &= R_2^T \times R_1, \\ b &= -R_2^T \times (P_2 - P_1) \end{aligned} \tag{4.1}$$

In these equations, R_1 and R_2 are the rotation matrices of the orientations of the reference view and the target view; P_1 and P_2 are the 3D positions (vectors) of the reference view and target view. Now formula $y = ax + b$ is complete, we can transform the point cloud from the reference viewpoint to the target viewpoint.

3. **Projection.** The final step is to project the transformed point cloud back to a 2D image, the process is similar to the gaze translation mentioned in Sec. 3.1.3, which is the opposite of the unprojection. In the process, we first want to get the spherical coordinate (θ', ϕ') for each 3D coordinate (X', Y', Z') using Eq. 3.1. Then based on the spherical coordinate, we can calculate the normalized 2D coordinate with Eq. 3.2, only with different FoV values. Now we need to filter out the coordinates that are outside the [0, 1] range for they won't be shown in the image, and eventually, we get the real pixel coordinate in the image with Eq. 3.3. To make the binary mask, we set the pixels with the projected coordinates to 1 and 0 to the others.

The mask we produce requires lots of computations, and most of them are repeated in the view synthesis since 3D warping is also applied in this process. Therefore, part of the data can be recycled, such as a 3D point cloud of the scene for the later use.

Hole Filtering

In this second process, we need to fill up the holes caused by the point cloud discontinuity, which are the tiny holes scattered on the should-be-covered area in the central figure of Fig. 4.2. Therefore, two main steps are taken in the hole filing process, including hole recognition and pixel binarization. In the hole recognition, we recognize and categorize each hole (value-0 pixel) in the mask by checking the coverage density to see if it needs to be filled up. Then for each hole needed filling, we apply binarization to it with a threshold.

(a) (b) (c)

Figure 4.3: The hole filtering process of the mask. (a) The image before hole filtering process, (b) the types of the holes, (c) the image after hole filtering process.

In the Fig. 4.3, the procedure of hole filtering is presented. Within it, Fig. 4.3(a) represents the image from the warping result, which has lots of tiny holes on it and needs to be fixed. The nature of causing these holes is the discontinuity of point cloud during the warping. That is, the pixels on the image are discrete (the coordinates are integers), and hence the unprojected coordinates are discrete as well since there's no mesh formed. The result point cloud is comparably sparse for the image projection, and so the seams between the 3D points lead to the holes on the image. Since the selection is based on the space coverage (number of the value-1 pixel) of each mask, we need these tiny holes to be filled so the final result would not be affected.

Then how do we know which holes to fill? As Fig. 4.3(b) indicates, there are two types of holes: the view coverage limitation causes the red ones; and the green ones are caused by the point cloud discontinuity. As we can see, the most apparent difference between these two is the scale of the hole. Compared to the messy and unruly point

cloud discontinuity holes (green holes), the view coverage limitation holes (red holes) have clear outlines and complete structures. Since the holes in the green region are more like noises than the real structural ones, we categorize the holes based on their coverage density. To be specific, we inspect the neighborhood area of a hole (value-0 pixel) and see the number of holes in the very area. If the number is larger than a certain threshold, than the inspected pixel is determined as a view coverage limitation (red) hole; otherwise, a point cloud discontinuity (green) hole.

Table 4.1: Symbol table of hole filtering algorithm.

Symbol	**Description**
M	Mask that needs hole filtering process
k	Size of kernel used for summation convolution
d	Coverage density of a pixel
τ	Threshold to determine the hole type (binarization)

Table 4.1 shows the symbols used for hole filtering algorithm, in which M is the warped image like Fig. 4.3(a), k is the size of the kernel used for summation convolution, d is the coverage density that shows the number of covered pixels around a pixel ($k\times$ k area), and τ is the threshold of binarization, if any hole's coverage density d is larger than it, the hole will be classified as a green one and will be filled up.

Algorithm 1 Hole Filtering Algorithm

```
1:  // Assign convolution kernel
2:  K ← kernel filled with 1 in size of k × k
3:  for each pixel p in M do
4:      if p = 0 then
5:          P ← the surrounding k × k area of p
6:          // Assign coverage density
7:          d ← K ∗ P
8:          // Fill the hole if coverage density d is big enough
9:          if d ≥ τ then
10:             p ← 1
11: return M
```

Algorithm 1 shows the pseudo-code of the algorithm. As shown in the code, we first assign the convolution kernel K with all elements being value-1 at line 2, and then we conduct a convolution operation $M * K$. For each hole, we get a corresponding coverage density d to see the number of the covered pixels around it at line 7. Next, by comparing

d to the threshold τ, we can determine if a hole is a noise or a part of a complete structure. Finally, we fill up a noise-like point cloud discontinuity hole at line 9.

Fig. 4.3(c) shows the result after applying the hole filtering. As we can see, compared to Fig. 4.3(a), the holes in the green area are filled while the holes in the red area remain untouched. The filtered mask shows the correct view coverage information of a reference view.

4.2.2 View Selection Algorithm

Now the masks containing the space information are generated, the view selection can be performed. We propose a view selection algorithm to select a suitable view set with the information in the masks, and since the covered information in the mask is represented as the value-1 pixels, the view selection problem can be considered as a problem to find the combination of masks with the maximum view coverage (the mask union that contains the most value-1 pixels). Also, due to the restriction of the selected view number, the problem can be mapped to a classical combinatorial problem: Maximum Coverage Problem (MCP) [60, 23, 41]. To better solve the MCP in polynomial time, we utilize the greedy algorithm [23]. That is, to select the currently optimal combination of views with the most covered pixel.

Maximum Coverage Problem

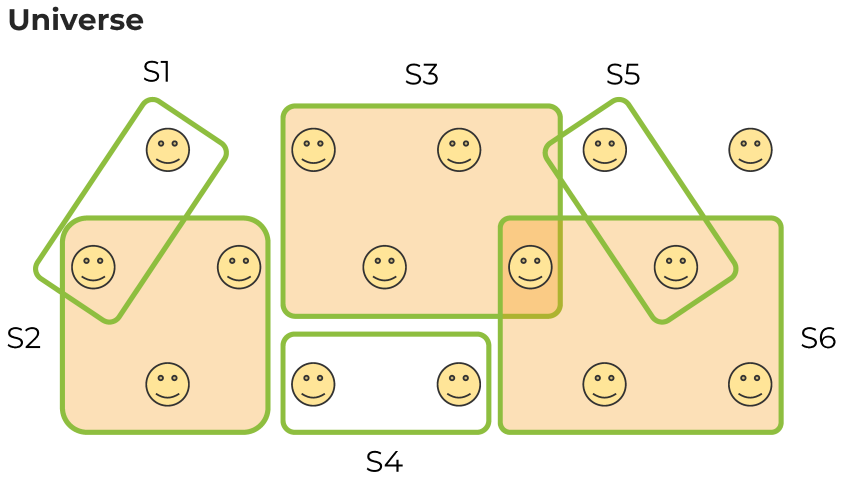


Figure 4.4: An example of the Maximum Coverage Problem (MCP): There are six sets that contain the elements in universe, and we want to find at most 3 sets to cover the most elements. The answer would be the collection [S2, S3, S6].

Maximum Coverage Problem (MCP) is a classical NP-hard combinatorial problem in

computer science and is widely taught in approximation algorithms. The basic principle of MCP is to select a certain number of sets that cover the most elements in the universe, A simple example is shown as Fig. 4.4, where there are 15 elements in the universe, and six sets that each covers a different part of the universe. If we want to find at most three out of six sets that cover the most number of elements, which sets should be chosen? The solution set of this example is [S2, S3, S6], which can cover ten elements.

So how do we map our view selection problem to MCP? First, we must define the sample universe, which in our problem is the pixel set in the target view image, each pixel in the image represents an element. Second, we need to define the sets that cover the elements, which in our problem is the covered (value-1) pixels in the mask of each reference view since those pixels represent the elements in the universe. Therefore, the mapped problem here would be to find a set of masks with limited mask number whose union can cover the most pixels in the target view image. Now with this mapped problem, we need to find an effective as well as an efficient algorithm to solve it.

Greedy Algorithm

Being one of the classical combinatorial problems, MCP has been researched year decades. Since the problem itself is *NP-hard*, there's no optimal polynomial-time solution for it until the day we prove $P = NP$. Over the years, researchers have been working on better approximation algorithms for MCP, and the best-possible algorithm they find is the generic greedy algorithm. To be specific, at each stage, we choose a set that contains the largest number of uncovered elements. The approximation ratio of the algorithm can reach $1 - 1/e \approx 0.632$, which is necessarily optimal under standard assumption [23]. To solve our view selection problem with the greedy algorithm, we symbolize the used terms and perform a complete formulation of the problem. Below are the details of our proposed algorithm.

Table 4.2: Symbol table of the view selection algorithm.

Symbol	**Description**
$\mathbf{M}$	Collection of the covered pixels in the masks
$\mathbf{S}$	Set of the selected views
C	Image used for union in each selection stage
$\mathbf{T}$	Union results of image C and all masks in $\mathbf{M}$
r	Coverage scores of $\mathbf{T}$
k	Maximum size of $\mathbf{S}$

The symbols of the greedy view selection algorithm are listed in Table 4.2. As we

can see, $\mathbf{M}$ is a collection of the covered (value-1) pixels in the mask of each candidate reference view, which is generated from the previous component. $\mathbf{S}$ is the set of the selected views and is the final result we want. C is the canvas used for greedy selection at each stage, which records the result of the previous stage. $\mathbf{T}$ is set for the union of C and all pixel sets in $\mathbf{M}$, and r is the coverage pixel number for each union result in $\mathbf{T}$. Finally, k is the size limitation of the selected view set $\mathbf{S}$. With the symbols, we can formulate the objective function as below.

$$\text{maximize} \left| \bigcup_{s \in \mathbf{S}} \mathbf{M}_s \right|, s.t. |\mathbf{S}| \leq k \tag{4.2}$$

The function shows that our goal is to right view set $\mathbf{S}$, in which the union of all masks inside the set is maximum under the constraint of set size k.

Algorithm 2 Greedy View Selection

```
1:  // Initialize canvas
2:  C ← image filled with value-0 pixels in the same size of views
3:  // Select one view in each stage
4:  for i from 0 to k − 1 do
5:      for n from 0 to |M| − 1 do
6:          T_n ← M_n ∪ C
7:          r_n ← number of value-1 pixels in T_n
8:      // Find the view combination with the max coverage
9:      idx ← index of max element in r
10:     // Assign selected view in this stage
11:     S_i ← idx
12:     // Update canvas with the selected combination
13:     C ← T_idx
14: return S
```

The pseudo-code of the algorithm is presented as Algorithm 2. In the code, we first initialize the canvas as an uncovered mask, where all the pixels are set to 0 at line 2. Then, we start the selection for k stages and select the one view per stage. At each stage, we go through all pixel sets in $\mathbf{M}$ to get their union result with the canvas C and store them in the temporary set $\mathbf{T}$ at line 6. Then we calculate the coverage score r of each union result in $\mathbf{T}$ at line 7. After going through all the masks and get their union coverage score, we get idx (view index) from the mask with the highest score at line 9. The idx is the view index we select in this stage, we then update the canvas by assigning the union result T_{idx} to it at line 13. After all k stages are through, we get the currently optimal view set $\mathbf{S}$ at line 14.

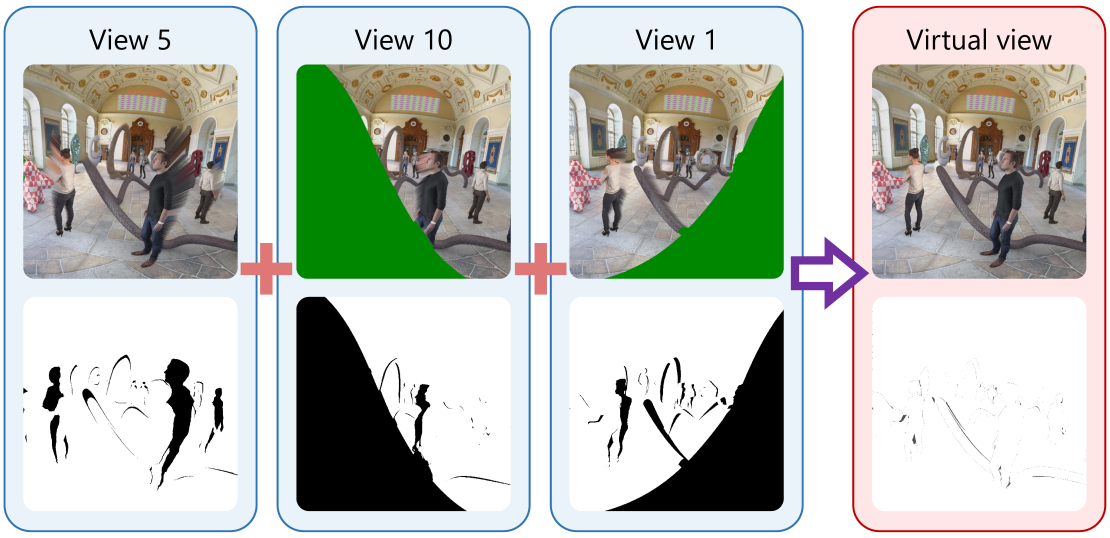


Figure 4.5: Example of our view selection strategy. In each stage, we choose the view combination that covers the most pixels in target view. The final selected view set [v5, v10, v1] leads to the coverage of 99.368%.

An example of the greedy view selection is shown as Fig. 4.5, in which the number of reference views is 12 and the size of the selection k is 3. From the figure, we see that at the first stage view 5 is selected for its coverage score is the highest among all reference views, which is up to 92.56%. The canvas will be updated with the mask of view 5 for the second stage selection. In the second stage, we select view 10 since it contains the most uncovered pixels in the canvas, which is now the mask of view 5. Then the canvas will be updated with the union result of view 5 and 10 for the third selection. At the final stage, view 1 will be selected for it again covers the most pixels in the canvas. The final synthesis result of view set [1, 5, 10] and its corresponding coverage mask are shown in the figure, the coverage percentage of the selected set is up to 99.37%.

The selection mentioned above algorithm is not the optimal solution, whereas it's the best solution in polynomial time. Since the selection process needs to be done in a short time, a more efficient algorithm with a certain quality (approximation ratio ≈ 0.632) is preferred. Moreover, to maintain the coherence between frames, we preserve the view set selected in the last frame and examine the coverage score it has for this frame (use the masks in this frame). If the score from the previously selected set is better than the current result, the set will be used in this frame again; also, if the previous result is worse than the current one but the difference is within a threshold (like 0.001%), the set will be used as well. This way, we can make the synthesized view coherent at a certain level.

4.3 Other Solutions

Besides the MCP-based greedy algorithm, we described in the previous section, we also implement other solutions for the view selection. One of them is the view selection algorithm based on the pixel importance, which is the number of views that cover a pixel, the smaller the number, the more critical. The other one is the offline view selection, instead of proposing a new selection algorithm, we design a mechanism to pre-select and store the view sets of certain viewpoints in advance so they can be utilized as the solution for similar target viewpoint and save us the time of on-the-fly selection. Both of the methods are elaborated in the following sections.

4.3.1 Pixel Importance-Based View Selection

Pixel importance-based view selection is a novel approach to tackle the problem, instead of using the coverage of the view, we utilize the *importance* property of each pixel to decide which views to pick. In our design, the importance of a pixel in the target view is defined by the number of the reference views that can cover the very pixel, the smaller the view number is, the more critical a pixel is. For instance, the pixel that is covered by three different views is less important than the one that is covered by only one view. This design ensures the pixels that are the least touched can be covered first. Also, several conditions are set for the conflicts during the selections.

- **Selection order.** We select the view starting from the ones that cover the most important pixels on the target view. Once the view is selected, the pixels it covers are discarded from the selection space so we won't get the same pixels again.
- **Multiple pixels with the same importance.** The conflict happens when multiple pixels have the same amount of covering views, and we need to find the right pixel among them. In our design, we select the one pixel that is the farthest from the previously selected pixel. Such selection makes sure that as many different parts of the target view can be covered as possible, which is crucial since empirically the least covered pixels locate in some remote areas of the image.
- **Multiple views cover a pixel.** The conflict happens when a pixel is covered by multiple reference views, which is typical for there can quite a few overlapping areas between the views. To deal with it, we select the one view with the maximum coverage area on the target view. The design ensures that as many covered pixels as possible for that is the primary goal of the view selection.

An example of the pixel importance-based selection method is shown as Fig. 4.6, which presents three stages of selection. At the first stage, the most important pixel,

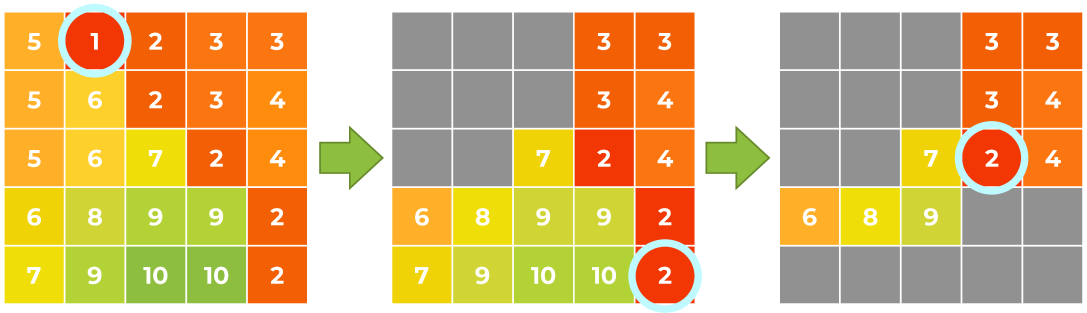


Figure 4.6: Procedure of pixel-importance-based view selection algorithm. In each stage, we choose view of the most important pixel under several conditions.

which only gets covered by one view, is picked, and that one view covers it is selected as the first view S_0. Then the pixels that S_0 covers are all discarded. At the second stage, three pixels share the same importance, and we pick the one that's farthest from the pixel picked at the first stage. Since two views cover the pixel, we select the one that has the most coverage area as the second view S_1. Finally, at the third stage, we again pick the pixel that has the least covering views, and from the two views, we select the one with the most coverage area as the last view S_2. After the stages are over, we get the result view set S.

This algorithm is dense in computation, the decisions it needs to make is far more complicated than the greedy algorithm. Therefore, this algorithm is not suitable for real system usage.

4.3.2 Offline View Selection

Our proposed selection method often requires lots of computation, e.g., the union operation between the canvas and the multiple masks, let alone the mask generation process, the warping is a costly operation. To save these computation power as well as the latency, we propose a method to pre-select and store the view set for certain viewpoints beforehand, and during the playout time, the sets can be used directly as the selection result. This way, we can skip the whole hole-aware view selection process and send the stored view set to the view synthesizer.

One big challenge here is that the 3DoF+ VR requires a smooth view transition, how many pre-select viewpoints we need to achieve that? In our design, we restrict the 6DoF transition to 4DoF by fixing the Z-axis position and roll-axis rotation, and in each remaining dimension, we set the start, end, and step variables to define the range of viewpoints. As shown in Fig. 4.7, the start, end, and step variables in X-axis movement are defined as

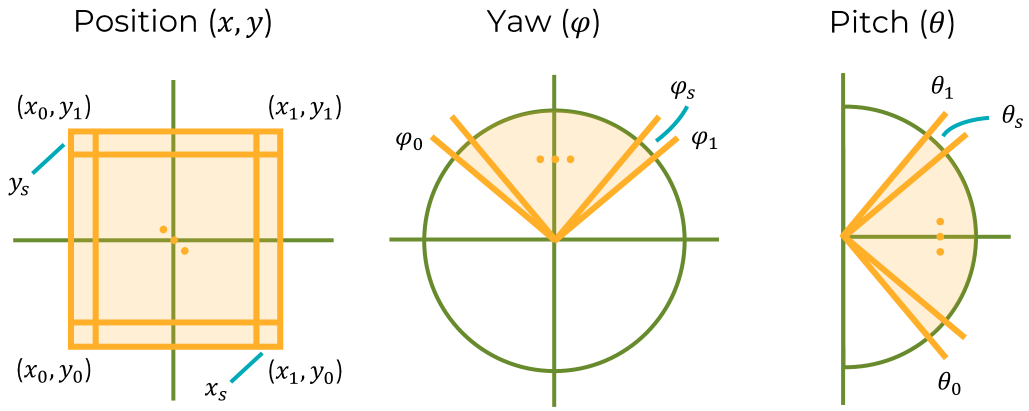


Figure 4.7: The stored viewpoints of offline view selection. The Z-axis position, as well as the roll-axis rotation, are fixed. For the rest 4 degree of freedoms, we define the start, end, step values to determine the pre-selected viewpoints.

(x_0, x_1, x_s), and the view number in X-axis is derived as $x_n = (x_1 - x_0)/x_s + 1$. The same principle can be applied to other dimensions and we eventually define the total viewpoint number as $x_n \times y_n \times \theta_n \times \phi_n$.

4.4 Evaluations

In the evaluation part, we held experiments to test the performance of the system considering both synthesis quality as well as process latency. Also, we compare the performance between offline and online view selection with a real user trace, which contains 1,200 frames of viewpoints. In our experiments, we compare our system's performance to others like Dziembowski et al. [37] as well as other algorithms. In the quality assessment, we utilize PSNR, SSIM and coverage percentage of the synthesis results as the evaluation metrics; and in the latency measurement, we compare the latencies between different down-sampling ratios of greedy selection. The details, as well as analysis of the experiments, are described in the following paragraph.

4.4.1 Implementation

Compared to the auto-refocus system, most of the components in this system are linked externally. We implemented the hole-aware view selector, including both of its sub-components, in python, and accelerate it with cython module. These modules are compiled to DLLs for the system usage. As for the view synthesizer component, we use

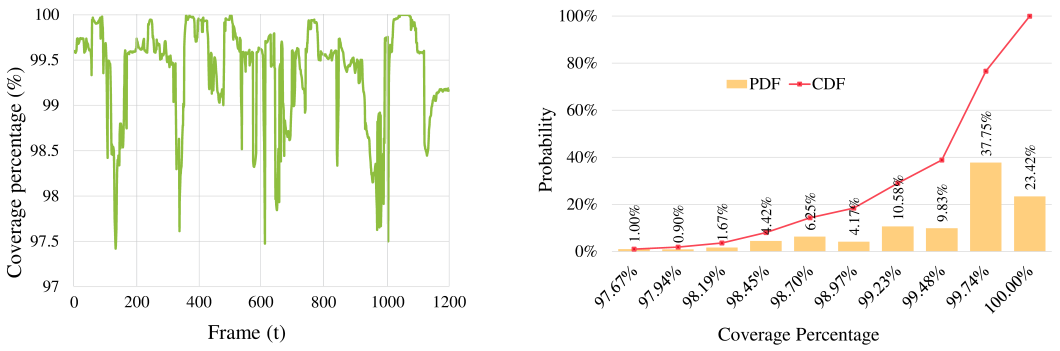
the software from MPEG-I group developed in 2018 called Reference View Synthesizer (RVS) [44]. The RVS is a reference software whose performance is not optimized; we use it for the conservative purpose. The panorama player is implemented on unity engine and is written in C# to connect the FOVE HMD, which can collect the user viewpoint information.

Table 4.3: Test material we use in our system.

Name	**Cameras**	**Frames**	**FoV**	**Depth**
Technicolor Museum	24	300	180° × 180°	Y
Technicolor Hijack	10	300	180° × 180°	Y
Classroom Video	15	120	360° × 180°	Y

The test materials we use in the system are shown in Table 4.3, which are all collected from the MPEG-I group. All of these materials are multi-viewpoint with depth (MVD) sequences that contain light information from different positions and orientations. The table shows the camera number, frame number, and field of view (horizontal × vertical angular range) of each sequence. In our experiments, we extract the first frame of the sequence *Technicolor Museum* [36] as our target scene, that is, we test our user traces and offline view set selection based on this scene. The details of the experiments will be further introduced in the following sections. The machine used in our experiments is a PC with an Intel i7-2600 CPU and 16 GB of memory.

4.4.2 Performance Analysis



(a) (b)

Figure 4.8: View selection results of the user trace data using the proposed algorithm: (a) Distribution of the synthesis result coverage, (b) the PDF (bar) and CDF (curve) of the coverage result.

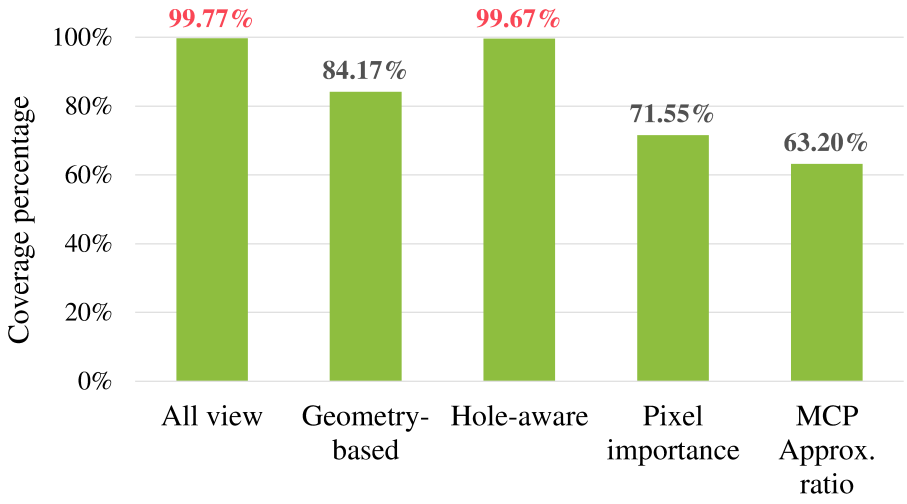


Figure 4.9: Results of average coverage percentage using different view selection algorithms.

In this section, we evaluate the performance of our view selection with two parts of experiments: quality assessment and latency measurement. The former compares the synthesis quality of the selected views between different algorithms; and the later measure the view selection latency of different conditions. In the performance test, we use all-views result as our ground truth for it is the *best possible synthesis result* using all the reference views. However, it's no guarantee that the ideal view would be the same as the real scene, there still can be some problems depends on the viewpoint's position and orientation. The baseline of the evaluation is the work from Dziembowski et al. [37], in which they select views based on the geometry relationship. That is, to select the most similar views as the results considering the only relation between the reference view parameters and the target view and not the real space information. All the synthesis results are from the 1200 viewpoints of the user trace, in which the PDF and the CDF of the synthesis result coverage are shown in Fig. 4.8. From the figures, we see that more than 80% of the distribution is above 99% of view coverage.

Fig 4.9 shows the average coverage percentages of the synthesis results from different algorithms. As we can see, our hole-aware view selection has the result of 99.67% average coverage percentage, which is 15.5% higher than the baseline and only 0.1% lower than the all-views result. The number proves that our proposed view selection method is advantageous. On the other hand, pixel importance-based algorithm is not that useful, the results show that its synthesis quality is merely higher than the MCP approximation ratio. In Fig. 4.10 we observe the similar results, the hole-aware view selection algorithm holds the best performance while the pixel importance-based algorithm performs poorly.

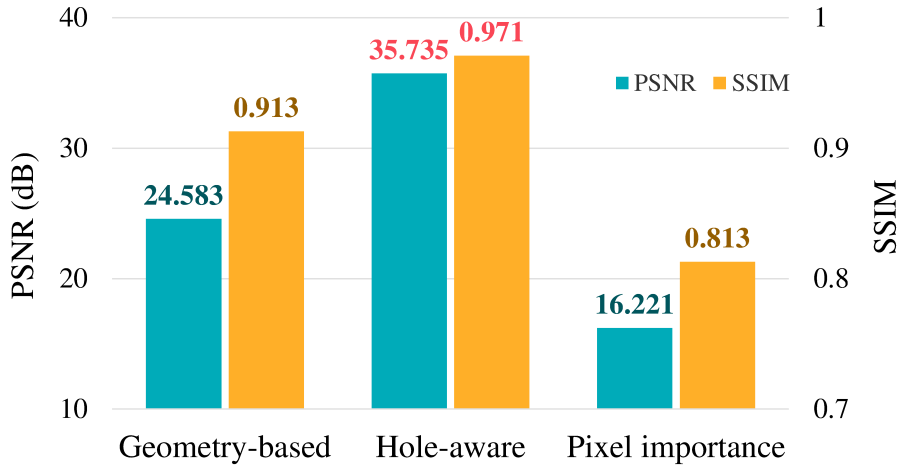


Figure 4.10: Results of PSNR (left Y-axis) and SSIM (right Y-axis) comparing to ground truth using different algorithms.

Note that the all-views result is not in the figure because it is the ground truth in this comparison.

Fig 4.11 presents the performance results of our hole-aware view selection algorithm under different down-sampling ratios. By applying down-sampling, we down-sample the reference view image resolution with the assigning value d. Specifically, let the view image resolution be 2048×2048, and the down-sampling ratio d be 2, then the image will be down-sampled to 1024×1024 before feeding to the mask generator. This way, we can reduce the number of processed pixels and save bandwidth and computation. From the figure, we can see that the view selection latency with $d = 1$ is exceptionally high, up to 2.5 seconds; and with $d = 16$, the processing time is reduced to 30 ms while the quality (coverage percentage) can still reach above 99% on average.

Fig. 4.12 shows the exact processing time inside the hole-aware view selector component, which contains two components as we previously introduced. From the results, it's obvious that the main execution time of the selection process is in the mask generation part, which in some case ($d = 16$) the difference is up to 60 times. In this figure, the **optimal** represents the optimal view set we can find with the limited set size, which in our case we use brute force search to find the answer (MCP is NP-hard), and it leads to 17.8 times of processing time compares to the hole-aware algorithm with the same down-sampling ratio. Finally, since the geometry-based method (baseline) only considers the camera parameter, it's way more efficient than the hole-aware algorithm, which generates and utilizes the space information. To sum up, we recommend using $d = 8$ for the process for it can be rather efficient while maintaining a certain level of quality.

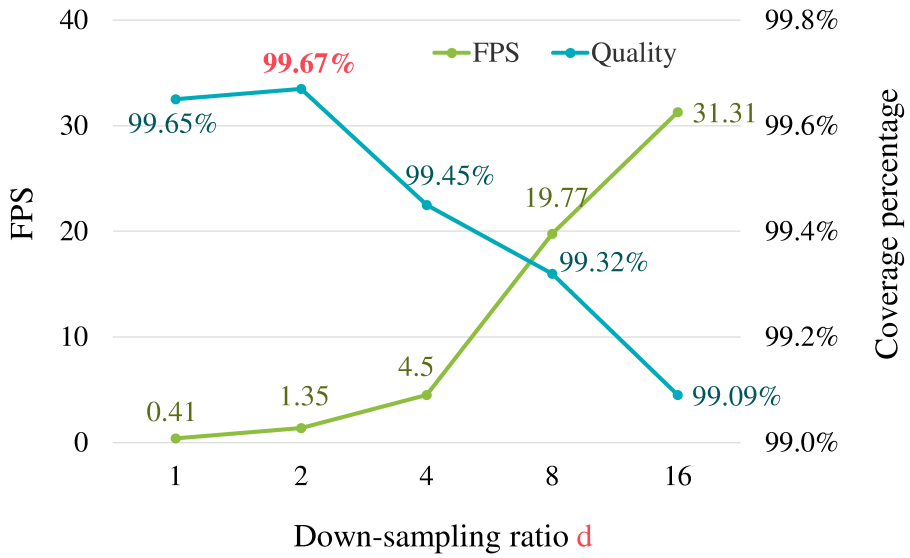


Figure 4.11: Performance of the hole-aware view selection algorithm under different down-sampling ratios.

4.4.3 Offline vs. Online View Selection

In this part of the evaluation, we examine the performance difference between online and offline view selection. As we previously mentioned in Sec. 4.3.2, we pre-select and store the view sets for a quick view set appliance. Since the pre-select viewpoints are discontinuous and they cannot cover all the user viewpoints, there must exist some deviance between the results of the stored view set and the on-the-fly selected view set. The experiments below show the differences, and we will analyze and discuss observations. In our experiments, the *online* scenario means that the view sets are generated in run-time with different algorithms (hole-aware or optimal aka brute force); the *offline* scenario means that we select the view sets for some viewpoints with different algorithms in advance and utilize those sets for similar viewpoint synthesis.

Table 4.4: Variables to define the range of the pre-selected viewpoints.

DoF	Start	End	Step	Size
x	-0.3	0.3	0.1	7
y	-0.3	0.3	0.1	7
θ	-50°	50°	10°	11
ϕ	-50°	50°	10°	11

To decide which viewpoints to store, we need to set the start, end, step variables for 4

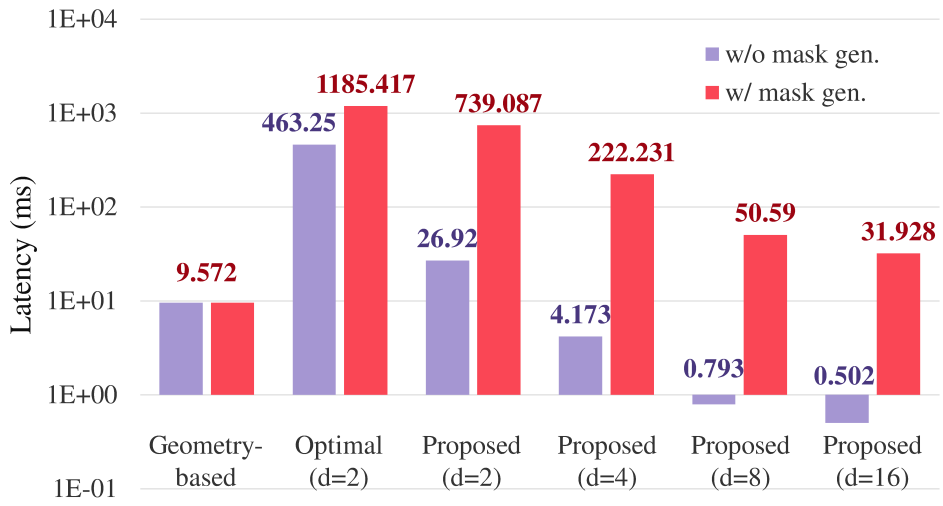
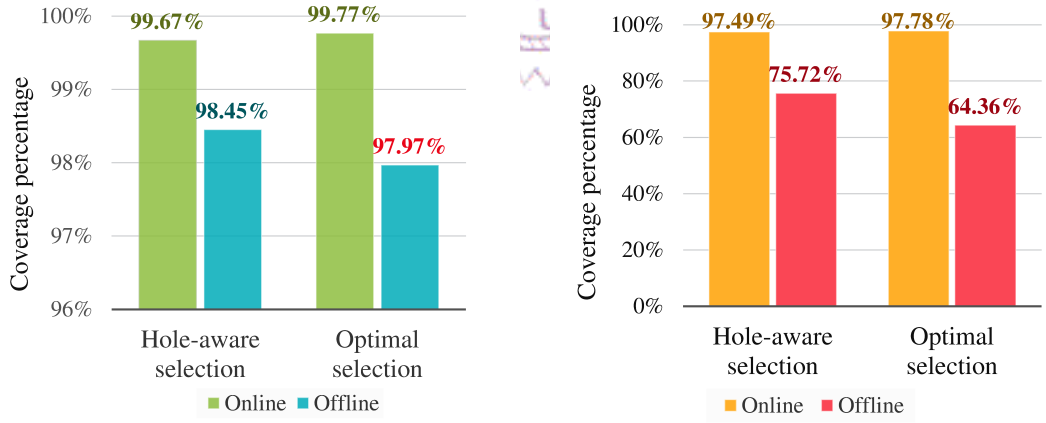


Figure 4.12: Results of processing time of hole-aware view selector component.

degrees of freedom as mentioned in Sec. 4.3.2. The variables are set in Table 4.4, where we have 49 positions and 121 orientations, hence the total number of viewpoints would be $49 \times 121 = 5929$. These viewpoints represent the offline selection, and its performance will be later evaluated with the user trace.



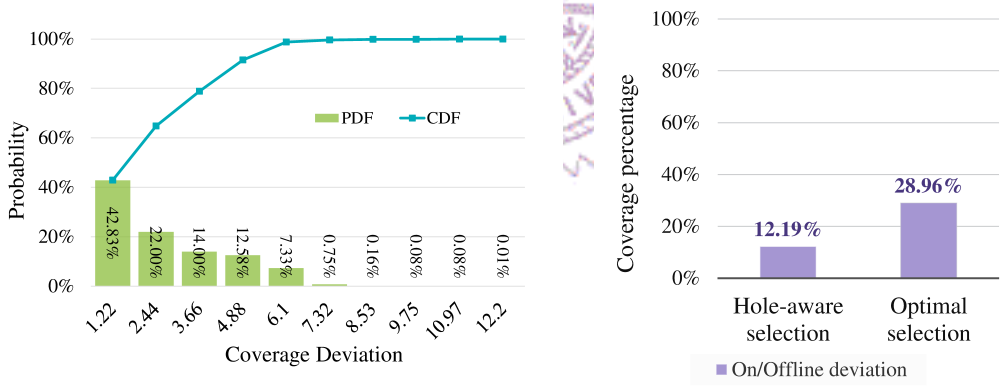
(a) (b)

Figure 4.13: Results of offline and online view selection applying different algorithms: (a) average coverage percentage among the views; (b) minimum coverage percentage among the views.

The results of the coverage percentage of the synthesis results using both algorithms are shown in Fig. 4.13. Fig. 4.13(a) shows the average coverage percentage of the synthesis results of both algorithms. From the figure, we can see that for both algorithms, their

average coverages of online selected sets are higher than that of offline selected sets. The relationship exists due to the deviation between the offline and online selection, we can solve the problem by increasing the alignment density of pre-selected viewpoint.

Also, an intriguing observation can be made is that while the online selected sets using optimal solution leads to better qualities than that using the proposed (hole-aware) solution, the offline selected sets using optimal solutions don't lead to the same outcome. The phenomenon is because of the *viewpoint specific selection*, in which the selected view set is only suitable for a particular viewpoint and cannot handle the deviations well. That is, the optimal view sets are way too specific for the viewpoint, it does not suit to other viewpoints even if the deviation is small. On the other hand, our proposed solution doesn't necessarily get the optimal view set for the viewpoint, its applicability is broader and can be suitably used for similar viewpoints. The similar phenomenon can be observed in Fig. 4.13(b), in which the minimum coverages of both scenarios are shown. Again, we see that the minimum coverage of offline selected set using optimal solution is lower than that using the proposed solution while the online scenario shows the opposite results, showing that the deviation between online and offline scenarios using the optimal solution is more severe than using the proposed solution.



(a) (b)

Figure 4.14: Results of the deviation between offline and online view selection using different algorithms: (a) deviation distribution of the views; (b) maximum deviation among the views.

Fig. 4.14 shows the overview of the deviations ($online - offline$) between offline and online selection using different methods, including their distributions and the max value among them. From Fig. 4.14(a), we can see that the online selection is always better than the offline selection since there are no negative values in the distribution. It is intuitive since the viewpoint deviations always cause more uncovered areas (holes). The coverage deviation, however, is not severe in most cases (90% of the views have the deviations

less than 4.88%). Fig. 4.14(b) shows the max deviation throughout the whole user trace of different algorithms. From the results, the viewpoint specific selection problem can be further validated: the result from the optimal solution is higher than that from the proposed solution in about 16.8%. Also, we can decrease the deviation between online and offline selection and avoid viewpoint specific selection problem by increasing the density of the pre-selected viewpoint alignment, which depends on the memory capacity of the server. The specific viewpoint selection indicates that our method has more suitability to the similar viewpoints than the optimal selection.

Chapter 5

Conclusion and Future Work

In this thesis, we study the development of the 360° video and the HMD VR system. We found that there are still some limitations in the current 360° HMD system, such as the fixed focal length and the fixed viewpoint, these limitations keep users from further exploring the technology. To conquer the limitations, we study light field (LF) technology as well as its potential application in HMD VR system. Based on its properties, we proposed and designed two HMD systems to deal with the limitations mentioned above. In the first system (auto-refocus VR system), we aim to apply dynamic focal length adaptation based on the user eye gaze. We utilize LF panoramas for the real-time refocus effect, and connect FOVE HMD for the eye gaze information collection. Also, two optimization methods are proposed to reduce the latency of the refocusing process; the evaluations show that the performance can reach up to 200 times faster than the unoptimized process on average. in the second system (3DoF+ VR system), we provide the VR experience where users' viewpoints in HMD can move along with their (head, body) positions. The multi-view video sequences from MPEG are used as test materials and use FOVE for position detection. To reduce the bandwidth as well as the computation, we propose a novel view selection algorithm for the view synthesis process that can leverage the real space information. The results from the experiments show that the synthesis result from our proposed algorithm is only 0.1% lower than the optimal solution while the processing time is 18 times faster.

Our works, still, can be extended in multiple directions:

- **Integration of the two proposed systems.** Since the systems we proposed are both based on the LF technology, we want to integrate them as one for the greater user experiences. Being able to experience VR content from different viewpoints with focal length adaptation can increase the immersion.

- **System acceleration with GPUs.** From the evaluations, we see that the latency

is a significant flaw in both of our systems, we should leverage GPU toolkit and parallel computing to accelerate the rendering and improve the performance. These enhancements will further consolidate the systems and increase the user experience.

- **6DoF VR Expansion.** 6DoF VR is the ultimate form of the VR experience, and it allows users to stroll in the virtual scene immersively. To achieve this, we need to scale up the light field data by capturing more light information from different positions, which leads to more complicated system design. Such improvement would significantly boost the user experience without a doubt.

By differentiating the pros and cons of both our systems, we open up new opportunities for researchers and engineers to further develop LF applications in AR/VR system in terms of user experience.

Bibliography

[1] Lytro Light Field camera. http://lightfield-forum.com/lytro/lytro-lightfield-camera/, 2013. Accessed August 2019.

[2] RayTrix R11 3D Light Field Camera. http://lightfield-forum.com/raytrix/raytrix-r11-3d-lightfield-camera/, 2014. Accessed August 2019.

[3] Lytro Illum - Professional Light Field Camera. http://lightfield-forum.com/lytro/lytro-illum-professional-light-field-camera/, 2015. Accessed August 2019.

[4] Facebook Spaces. https://www.facebook.com/spaces, 2017. Accessed April 2018.

[5] Lytro Support. https://support.lytro.com/hc/en-us, 2017.

[6] Meet the Lytro Immerge 2.0, a 95-lens VR camera that could soon shoot in 10K. https://www.digitaltrends.com/photography/meet-the-lytro-immerge-2/, 2017. Accessed August 2019.

[7] csmatio .NET Library for Matlab MAT-files. https://sourceforge.net/projects/csmatio/files/, 2018.

[8] Facebook Oculus Rift. https://www.oculus.com, 2018. Accessed May 2018.

[9] Filming the Future with RED and Facebook 360. https://facebook360.fb.com/2018/09/26/film-the-future-with-red-and-facebook-360/, 2018. Accessed August 2019.

[10] FOVE: Eye-Tracking Virtual Reality Headset. https://www.getfove.com/, 2018.

[11] GOOGLE AR AND VR - Experimenting with Light Fields. https://www.blog.google/products/google-ar-vr/experimenting-light-fields/, 2018. Accessed August 2019.

[12] Google Cardboard. https://vr.google.com/cardboard/, 2018. Accessed May 2018.

[13] HTC Vive. https://www.htcvive.com, 2018. Accessed May 2018.

[14] HTC Vive Focus. https://www.vive.com/cn/product/vive-focus-en/, 2018. Accessed May 2018.

[15] Luna 360 VR. http://luna.camera/, 2018. Accessed May 2018.

[16] OpenCV (Open Source Computer Vision Library). https://opencv.org/, 2018.

[17] OpenCVSharp for Unity - Unity Asset Store. https://assetstore.unity.com/packages/tools/integration/opencv-for-unity-100374, 2018.

[18] RayTrix: 3D Light Field Camera Technology. https://raytrix.de/, 2018.

[19] Richo Theta S. https://theta360.com, 2018. Accessed May 2018.

[20] Samsung Gear 360. http://www.samsung.com/global/galaxy/gear-360/, 2018. Accessed May 2018.

[21] Sony Playstation VR. https://www.playstation.com/en-au/explore/playstation-vr/, 2018. Accessed May 2018.

[22] Unity. https://unity.com/, 2018.

[23] A. A. Ageev and M. I. Sviridenko. Approximation algorithms for maximum coverage and max cut with given sizes of parts. In *Integer Programming and Combinatorial Optimization*, pages 17–30. Springer Berlin Heidelberg, 1999.

[24] S. Avidan and A. Shashua. Novel view synthesis in tensor space. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1034–1040, June 1997.

[25] J. Berent and P. L. Dragotti. Segmentation of epipolar-plane image volumes with occlusion and disocclusion competition. In *2006 IEEE Workshop on Multimedia Signal Processing*, pages 182–185, Oct 2006.

[26] C. Birklbauer and O. Bimber. Panorama light-field imaging. *EUROGRAPHIC*, 33(2):43–52, May 2014.

[27] R. C. Bolles, H. H. Baker, and D. H. Marimont. Epipolar-plane image analysis: An approach to determining structure from motion. *International Journal of Computer Vision*, 1(1):7–55, Mar. 1987.

[28] V. Boominathan, K. Mitra, and A. Veeraraghavan. Improving resolution and depth-of-field of light field cameras using a hybrid imaging system. In *2014 IEEE International Conference on Computational Photography (ICCP)*, pages 1–10, May 2014.

[29] K. Carnegie and T. Rhee. Reducing visual discomfort with hmds using dynamic depth of field. *IEEE Computer Graphics and Applications*, 35(5):34–41, September 2015.

[30] J. Chakareski. Adaptive multiview video streaming: challenges and opportunities. *IEEE Communications Magazine*, 51(5):94–100, May 2013.

[31] S. E. Chen and L. Williams. View interpolation for image synthesis. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques (SIGGRAPH '93)*, pages 279–288. ACM Press, 1993.

[32] B. Clipp, J. Kim, J. Frahm, M. Pollefeys, and R. Hartley. Robust 6dof motion estimation for non-overlapping, multi-camera systems. In *2008 IEEE Workshop on Applications of Computer Vision*, pages 1–8. IEEE, Jan. 2008.

[33] A. Collet, M. Chuang, P. Sweeney, D. Gillett, D. Evseev, D. Calabrese, H. Hoppe, A. Kirk, and S. Sullivan. High-quality streamable free-viewpoint video. *ACM Trans. Graph.*, 34(4):69:1–69:13, July 2015.

[34] D. Dansereau. Matlab Light Field Toolbox v0.4. `https://www.mathworks.com/matlabcentral/fileexchange/49683-light-field-toolbox-v0-4`, 2015.

[35] D. G. Dansereau, O. Pizarro, and S. B. Williams. Linear volumetric focus for light field cameras. *ACM Trans. Graph.*, 34(2):15:1–15:20, Mar. 2015.

[36] R. Doré, G. Briand, and T. Tapie. Technicolor 3DoFPlus Test Materials. International Organization for Standardization Meeting Document ISO/IEC JTC1/SC29/WG11 MPEG/M42349, 2018. Meeting held at San Diego USA.

[37] A. Dziembowski, J. Samelak, and M. Domański. View selection for virtual view synthesis in free navigation systems. In *2018 International Conference on Signals and Electronic Systems (ICSES)*, pages 83–87, Sept. 2018.

[38] S. Fachada, D. Bonatto, A. Schenkel, and G. Lafruit. Depth image based view synthesis with multiple reference views for virtual reality. In *2018 - 3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON)*, pages 1–4, June 2018.

[39] T. Georgiev and A. Lumsdaine. Superresolution with plenoptic camera 2.0. 2009.

[40] X. Guo, Z. Yu, S. B. Kang, H. Lin, and J. Yu. Enhancing light fields through ray-space stitching. *IEEE Transactions on Visualization and Computer Graphics*, 22(7):1852–1861, July 2016.

[41] S. Khuller, A. Moss, and J. S. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.

[42] C. Kim, H. Zimmer, Y. Pritch, A. Sorkine-Hornung, and M. Gross. Scene reconstruction from high spatio-angular resolution light fields. *ACM Trans. Graph.*, 32:73:1–73:12, 2013.

[43] B. Krolla, M. Diebold, B. Goldluecke, and D. Stricker. Spherical light fields. In *Proceedings of the British Machine Vision Conference 2014 : BMVC, 2014, Nottingham*, Durham, 2014. BMVA Press.

[44] B. Kroon. Reference View Synthesizer (RVS) manual. International Organization for Standardization Meeting Document ISO/IEC JTC1/SC29/WG11 MPEG/N18068, 2018. Meeting held at Macau SAR CN.

[45] Y. Lai and C. Hsu. Refocusing supports of panorama light-field images in head-mounted virtual reality. In *Proceedings of the 3rd International Workshop on Multimedia Alternate Realities*, AltMM'18, pages 15–20. ACM, 2018.

[46] M. Levoy and P. Hanrahan. Light field rendering. In *Proc. of ACM International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'96)*, pages 31–42, New Orleans, USA, August 1996.

[47] M. Levoy, R. Ng, A. Adams, M. Footer, and M. Horowitz. Light field microscopy. *ACM Trans. Graph.*, 25(3):924–934, July 2006.

[48] K. Müller, A. Smolic, K. Dix, P. Merkle, P. Kauff, and T. Wiegand. View synthesis for advanced 3d video systems. *EURASIP Journal on Image and Video Processing*, 2008(1), Feb. 2009.

[49] J. Moss, J. Scisco, and E. Muth. Simulator sickness during head mounted display (hmd) of real world video captured scenes. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 52(19):1631–1634, September 2008.

[50] R. Ng, M. Levoy, M. Brédif, G. Duval, M. Horowitz, and P. Hanrahan. Light field photography with a hand-held plenoptic camera. Stanford Tech Report CTSR 2005-02, 2005.

[51] S. Ohl. Tele-immersion concepts. *IEEE Transactions on Visualization and Computer Graphics*, 24(10):2827–2842, Oct. 2018.

[52] N. Padmanaban, R. Konrad, E. A. Cooper, and G. Wetzstein. Optimizing vr for all users through adaptive focus displays. In *Proc. of ACM International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'17) Talks*, pages 77:1–77:2, Los Angeles, USA, July 2017.

[53] B. Ray, J. Jung, and M. Larabi. On the possibility to achieve 6-dof for 360 video using divergent multi-view content. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 211–215, Sept. 2018.

[54] Z. M. 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. `https://www.zionmarketresearch.com/report/virtual-reality-market`, 2017. Accessed August 2019.

[55] M. Shirer and S. Murray. IDC Sees the Dawn of the DX Economy and the Rise of the Digital-Native Enterprise. `https://www.businesswire.com/news/home/20161101005193/en/IDC-Sees-Dawn-DX-Economy-Rise-Digital-Native`, 2016. Accessed April 2018.

[56] A. Smolic, K. Mueller, P. Merkle, C. Fehn, P. Kauff, P. Eisert, and T. Wiegand. 3d video and free viewpoint video - technologies, applications and mpeg standards. In *2006 IEEE International Conference on Multimedia and Expo*, pages 2161–2164, July 2006.

[57] M. W. Tao, S. Hadap, J. Malik, and R. Ramamoorthi. Depth from combining defocus and correspondence using light-field cameras. In *2013 IEEE International Conference on Computer Vision*, pages 673–680, Dec 2013.

[58] D. Tian, P. Lai, P. Lopez, and C. Gomila. View synthesis techniques for 3d video. In *Applications of Digital Image Processing XXXII*, volume 7443. International Society for Optics and Photonics, Sept. 2009.

[59] J. Unger, A. Wenger, T. Hawkins, A. Gardner, and P. Debevec. Capturing and rendering with incident light fields. In *Proceedings of the 14th Eurographics Workshop on Rendering*, pages 141–149, 2003.

[60] V. Vazirani. *Approximation algorithms*. Springer, Berlin New York, 2001.

[61] K. Venkataraman, D. Lelescu, J. Duparré, A. McMahon, G. Molina, P. Chatterjee, R. Mullis, and S. Nayar. Picam: An ultra-thin high performance monolithic camera array. *ACM Trans. Graph.*, 32(6):166:1–166:13, Nov. 2013.

[62] T. Wang, J. Zhu, N. K. Kalantari, A. A. Efros, and R. Ramamoorthi. Light field video capture using a learning-based hybrid imaging system. *ACM Trans. Graph.*, 36(4):133:1–133:13, July 2017.

[63] X. Wang, L. Chen, S. Zhao, and S. Lei. From OMAF for 3DoF VR to MPEG-I Media Format for 3DoF+, Windowed 6DoF and 6DoF VR. International Organization for Standardization Meeting Document ISO/IEC JTC1/SC29/WG11 MPEG/M41197, 2017. Meeting held at Torino, Italy.

[64] S. Wanner and B. Goldluecke. Variational light field analysis for disparity estimation and super-resolution. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(3):606–619, March 2014.

[65] B. Wilburn, N. Joshi, V. Vaish, E. Talvala, E. Antunez, A. Barth, A. Adams, M. Horowitz, and M. Levoy. High performance imaging using large camera arrays. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 765–776. ACM, 2005.

[66] G. Wu, B. Masia, A. Jarabo, Y. Zhang, L. Wang, Q. Dai, T. Chai, and Y. Liu. Light field image processing: An overview. *IEEE Journal of Selected Topics in Signal Processing*, 11(7):926–954, Oct 2017.

[67] J. C. Yang, M. Everett, C. Buehler, and L. McMillan. A real-time distributed light field camera. In *Proceedings of the 13th Eurographics Workshop on Rendering*, EGRW '02, pages 77–86, 2002.

[68] T. Yang, Y. Zhang, J. Yu, J. Li, W. Ma, X. Tong, R. Yu, and L. Ran. All-in-focus synthetic aperture imaging. In *Computer Vision – ECCV 2014*, pages 1–15, 2014.

[69] Z. Yu, X. Guo, H. Ling, A. Lumsdaine, and J. Yu. Line assisted light field triangulation and stereo matching. In *2013 IEEE International Conference on Computer Vision*, pages 2792–2799, Dec 2013.

[70] K. Yücer, A. Sorkine-Hornung, O. Wang, and O. Sorkine-Hornung. Efficient 3d object segmentation from densely sampled light fields with applications to 3d reconstruction. *ACM Trans. Graph.*, 35(3):22:1–22:15, Mar. 2016.

[71] M. Zink, R. Sitaraman, and K. Nahrstedt. Scalable 360° video stream delivery: Challenges, solutions, and opportunities. *Proceedings of the IEEE*, pages 1–12, 2019.