A New Algorithm for Ray Tracing Synthetic Light Fields on the GPU

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science in ***Computer Science and Engineering*** *by Research*

by

Udyan Khurana
201202101
udyan.khurana@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
April 2020

Copyright © Udyan Khurana, 2020
All Rights Reserved

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "A New Algorithm for Ray Tracing Synthetic Light Fields on the GPU" by Udyan Khurana, has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Adviser: Prof. P. J. Narayanan

To my Parents

Acknowledgments

I would like to express my sincere gratitude to my advisor Prof. P.J. Narayanan for his continuous support in my research, and for all the patience and motivational guidance, especially when I felt like nothing was going the way I wanted it to. I would like to thank Parikshit Sakurikar for mentoring me and guiding me throughout, and for the support, effort and time he has given in helping me shape my research work. I want thank PJN Sir and Parikshit not just for my work, but also in fostering the right attitude towards research in general.

I would also like to thank Saurabh Saini, Rajvi Shah and my fellow labmates for our numerous discussions, both on and off the subject, and to everyone, who lent their hand in this venture, directly or indirectly. Last but not the least, I would like to thank my family, for their constant love and support, and to my friends for all their help in making this journey enjoyable through all my ups and downs.

Abstract

Ray Tracing is one of the two major approaches to 3D rendering. Images produced through ray tracing show a higher degree of visual realism, and are more accurate compared to other 3D rendering methods like rasterisation. It is primarily used to generate photo-realistic imagery, as it is based on tracing the path of light going into the scene space. Ray tracing was initially an offline algorithm, and was seen as more suitable for applications which required high quality rendering, where the render time constraints were not as strict. Still images, animated films and television series thus used Ray tracing for rendering visual effects like reflections, shadows, refractions etc. accurately.

Computer Generated Imagery (CGI) was first used in the science fiction movie Westworld in 1973. In 1995, 'Toy Story' became the first film to be fully developed using computer animation. RenderMan, a Ray tracing engine developed by Pixar Studios, made this possible. At that time, rendering enough frames to make a 80-minute film took more than 800,000 machine hours. With massive increases in computational power, and the invention of Graphics Processing Units (GPUs), things have considerably improved, leading to renewed interest in ray tracing research. GPUs gave a huge boost to Ray tracing because of it's inherently parallel nature. Though GPUs accelerate almost all parts of the Ray tracing pipeline, it was still considered to be impractical for interactive, real-time rendering applications, like games. With the annoucement of specalised hardware units for Ray tracing in the Turing GPU microarchitecture by NVIDIA, real-time ray tracing is believed to be possible now.

Camera technology has advanced a lot over the years. With standard optical cameras, it was important to focus accurately on the subject, as slight errors would cause a substantial loss of quality in the desired portions of the image. Digital cameras today have the capability to adjust focus and aperture on their own to accurately focus on a subject. While simulating a camera-esque setup using computer graphics, we don't have the flexibility of a refocusing after rendering is completed. The only way is to render the setup again with the correct setting. The light field representation offers a viable solution to this problem.

Light field is a 4D function that captures all the radiance information of a scene. Traditionally, the creation of light fields was done by image-based rendering mechanisms. These reconstruct the 4D space using pre-captured imagery from various views and employ refocusing algorithms to generate output images. Plenoptic cameras capture the information of all light coming towards the camera, and use that to generate images at any focus setting. Handheld plenoptic cameras, like the Lytro camera series,

also capture light fields using a microlens array between the sensor and the main lens, but physical constraints have limited their spatial and angular resolutions.

The first part of this thesis presents a GPU-based synthetic light field rendering framework that is robust and physically accurate. It can create light fields of very high resolutions, which are orders of magnitude higher than currently available in commercial plenoptic cameras. We show how light field rendering is possible and viable in a synthetic rendering setup, and high-quality images at any desired focus and aperture setting can be generated in reasonable times with a very low memory footprint. We explain the theory behind synthetic light field creation, different representations possible for a 4D light field, and show results based on our implementation, as compared with state-of-the-art ray tracing frameworks like POVRay.

In the second part of the thesis, we attempt to solve the problem of light field storage. The 4D light field is inherently rich in information, but is very bulky to store. We address this issue by separating the light field creation and output image generation passes of a synthetic light field framework. Our thesis presents a compact representation of the 4D light slabs using a video compression codec and demonstrate different quality-size combinations using this representation. We demonstrate the equivalence of the standard light field camera representation, called the sub-aperture representation, with light slab representation for synthetic light fields. We use this to exhibit the capability of our framework, not only to trace light fields of very high resolutions, but also to store them in memory. The required images can therefore be generated independently from the ray tracing pass with a very small cost.

Contents

Chapter Page

1 Introduction . . . 1
1.1 Motivation . . . 5
1.2 Summary of Contributions . . . 6
1.3 Thesis Organization . . . 6

2 Background and Related Work . . . 7
2.1 Ray Tracing . . . 7
2.2 Light Fields . . . 8
2.3 Obtaining Light Fields . . . 9
2.4 Compressed Light Field Storage . . . 11
2.5 Summary . . . 13

3 Light Fields . . . 14
3.1 The Light Slab Representation . . . 16
3.2 The Sub-aperture Representation . . . 16
3.2.1 Light Field Photographs . . . 19
3.3 Equivalence of the two Light Field representations . . . 20

4 Tracing Light Fields . . . 21
4.1 Tracing the Light Slab . . . 21
4.1.1 Results . . . 23
4.1.2 Comparison with other ray tracers . . . 27
4.2 The Sub-aperture Representation . . . 27
4.2.1 Results . . . 28
4.3 Analysis of the two LF representations . . . 28

5 Storing Light Fields . . . 30
5.1 Visualization of the light slab as video . . . 30
5.2 SLFT with Video Storage - Algorithm . . . 32
5.3 Visualizing the New SLFT Pipeline . . . 32

6 Digital Refocusing of Light Fields . . . 35
6.1 Slicing-Based Refocusing . . . 35
6.2 Shift-Sum Refocusing . . . 36
6.3 Discussion . . . 37

6.3.1 Refocusing after Tracing 38
6.3.2 Refocusing with Tracing 38

7 Results, Observations and Analysis 40
7.1 Implementation Details 40
7.2 Results 40
7.3 Observations and Analysis 45

8 Conclusions 50
8.1 Future Work 50

Bibliography 53

List of Figures

Figure Page

1.1 Understanding depth of field and aperture, and their effect on images. 2
1.2 Image illustrating depth of field in digital light field photography using the sub-aperture representation. Taken from **(author?)** [40]. 3
1.3 Basic ray tracing. From **(author?)** [45]. 4

2.1 The Ray Tracing Algorithm. Image taken from the Wikipedia page on Ray Tracing. . . 8
2.2 Slab representation of the 4D light field given by Levoy et. al. 9
2.3 Optical system of Adelson and Wang's plenoptic camera [4] 10
2.4 The Lytro Illum 2015, based on the camera model by **(author?)** [40]. 10
2.5 Earlier methods of capturing light fields. 12

3.1 The 5D plenoptic function, representing the flow of light through 3D space (from the Wikipedia page on Light Fields). (a) Radiance L along a ray can be thought of as the amount of light traveling along all possible straight lines through a tube whose size is determined by its solid angle and cross-sectional area. (b) Parameterizing a ray by position (x, y, z) and direction (θ, ϕ). (c) Radiance along a ray remains constant if there are no blockers. This allows for a 4D representation of light fields, due to redundancy in the plenoptic function. 14
3.2 Ray diagrams for the Light Slab Representation. 17
3.3 Ray diagram for the Sub-Aperture Representation. The red, green and blue rays depict rays from different view angles. A microlens point conjugates with a point on the world space, and a photosensor pixel corresponds to a region on the main lens. This is illustrated by three red rays originating from PS #1 going to different locations in the object space. Best viewed in color. 18
3.4 Sub-aperture images of a scene with a bike model.(a) and (b) show extreme sub-apertures when the bike is in focus, and (c) and (d) show extreme sub-apertures when the bike is out of focus. 20

4.1 (a) Light Slab, the 2-plane parameterization of the light field. (b) The 2D ray-space diagram. 22
4.2 Focal stack of bedroom scene generated using SLFT. The figures show shifting of focus from the back of the bedroom to the front in going from (a) to (e). Generated using light slab sampling algorithm. 24
4.3 Aperture stack of bedroom scene generated using SLFT. The figures show decreasing aperture size going from left to right, resulting in dimming of the image. 25

4.4 Aperture stack of bedroom scene generated using SLFT. The figures show decreasing aperture size going from left to right, but the dimming is compensated for, demonstrating increasing depth of field. 26
4.5 Our light field ray tracing compared with a starndard rendering framework. Figure (a) has 2 renderings by our method at different focus depths, and figure (b) has the corresponding images generated by POVRay. 27
4.6 Two full light field photographs of the same scene at different depths, as generated by our framework. 29

5.1 Ray diagram for Visualisation of video frames of light slab storage. Sampling the [u, v] plane gives small sections of the world space as frames of the video, and sampling the [s,t] planes gives sub-aperture images as video frames. High spatial resolutions lead to small perspective shifts, and efficient video encoding. Best viewed in color. 31
5.2 Diagrammatic representation of the SLFT pipeline with storage. 34

6.1 Slicing-Based Refocusing. 36
6.2 Processing the light field recorded using a plenoptic camera. Taken from **(author?)** [39]. 37
6.3 Images generated by SLFT demonstrating shift-sum refocusing on light slabs. (a)-(d) show sub-aperture images of the four corner viewing angles. (e) and (f) show outputs of digital refocusing applied on the sub-aperture images, with front and back in focus respectively. 39

7.1 Images of the test scenes generated by SLFT with shadow ray tracing disabled. Two different focal depths are shown for each scene. 41

List of Tables

Table Page

7.1 Comparison of PSNRs due to varying angular resolution in the light slab. The PSNR and SSIM values are computed by taking the highest angular resolution as reference (N=512). Time taken is the runtime of the complete algorithm 1. The high PSNR values suggest good quality reconstruction, increasing with angular resolution. 47

7.2 Compressed video sizes and times with varying angular resolution in the light slab. The table shows increasing video compression rates with increasing angular resolution. Time taken is the runtime of the video generation code. 48

7.3 Comparison of PSNRs with varying angular resolution due to H.264 video compression. Time taken is the runtime of the video decoding and output plane generation code. The table shows that a constant quality reconstruction is achieved across all angular resolutions. As image quality of reconstructed results increases with increasing angular resolution, therefore video compression provides a viable alternative to generate images of any focus and aperture setting post-trace, with no significant loss in quality. 49

Chapter 1

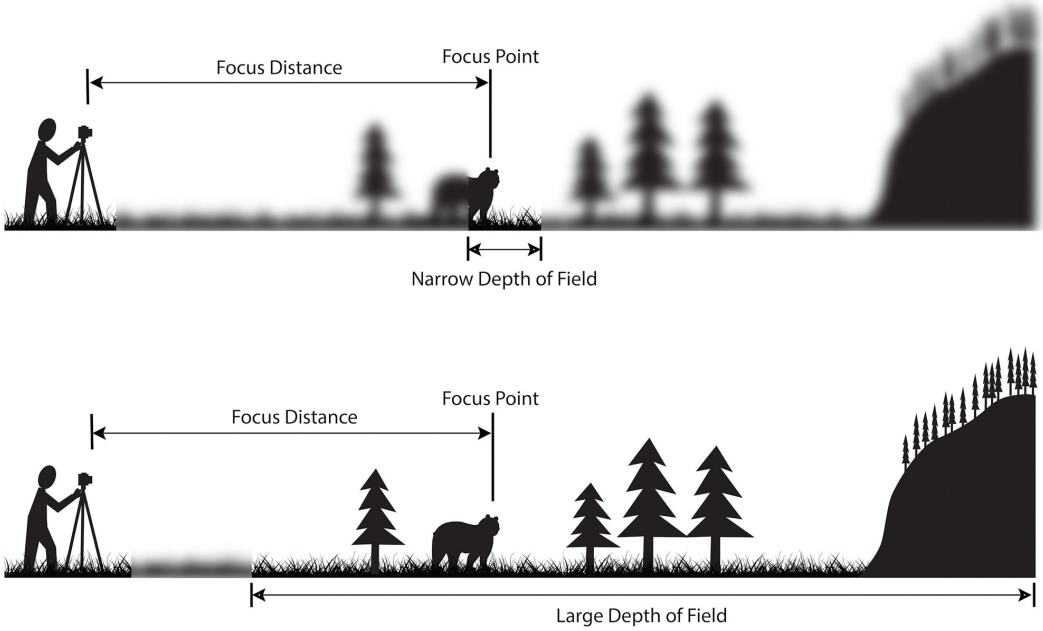
Introduction

Conventionally, 3D rendering techniques like ray tracing and rasterization simulate a pin-hole camera with an infinitesimal aperture and infinite depth of field. Finite depth of field images are rarely used in Computer Graphics, which is in contrast to photography, which uses an assembly of lenses to create images similar to what the eye would see. In real cameras, there is a need for accurate focusing on the subject to get a high-quality image. Quoting from Ren Ng's Ph.D. Thesis [39]: "*The most obvious problem is the burden of focusing accurately on the subject before exposure. A badly focused photograph evokes a universal sense of loss, because we all take it for granted that we cannot change the focus in a photograph after the fact. And focusing accurately is not easy. Although modern auto-focus systems provide assistance, a mistake of a 'fraction of an inch' in the position of the film plane may mean accidentally focusing past your model onto the wall in the background or worse. This is the quintessential manifestation of the focus problem*".

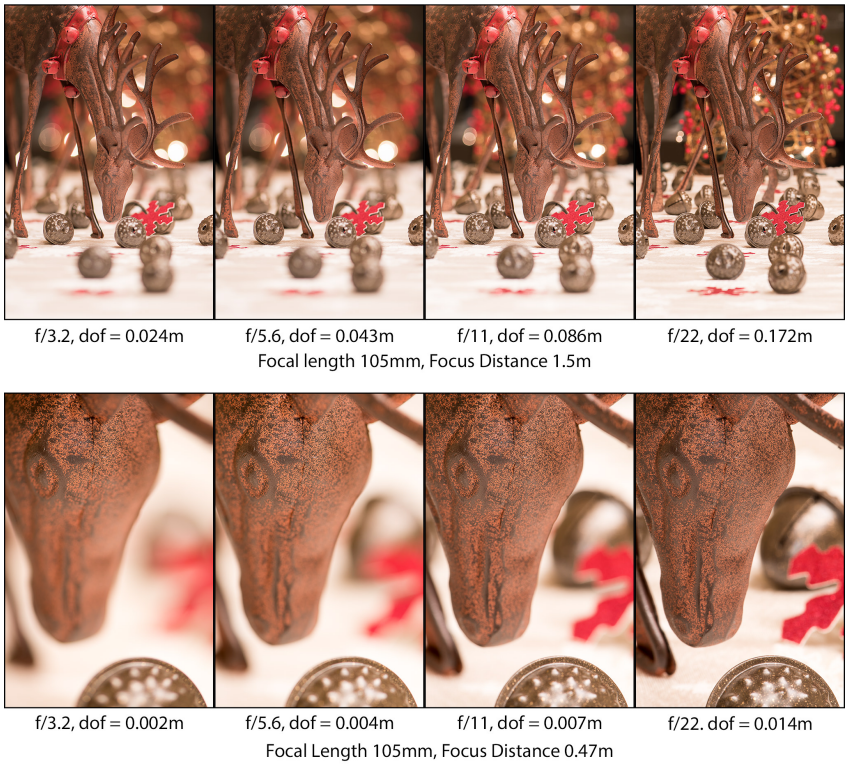
The demand for realism from rendered images has been increasing in recent years, where focus effects like depth of field are very important, along with optical effects like shadows and reflections. For higher degrees of realism, rendered images simulate the behavior of modern digital cameras, allowing for modifications in focus distance, depth of field, aperture etc. Figures 1.1 and 1.2 illustrate the fundamentals of depth of field and it's impact on image sharpness. Figure 1.1(a) depicts that a narrow depth of field results in a small slice of the image being in focus, and a wide depth of field means that more of the scene is in focus. Figure 1.1(b) shows the sharpness of the images increasing from left to right, with decreasing aperture leading to increasing depth of field.

If we capture an image with a digital camera today, the camera tries to come up with the best possible focus setting, given the lighting and scene information. While rendering an image using computer graphics, we are essentially computing the image based on the object space and lighting information. Using some more computational power, we can sample all the rays of light that contribute to the final image, instead of just sampling the sensor. With this, we can go beyond capturing just the scene with one specific focus setting. This is called the light field.

The light field is a vector function that captures information of light traveling in every direction through every point of the world space. In the most general terms, this becomes a 7-dimensional function



(a) Image illustrating the concept of depth of field (From [1])



(b) Examples of scenes with varying depth of field due to varying aperture.

Figure 1.1 Understanding depth of field and aperture, and their effect on images.

Figure 1.2 Image illustrating depth of field in digital light field photography using the sub-aperture representation. Taken from **(author?)** [40].

$L(x, y, z, \theta, \phi, \lambda, t)$, equivalent to describing a hologram. With these 7 parameters, we can describe everything that can be seen. (x, y, z) describe the 3D position, (θ, ϕ) describe the direction of viewing, λ describes the wavelength of light (for color) and t maps to time. If we ignore time and wavelength (λ, t) by keeping a monochromatic light, a static scene and focusing on the spatial structure flowing through the world space, we get the 5D plenoptic function $L(x, y, z, \theta, \phi)$ as described by **(author?)** [3]. This 5D function can be reduced to 4D in free space, i.e., outside the convex hull of the object. The 4D light field can be used to create images with variable depth of field, by changing the aperture and focus distance after capture.

In Computer Graphics a 4D function that defines all light rays passing through an empty volume of space is also called a Lumigraph, coined by **(author?)** [16]. The light field as well as lumigraph are four-dimensional (4D) simplifications of the plenoptic function for static scenes. The Lumigraph often refers to a light field augmented with a partial 3D model of the scene, which is used in the image-based rendering process. It can be used to generate a unique image from any viewpoint within that space.

There are two representations of a 4D light field: the *light-slab* representation [31] and the *sub-aperture* representation [40]. The light-slab representation captures the full volume of light rays sampled between two planes, entering at one plane and exiting through the other. Very high quality and high resolution light fields can be generated using the light slab representation, but they are expensive to compute and have a massive storage requirement.

The sub-aperture representation of light fields efficiently stores a sampled light field with a series of images of the scene from different view positions [40]. This representation is a modification of the light slab to make physical light field capture possible, which is why handheld light field cameras, such as the Lytro camera series, use this representation. This approach, however, suffers from the limitation of pixel as well as the sensor size, limiting both the spatial and angular resolutions.

Light field rendering has mostly been an image-based technique, requiring pre-captured photographs to generate interesting depth-of-field effects in images. In this thesis, we present a method to trace,

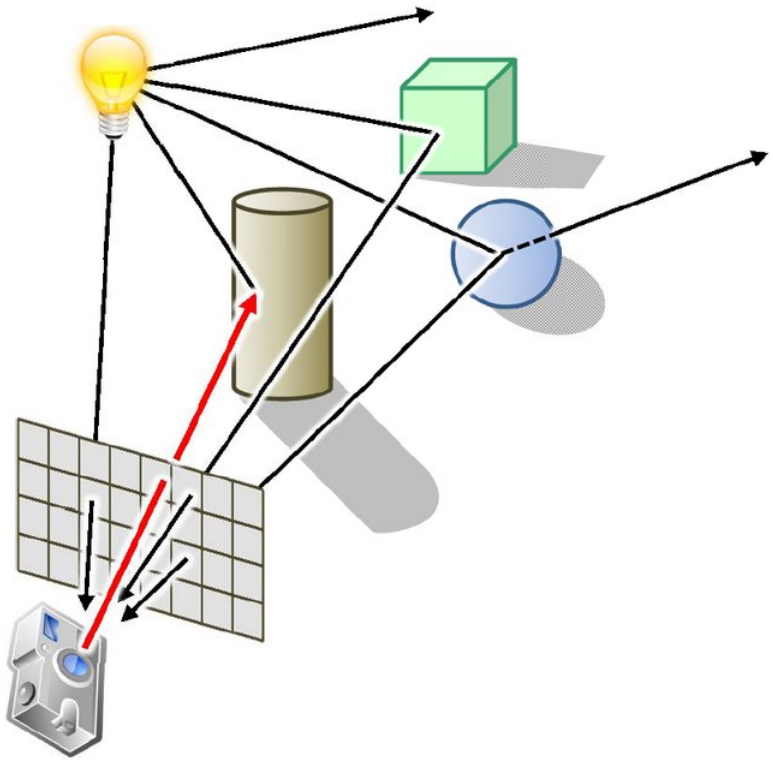


Figure 1.3 Basic ray tracing. From **(author?)** [45].

represent, and manipulate light fields of any 3D scene. We capture synthetic light fields by ray tracing the light slab. We propose synthetic light field tracing as an object-based technique to create physically accurate images of 3D scenes using the light field parameterization.

In a 3D computer graphics environment, greater photorealism can be achieved in images by optical ray tracing methods than other methods like scanline rendering or rasterization. It works by tracing a path from an imaginary eye through each pixel in a virtual screen, and calculating the color of the object visible through it. Effects such as reflections and shadows, which are difficult to simulate using other algorithms, are a natural result of the ray tracing algorithm. The ray tracing algorithm builds an image by extending rays into a scene. Figure 1.3 gives an illustration of ray tracing. Light flows to the scene space, illuminating certain objects. The rays reflected and refracted from the scene space objects go toward the camera sensor, where the image is formed. Ray tracing goes in the opposite way, so as to simulate only rays that we know are going to reach the camera sensor (shown by the red arrow).

Since in ray tracing each ray is treated afresh and is computationally independent, it makes parallel processing feasible. Though it requires handling a lot of data, but has the advantage of superior image quality. The applications which benefit most from GPU-based acceleration are those which have a inherent independence in their subtasks. Ray tracing is an inherently parallel algorithm, as each ray's execution can be treated as an independent subtask for the program to complete. This makes it best suited for implementation on the GPU.

In this thesis, we present the Synthetic Light Field Tracer (SLFT), a GPU-based framework designed to provide flexibility to choose between focus and aperture settings post capture. Each individual setting produces a unique image, and the pipeline traces a large number of rays to generate a collection of such images together, with different focus distances and aperture values, at negligible additional cost. This flexibility is not offered by other tracers, which only produce a single 2D image in one run.

Tracing and storing light fields requires a lot of computation. With SLFT, we have aimed to create a robust, accurate and practical light field (LF) tracer. The structure of our pipeline also allows for its extensibility with more sophisticated raytracing algorithms, and next-generation compression codecs. Our efficient storage representation encodes all the light information of the visible scene. At a small cost, we can generate as many images as required, even up to the complexity of a trillion rays for high detail in images. Our initial studies and implementation of SLFT were presented in [26].

1.1 Motivation

Light-fields enable the user to computationally vary depth and aperture post-capture and have been studied primarily for real cameras. However, there was no existing framework that could render a full 4D light field in a synthetic setting. Rendering synthetic light fields has its applications in:

1. Generation of large, physically accurate datasets for computational photography research. **(author?)** [47] and **(author?)** [48] describe the use-cases for this, as training data for conditional adversarial networks (*cGANs*) for single image refocusing using wide-aperture images, and defocus magnification of narrow aperture images respectively.

2. Making animated movies: Directors/artists can have high degree of control over output frames by precisely calibrating focus and aperture settings for any scene post-render.

3. Collaborative VR (Virtual Reality) applications: Multi-perspective images for different users can be simultaneously generated using LF rendering at a small cost as no re-renders are required inside the LF volume.

4. Computational 3D displays (like tensor displays) may require multi-perspective focus rendering for different users. Foveated rendering in a multi-user environment might also require rendering of multiple focus/aperture effects etc.

If we have access to a compressed light field representation, we can solve all these problems in a logically simpler and more accessible way, requiring much less manual effort overall. For example, if a large dataset of images is obtained computationally, a re-render will be required every time the focus or aperture settings of the scene are changed. The possibility of batch generation of images with all the desired images at various settings of depth and aperture independent of ray tracing calculations holds a lot of promise.

The above motivated this research project, for which we created a framework which allows for tracing and storing a high resolution light field. Having an efficient and parallel implementation of a light field rendering system which allows for post-render control of focus and aperture settings, using a stored representation of the 4D light field, would help us move forward in all the above scenarios, and many more.

1.2 Summary of Contributions

1. A robust and physically accurate light field rendering pipeline, along with theory and analysis of associated light field representations.

2. Solving the challenging issue of light field storage, which results in massive reductions in the memory footprint of storing a 4D light field of any 3D scene. For example, the memory requirement of a fairly complex scene rendered with 260+ billion rays is reduced from almost 1 TB to less than 2GB (table 7.2).

3. An associated implementation which is based on the above principles, allowing the user to create and store high resolution light fields of any scene of his choosing, and using that stored representation to create multiple images at any desired focus and aperture setting.

1.3 Thesis Organization

Chapter 2 provides an overview of prior work done in the fields of Ray Tracing, Light Fields and Computational Imaging. It also examines research done on storage mechanisms of light fields. Chapter 3 explains the different representations of a light field, their advantages and disadvantages, the links between them, and inter-conversion from one to the other. Chapter 4 analyzes the ways to trace light fields of the representations described in Chapter 3, along with discussion on generated results and parallels drawn with other ray tracing frameworks. Chapter 5 describes the light field storage representation, and associated algorithms to store the 4D light field in memory, and it's retrieval to generate images at the required focus and aperture setting. Chapter 6 studies digital refocusing of light fields, and how it is done as part of tracing a light field, both during and after the actual rendering happens. Chapter 7 summarizes all the results of the research work through images and tables, along with observations. Chapter 8 concludes the thesis, with a discussion on future work.

Chapter 2

Background and Related Work

2.1 Ray Tracing

Ray tracing is a rendering technique for generating photorealistic images by tracing light paths and simulating the effects of their encounters with scene objects (Figure 2.1). In 1730, **(author?)** [38] published his book *Opticks*, which analyzed the fundamental nature of light through its reflections and refractions with the help of his *corpuscular* theory of light. His work was the first to formally define and investigate some ray tracing and optics problems. The first algorithm of ray tracing for rendering is described in **(author?)** [5]. While earlier approaches on ray casting stopped at the first hit, **(author?)** [52] introduced recursive ray tracing for simulating global illumination effects, at the expense of greater computational cost. Beam tracing was introduced by **(author?)** [18] to exploit the spatial coherence of objects in polygonal environments. The books by Andrew Glassner ([14] and [15]) are good references on the fundamentals of computational ray tracing algorithms.

Ray tracing was always a slow and expensive process. To speed up the image generation pipeline, acceleration structures were used. These reduced the total number of intersection tests to be performed. This, combined with hardware improvements brought ray tracing to near interactive speeds. The grid data structure organized the scene to small segments. **(author?)** [22] show efficient strategies of creating this structure. KD-Trees and Bounding Volume Hierarchies (BVHs) divided the scene hierarchically to further reduce triangle intersection tests to be performed. **(author?)** [55] proposed an algorithm for construction as well as traversal of KD-Trees on the GPU. **(author?)** [19] improved upon it with better memory allocation techniques for scene size scalability. **(author?)** [51] demonstrated usage of the best KD-Tree construction methodologies and used them in BVH construction. **(author?)** [29] improved on it by using Morton ordering to sort triangles efficiently to optimize both construction and traversal time of BVHs, calling their structure LBVH. **(author?)** [49] derived from both the spatial subdivision philosophy of KD-Trees and the object subdivision philosophy of BVHs to create the Split-BVH (SBVH) which chooses between object and spatial splits by using the surface area heuristic.

(author?) [42] and **(author?)** [27] introduce realistic lens and camera modeling for physically accurate rendering. Each ray is traced into the scene till it hits an object. Some rays hit objects at the focal

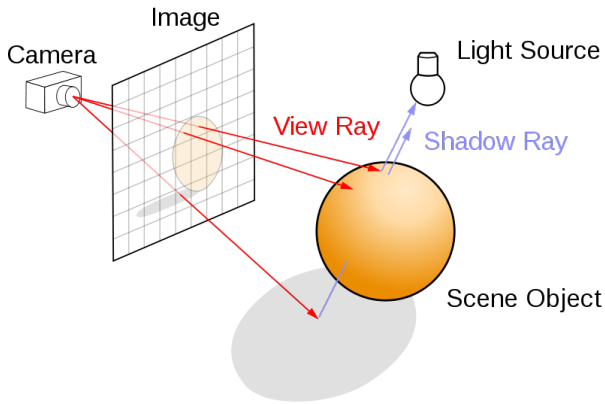


Figure 2.1 The Ray Tracing Algorithm. Image taken from the Wikipedia page on Ray Tracing.

plane of the lens, and some hit objects before or after that. Moving away from the focal plane of the lens creates a circle of confusion, which contributes to the intensities of more than one pixel on a sensor plane, simulating a finite depth of field. **(author?)** [43] demonstrate faster ray tracing methods using parallel processing on programmable graphics hardware. The computational independence of each ray makes parallel processing possible in ray tracing. However, the coherence of rays getting traced together makes an impact on performance of a parallel task.

2.2 Light Fields

A light field is a vector function which defines the radiance of all rays traveling through the world space. A light-field thereby defines the amount of light flowing in every direction. Light fields can be captured from a real-world scene or produced by rendering a 3D model. **(author?)** [3] derived a single plenoptic function to describe the structure of the information in the light impinging on an observer. As one cannot simultaneously look at a scene from every possible point of view for every wavelength at every moment of time, but through the 7D plenoptic function one can examine the structure of the information that is potentially available to an observer by visual means. This idea was independently explored by **(author?)** [36] in their paper *The Photic Field*. The full light field is described as a 7D function, but **(author?)** [31] gave a sampled 4D representation of light fields by extracting 2D slices in appropriate directions (figure 2.2). However, their approach required rendering or acquiring a large number of images with a high sampling density, which consumes a lot of time and memory. A similar representation based on the 4D plenoptic function was also given by **(author?)** [16], called the Lumigraph.

A 7D light field $L(x, y, z, \theta, \phi, \lambda, t)$ is equivalent to describing a holographic movie, i.e. the intensity of light seen from any viewpoint, at any time, as a function of λ. Similar representations have also been given by **(author?)** [8] using epipolar volumes and **(author?)** [6] for horizontal-parallax-only holographic stereograms. An epipolar volume is created with a series of images generated by moving a

camera in a defined distance increment along one direction, and a holographic stereogram is formed by exposing a piece of film to an array of images captured by a camera moving sideways (from [31]).

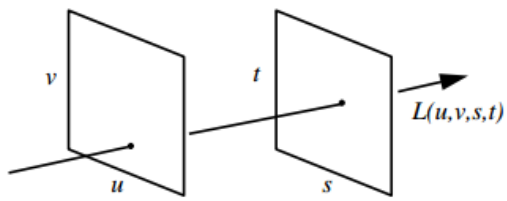


Figure 2.2 Slab representation of the 4D light field given by Levoy et. al.

In a lecture in 1846, **(author?)** [10] explained that light, like other forms of radiation occurs as "lines of force", similar to "magnetic lines of force", which means that light also exists as a field, like the magnetic field. **(author?)** [13] properly defined the concept of a light field in 1936, which gives the amount of light traveling in every direction through every point in space. The plenoptic camera finds its genesis in the integral photography methods pioneered by **(author?)** [33] and **(author?)** [21]. Many variants of integral cameras have been built in the last century, and described in books on 3D imaging such as **(author?)** [41] and **(author?)** [23]. A different approach to capturing light fields in a single exposure is an array of cameras as described in **(author?)** [54]. Our work picks up its first threads from the famous work of **(author?)** [4], who developed an optical model of a plenoptic camera that generates the image of a scene when viewed from a continuum of possible viewpoints bounded by the main lens aperture (figure 2.3). **(author?)** [12] extended this idea to add super-resolution to the refocusing algorithm of a plenoptic camera using image processing techniques (like color filter arrays) on raw color data.

(author?) [40] used a lesser complex optical model of plenoptic camera and demonstrated that acquired light fields provide unique photographic capabilities including ability to digitally refocus the scene after exposure, extend the depth of field, and alter the viewpoint and perspective. They developed a hand-held plenoptic camera (called Lytro [2]) based on the lenslet-based design of **(author?)** [4] (figure 2.4).

Recently, end-to-end image refocusing pipelines based on cGANs were proposed by Sakurikar et al. for refocusing single images using blur estimation DNNs (deep neural networks) [47] and simulating shallow depth of fields in camera-captured images through defocus magnification [48].

2.3 Obtaining Light Fields

Currently, light fields are obtained mostly by physical capture, as it is popular as an image-based rendering technique. In **(author?)** [31], a method using a rotating gantry was described. Strict control was required on the lighting, and also the person capturing had to stay well clear of the setup, to avoid

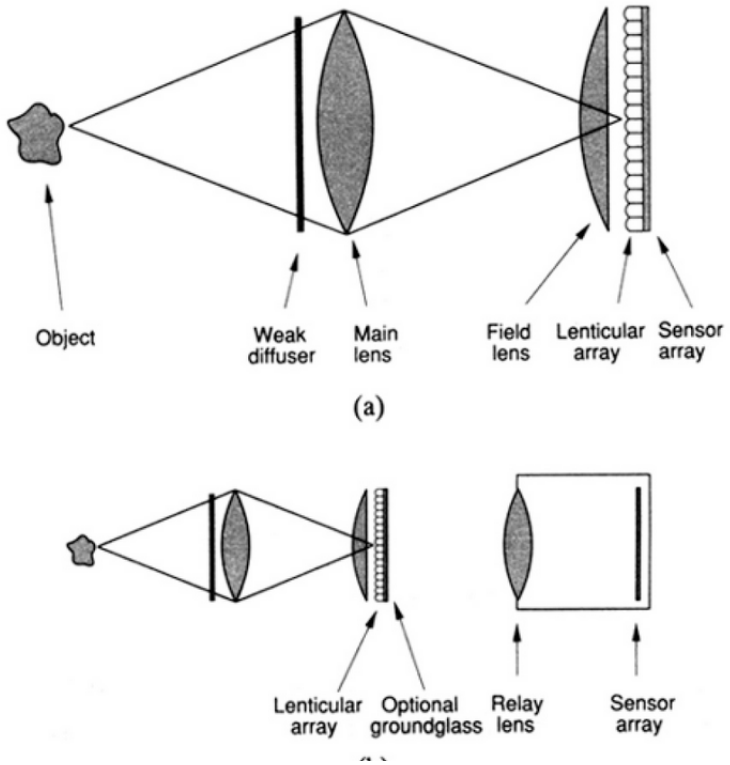


Figure 2.3 Optical system of Adelson and Wang's plenoptic camera [4]

Figure 2.4 The Lytro Illum 2015, based on the camera model by **(author?)** [40].

unwated shadows. A similar method of stepping through known camera positions and serially taking pictures was also given by **(author?)** [20]. They also came up with a reparameterization to demonstrate post-process refocusing (generally known as synthetic aperture photography). **(author?)** [54] developed a huge camera array setup to record light fields. The development for this started in 1999, and a comprehensive study was published in 2005. Figure 2.5 shows the setups, with images taken from the original papers. **(author?)** [11] present an optical analysis of light field rendering, defining the key elements of the imaging system including the aperture, circle of confusion, depth of field and hyperfocal distance, with analogy to a conventional optical imaging system. For a minimum and maximum depths of the scene, they derive optimal constant depth and the hyperfocal distance to achieve the best rendering quality.

The measurement or recording of the light field can be done by using a Light Field Camera. A standard photographic camera gives a 2D representation of the light rays reaching the lens at any given position. The image sensor of the camera records the sum of brightness and colour of all light rays arriving at each individual picture element or a pixel. A light field camera has the ability to record not only brightness and colour values in the 2D imaging sensor, but also the direction or the angle of arrival (AOA) of all the light rays at this image sensor. This information provides an additional capability to reconstruct the path of each light ray before reaching the camera, and makes it possible to generate a 2D image of the scene at any given focus depth and aperture value.

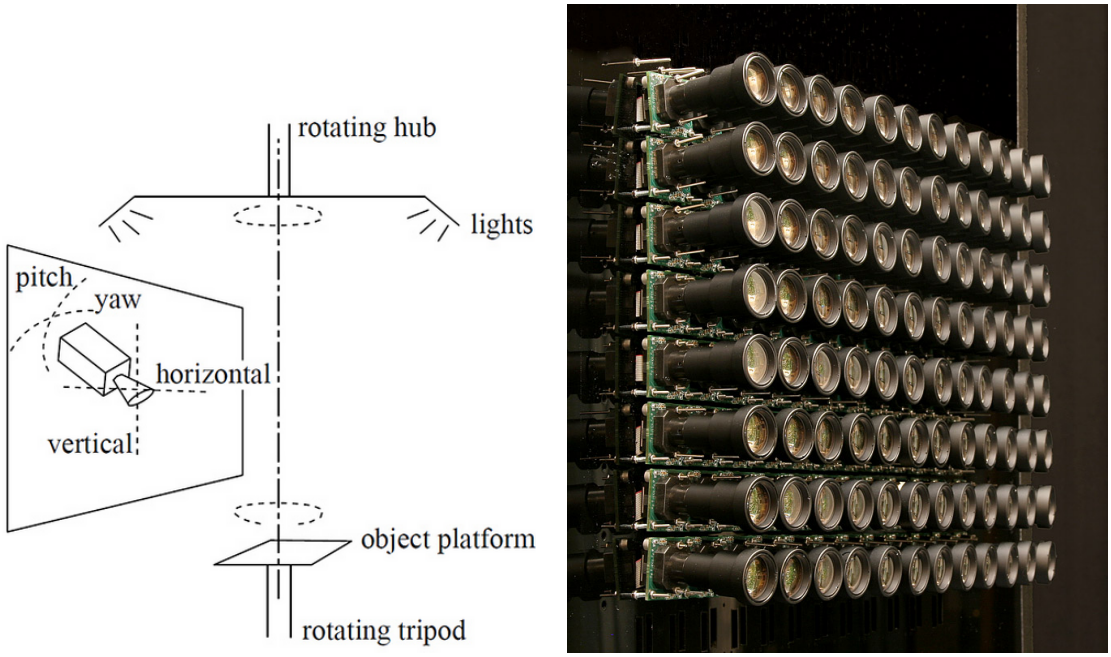
In 2005, **(author?)** [40] further miniaturized light field technology and put the power to capture light fields in the hands of the end user with a hand-held camera capable of capturing light fields and refocusing them in software, as described above. This was called the Lytro camera series. Figure 2.4 shows a newer model of the camera.

With the work presented in this thesis, we aim to make light field capture more accessible in the synthetic domain, with the setup capable of both capturing the light field, as well as using it to generate the image of any 2D slice within the light field volume.

2.4 Compressed Light Field Storage

Any light field rendering application faces the challenge of data storage. Rendering quality depends on both the number of images and their spatial resolution. Since a very high number of images is required to ensure photorealistic results, compression becomes essential when dealing with a dataset of large sizes. **(author?)** [35] highlighted these issues, and proposed compression of light field images using modified video compression techniques and disparity compensated image prediction, which works well for small light fields storable in memory. However, light fields of high angular resolutions cannot be directly stored in memory and compressed afterwards.

In recent times, video compression technology has improved quite a lot, from MPEG-1 in the 1990s to recent standards like H.264/AVC, proposed by **(author?)** [53]. Bitrates have stayed low, resolutions have increased manifold, and more advanced arithmetic coding techniques are being used. **(author?)**



(a) The gantry setup of capturing light fields. Image taken from **(author?)** [31].

(b) A model of the Stanford multi-camera array by **(author?)** [54].

Figure 2.5 Earlier methods of capturing light fields.

[28] extended the idea of video compression (using the Multi-View Coding (MVC) amendment to H.264 given by **(author?)** [37]) for compressing light fields to be used in 3D holographic displays. A setup of projection modules emitting light rays onto a holographic screen is required for this. In the past few years, some research effort was also done on high-efficiency coding of captured light field sub-aperture images as well as all-in-focus images to reconstruct an image with specific focus and aperture settings (**(author?)** [46]). A study was recently done on a scalable light field compression algorithm by selecting sparse subsets of sub-aperture images (or views) and using them for reconstruction of the light field in the continuous Fourier domain (**(author?)** [17]. Studies was also done by selecting subsets of sub-aperture images targeted towards image refocusing quality (**(author?)** [50]) and compression techniques based on depth image based view synthesis (**(author?)** [24]).

We wanted to solve the problem of light field rendering and compression for purely synthetic settings. In this thesis, we present an algorithm to compress light fields of very high sampling density efficiently using modern video compression techniques. This simplifies the problem so that standard video compression algorithms are applicable to light field data storage.

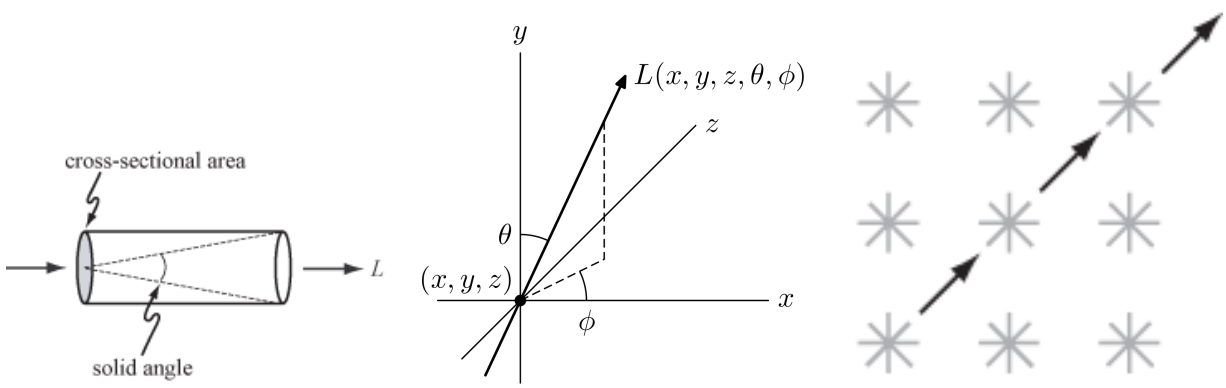
2.5 Summary

Most of the research on light field rendering uses real imagery and the sub-aperture representation. We present a raytracing-based framework based on these concepts, which uses the light slab representation to generate high quality synthetic light fields. This can be used to create images of any focus and aperture setting at acceptable runtimes. SLFT has the capability of storing very high resolution synthetic light fields (even up to a trillion rays) with very high compression factors of up to 1500x.

Chapter 3

Light Fields

The light field constitutes all light rays flowing through every point from every direction, in a global 3D space. **(author?)** [13] coined the term light field for studying surface illumination with artificial lightings. Mathematically, light field is a function that describes the amount of light in radiance along light rays traveling in every direction through every point in empty space. **(author?)** [3] coined the plenoptic function that describes the intensity of each light ray in the world space as a function of viewing angle, wavelength, time and viewing position. The term *plenoptic* got derived from word roots *plen*, meaning complete or full and *optic* meaning eye or view.



(a) (b) (c)

Figure 3.1 The 5D plenoptic function, representing the flow of light through 3D space (from the Wikipedia page on Light Fields). (a) Radiance L along a ray can be thought of as the amount of light traveling along all possible straight lines through a tube whose size is determined by its solid angle and cross-sectional area. (b) Parameterizing a ray by position (x, y, z) and direction (θ, ϕ). (c) Radiance along a ray remains constant if there are no blockers. This allows for a 4D representation of light fields, due to redundancy in the plenoptic function.

A light field can be completely characterized by a 7D plenoptic function $L(x, y, z, \theta, \phi, \lambda, t)$, where (x, y, z) define every position of light ray, (θ, ϕ) give all the angles of viewing, λ gives information of every wavelength and t is the time variable. For a monochromatic light (i.e. fixed λ) and a static scene (fixed t), this function reduces to a 5D function. In fact, a human visual sensor or eye only observes, samples and projections of the plenoptic function which encapsulates the full information that is communicated to the observer. **(author?)** [31] described light field as radiance that is a function of position and direction, and which could be described by a 5D plenoptic function. A light ray, therefore can be parameterized in a 3D space by position (x, y, z) and direction (θ, ϕ). Figure 3.1 illustrates this concept (taken from **(author?)** [30]).

The light field function, that encodes the entire visual information of a defined volume in free space, includes spectral and temporal information of light in addition to the 3D position and angular distribution of light rays. This 4D light function completely characterizes the flow of light in free space in a scene with explicit illumination.

For 4D representation of light fields, lines or light rays are parameterized by their intersections with two planes in arbitrary position (figure 2.2) along the direction of travel. The coordinate system on the first plane is (u, v) and on the second plane is (s, t), by convention. Any oriented line or light ray is defined by connecting a point on the (u, v) plane to a point on the (s, t) plane. Generally, u, v, s and t are restricted to lie between 0 and 1, and thus points on each plane are restricted to lie within a convex quadrilateral. This representation, depicted by two planes is called a *light slab*, which is one of the two representations of the light field. A light slab represents the light ray entering one quadrilateral plane and exiting through another quadrilateral plane. For this representation one of the planes may be placed at infinity, and the lines can be parameterized by a point and a direction.

Conventional cameras can record only a 2D image of a scene out of the total light received by the camera photo sensor. But a 4D light field can capture the directional information of the light on each location of the sensor and hence can measure the amount of light traveling along each ray that reaches the sensor. The capture of an additional two dimensions of data allows for application of ray tracing techniques, providing an additional capability of computing synthetic images from acquired light fields.

A Light Field created in a virtual environment is called Synthetic Light Field (SLF). Traditionally light fields are captured by taking images of a real world scene but with increasing applications of SLFs in virtual reality (VR), animated movies and computational photography, we aimed to move the LF creation to the virtual domain in our work. We generated an efficient synthetic light field ray tracing framework that is capable of catering to current application requirements of SLFs.

The following sections explain the two different ways to represent a light-field, i.e. the light slab and sub-aperture representation. We explain image formation through each of these using their optical setup. We go on to establish the equivalence between the representations, and discuss their benefits and limitations.

3.1 The Light Slab Representation

In the light slab representation, uniformly-sampled lines are parameterized between two planes in arbitrary positions for efficient geometric calculations [31]. This can be visualized as a 2D collection of perspective images of one plane viewed from the other plane. From Levoy and Hanrahan's 1996 paper on light fields, "*In free space, the light field is a 4D, not a 5D function. An image is a two dimensional slice of the 4D light field. Creating a light field from a set of images corresponds to inserting each 2D slice into the 4D light field representation. Similarly, generating new views corresponds to extracting and resampling a slice.*"

Figure 3.2(a) explains the optical setup for the light slab. The space between the two limiting planes is the extent of the light field which is captured. The number of discrete points from where light rays are shot towards the scene determines the spatial resolution of the light slab (i.e. the sampling of the world space). This is the [u, v] plane in our case. Let this number be M. The number of pixels (say N) on the [s, t] plane determines the angular (or directional) resolution of the light slab. We can sample the light slab from either side, and use this representation to generate N^2 images of the scene, each of $M \times M$ pixels, or M^2 images, with each of $N \times N$ pixels.

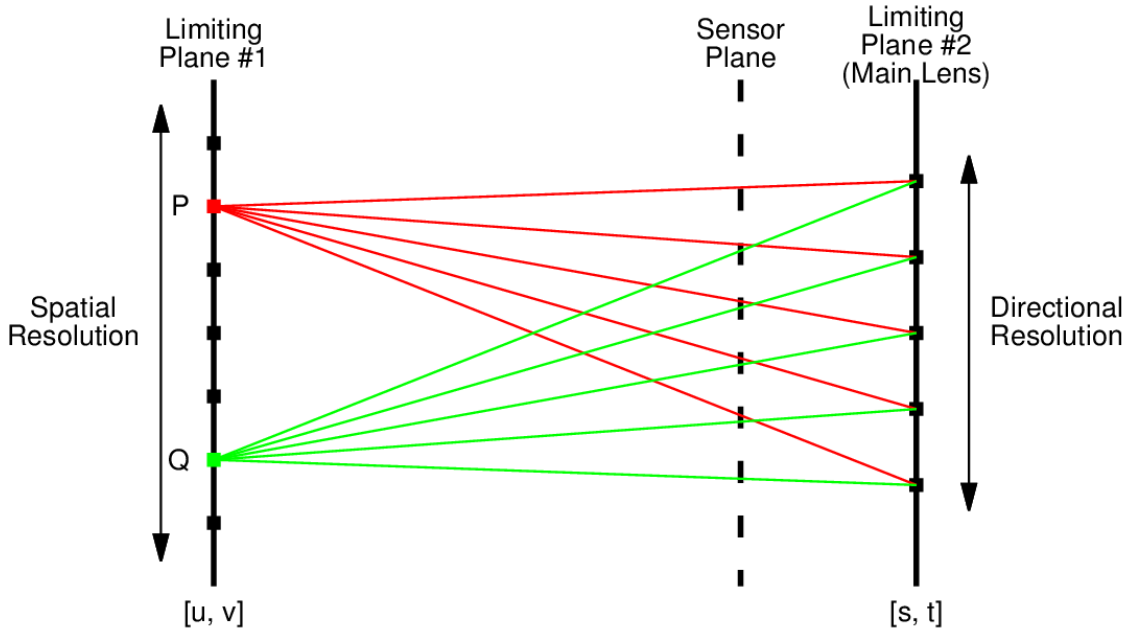
In our implementation of the light slab, one of the planes of the 4D light field [s,t] is fixed to be the thin lens. The number of discrete points sampled on the lens plane govern the size and shape of the simulated aperture. In the former sampling case, each image would contain the mapping of one world space point through M^2 viewing angles, and we would obtain N^2 such images, one for each sampling area of the world space. In the latter case, each image would be the full scene as seen from any one point on the lens plane (i.e. [s, t] plane), and we would obtain M^2 such images, one for each view angle on the [s, t] plane. The exploitation of this symmetry of the sampling space to create a compact representation of the 4D light field is present in chapter 5, where figure 5.1 extends the representation shown in figure 3.2(a).

By tracing all pairs of rays between the two limiting planes in figure 3.2(a), we can trace as many focal planes as we want in a single rendering step. Any new sensor focus position can be simulated as an intersection of the ray volume with the sensor plane.

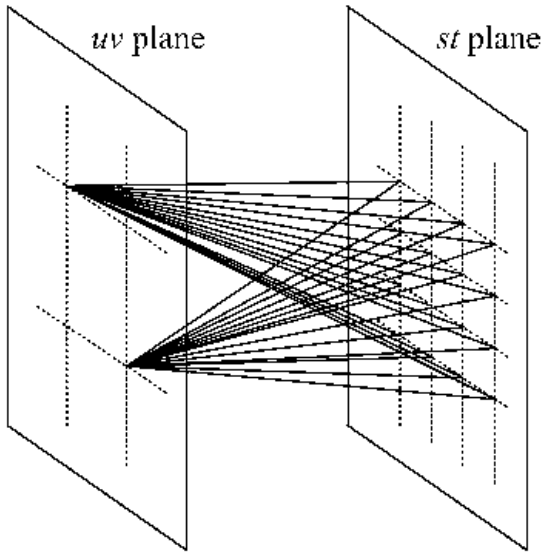
3.2 The Sub-aperture Representation

The sub-aperture representation is another approach of sampling the 4D light field. This representation is based on a tightly packed grid of very tiny lenses, called the microlens array (MLA) [40]. The actual sensor lies behind the MLA, at the focus of the microlenses. Any specific focus and aperture setting can be created post-capture by a series of image processing operations which is referred to as digital refocusing [40].

Figure 3.3 shows the optical setup, with a 4-lenslet microlens array for demonstration. The number of lenslets determines the dimensions of each sub-aperture image (and consequently the final refocused



(a) Light rays are traced starting from points of an arbitrary [u,v] plane behind the lens (for ex. P and Q) to all discrete locations on the lens. The light rays projected onto the lens aperture are refracted appropriately and traced into the scene, which is to the right of the main lens.



(b) A 3D illustration of the light slab representation. Taken from **(author?)** [32].

Figure 3.2 Ray diagrams for the Light Slab Representation.

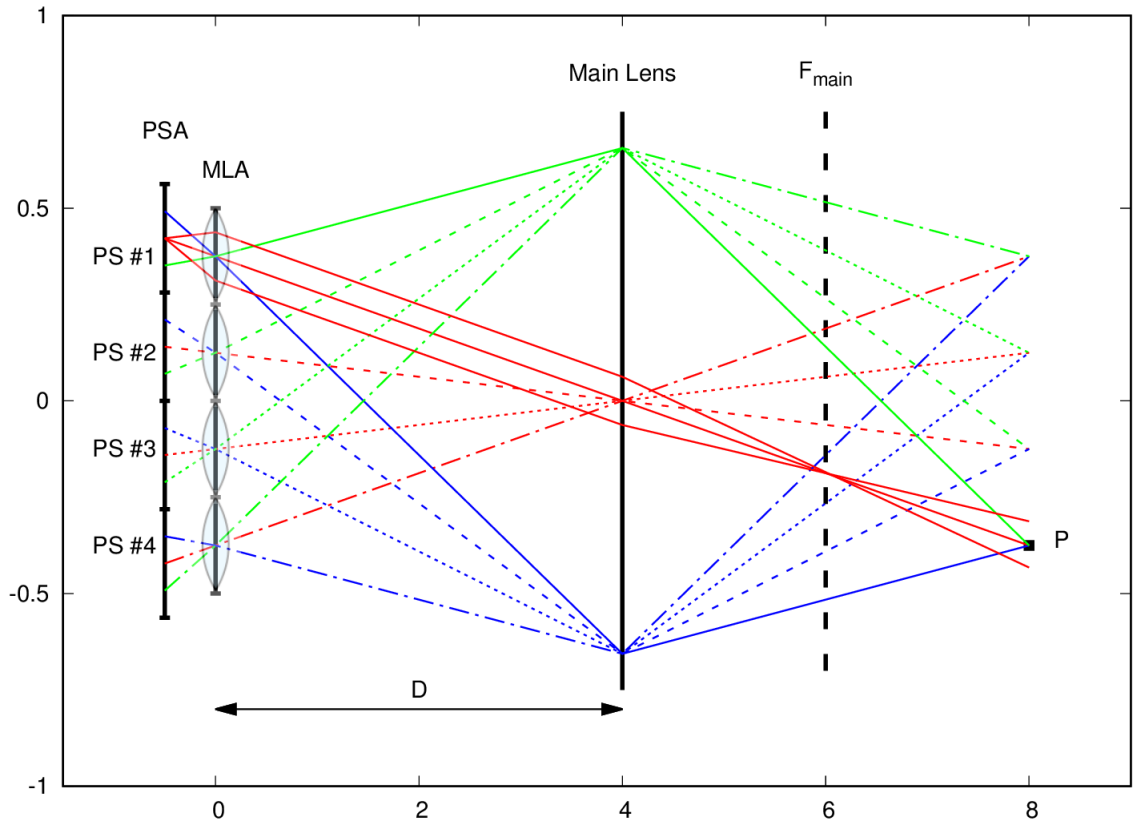


Figure 3.3 Ray diagram for the Sub-Aperture Representation. The red, green and blue rays depict rays from different view angles. A microlens point conjugates with a point on the world space, and a photosensor pixel corresponds to a region on the main lens. This is illustrated by three red rays originating from PS #1 going to different locations in the object space. Best viewed in color.

image), i.e. spatial resolution. Let this number be M. The number of pixels (say N) behind each microlens defines the directional resolution of the image, i.e. the number of viewing angles. We use this representation to generate N^2 images of the scene, each of $M * M$ pixels.

Figure 3.3 shows rays of three different colors - red, blue and green, which denote different pixels under a lenslet, i.e. different viewing angles. We can see that in the ray diagram, all the different-colored rays meet at distinct points on the right of the main lens, and all the same-colored rays meet at a single point on the main lens. This phenomenon is as explained by **(author?)** [40], where a microlens point conjugates with a point on the world space, and a photosensor pixel corresponds to a point/region on the main lens respectively. Although, the former argument is only correct to a certain extent. We can see the two additional rays in PS #1, which go to extreme corners of its corresponding microlens, and do not meet all the other rays which go out from PS #1 and meet at point P on the object in figure 3.3. Thus, each pixel on the photosensor carries information of a region in the world space, not a point. But there is an important feature of this setup that compensates for the error. As the number of microlenses is high, so each microlens is vanishingly small, and thus the spot size on the right side is very small. Also, the distance between the MLA and main lens becomes such that it is finite for the main lens, but seems infinity from a microlens perspective.

When the microlenses are small enough, and kept at a distance D that is more than the hyperfocal distance of a microlens, the region where its rays spread is very small (figure 3.3), and can be approximated to a point. Thus we see the entire scene in acceptable focus in all sub-aperture images of the scene (figure 3.4).

3.2.1 Light Field Photographs

On tracing the setup explained in figure 3.3, we get a full light field photograph, where each 2D block of $N * N$ pixels represents one point in the world seen from N^2 different angles. The size of the full image would be $MN * MN$ pixels. To generate a sub-aperture image, we extract the same pixel from each $N * N$ block, and put them together to generate the $M * M$ image. The same is done for each pixel in the $N * N$ block to generate all sub-apertures.

Figure 3.4 shows another scene with a bike model kept in two settings - one with the front-most plane in focus (3.4 (a) and 3.4 (b)), and another with the focus behind the bike (3.4 (c) and 3.4 (d)). The two images inside each of fig A and fig B are sub-aperture images of opposite viewing angles - the top-left corner and the bottom-right corner respectively. As we can see, the images (c) and (d) seem to move a lot with respect to each other, and the images (a) and (b) do not. This also follows from the fact mentioned above that adding all images without shifting, focuses the scene exactly where it was placed. So the object should not move a lot across different sub-apertures in an infocus setting, and the movement increases as we keep going further away from focus.

(a) Top-Left View

(b) Bottom-Right View

(a) Top-Left View

(b) Bottom-Right View

Figure 3.4 Sub-aperture images of a scene with a bike model.(a) and (b) show extreme sub-apertures when the bike is in focus, and (c) and (d) show extreme sub-apertures when the bike is out of focus.

3.3 Equivalence of the two Light Field representations

A real light field camera (like Lytro) uses the sub-aperture representation because its sensor is an array of photodiodes, each of which can store just a single color value. All the rays coming from a single point and carrying directional information meet at a single point on the [u, v] plane. A microlens at that position splits the converging rays in different directions, which are stored at different pixels on a sensor sitting at the focal plane of the microlenses. From the MLA to the main lens, the space of rays that both representations operate on is the same (figure 3.2(a) & 3.3). This brings us to the conclusion that while rendering light fields, the sub-aperture representation is equivalent to a light slab representation with the second limiting plane of the light field at the microlens array's position. In a synthetic setting, we can create a buffer for each pixel on the back limiting plane in the light slab, and store the pixel values corresponding to a point/region viewed from all directions in it.

Chapter 4

Tracing Light Fields

In computer graphics, ray tracing is a rendering technique which generates an image by tracing the path of light as pixels in an image plane. Ray tracing is a well-known method of virtual 3D scene visualization that is used for the rendering of photo-realistic images by simulating the natural behaviour of light rays as they hit and reflect objects in the scene. Historically, rendering light fields has required pre-captured imagery of the scene from various view positions, and these images are used to create the images of the scene at the desired focus and aperture settings. As a software system used the image data to create further meaningful images, it was classified as an image-based rendering (IBR) technique. On the other hand, object-based rendering (OBR) is a technique that allows us to mathematically describe the different parts of the scene space as objects, and uses a rendering algorithm like ray tracing to create the final image of the scene. If we look at some of the important use-cases of light field rendering (as outlined in section 1.1), all of these cases benefit from a technique which provides us with the freedom to re-render the 4D light field using the object specification, even if we want to use IBR for generating the final image focused at a specific depth and a given aperture value.

In this chapter, we present synthetic light field rendering (SLFR) as an object-based rendering strategy, and explain the different methods of tracing a 4D light field. We discuss the algorithms involved to generate the different light field representations using ray tracing, and explore methodologies to extract images at custom focus or aperture settings, with relevant examples.

4.1 Tracing the Light Slab

Figure 4.1 shows the light slab representation, and also a ray-space diagram for the same (from **(author?)** [44]). As explained in chapter 3, the 4D light field is described as $L(u, v, s, t)$. In a 2D cartesian ray space, this can be described as $L(u, s)$. The points P_1 and P_2 in figure 4.1(a) map to the blue and red lines in 4.1(b) respectively. A slice of the [u, s] plane is one view of the scene at a specific focal and aperture setting.

Algorithm 1 explains how to capture any custom focus and aperture setting of a given 3D scene. The geometry and camera configuration parameters are input to our framework in the form of a simple

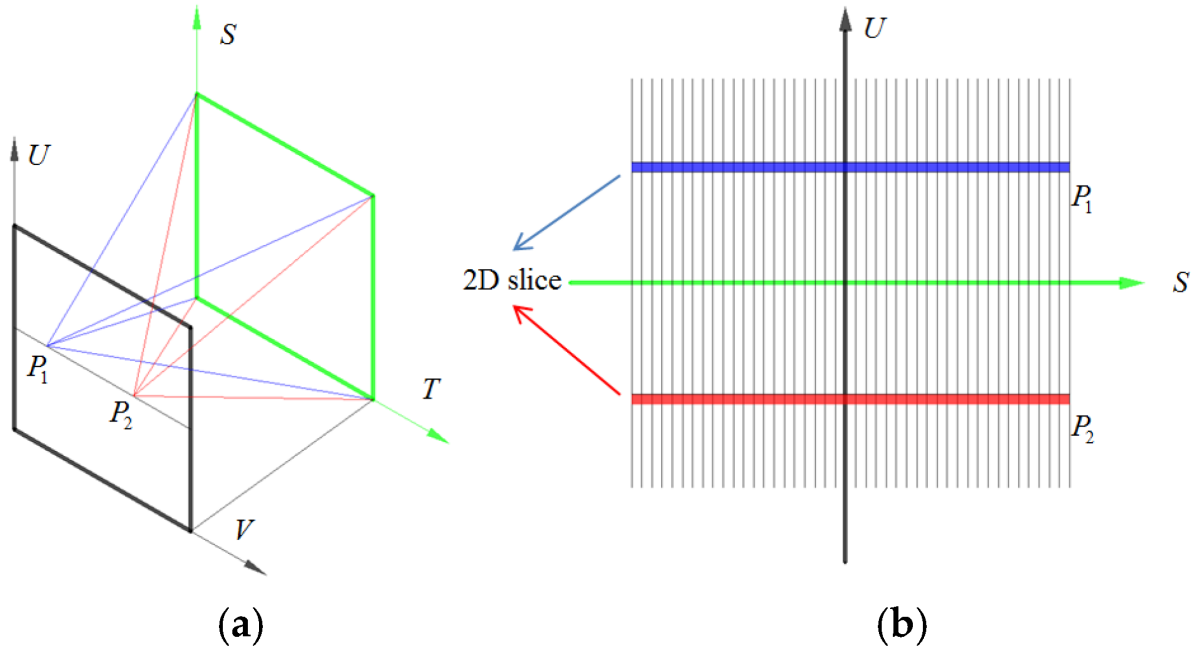


Figure 4.1 (a) Light Slab, the 2-plane parameterization of the light field. (b) The 2D ray-space diagram.

scene file specification. Once the geometry is loaded, we create a BVH [25] using an SAH-based split algorithm [34] on the CPU. The BVH is then flattened recursively into a 1-dimensional list of nodes and transferred to the GPU along with other structures holding the geometry and texture data.

If the dimensions of the [u,v] and [s,t] planes are $M * M$ and $N * N$ respectively, our approach will trace M^2N^2 rays. We serialize over the [u,v] plane, and give the GPU N^2 rays to work on in parallel at a time. After all the structures are filled and all temporary buffers are initialized, the core rendering process is divided into three parts, each implemented as a GPU kernel (steps 7-9 of algorithm 1).

Kernel #1 - Tracing primary rays: The primary rays are launched from one point the [u,v] plane, which is then mapped using the thin lens formula from the lens plane to the object space. These rays are created in the kernel and traced into the scene. This involves a stack-based BVH traversal of the ray, returning the index of the triangle hit by it. R encodes the aperture mask of rays for the sensor plane(s) to be generated. Before a ray is traced, it is checked that it lies within the maximum aperture bound settings of the sensor plane(s).

Kernel #2 - Computing color values: We pick the rays that return a valid triangle, and fetch the triangle geometry for those. All shadow rays (from the ray's hit point to all light sources in the scene) are also traced. The final RGB value at the hit point for the source ray is determined by computing the lighting equation, and summing up the component values from all light sources returning a successful shadow ray trace. We use the Blinn-Phong model [7] for this purpose.

Algorithm 1 SLFT - Tracing a Light Slab

```
scene_config ← ParseSceneParams()
scene_data ← ReadSceneFile(scene_config)
B ← MakeSceneBVH(scene_data)
R ← (s, t) rays in aperture bounds
Initialize GPU Structures
for all points on the (u, v) plane do
    rays ← TracePrimaryRays(scene_data, B, R)
    cols ← ComputeColors(scene_data, rays)
    P ← CreateSensorPlane(scene_config, cols)

Normalize sensor pixel values
Output sensor plane P to image
```

Kernel #3 - Generating an output image: This kernel populates the output plane p. It uses the color values of the rays from second kernel. For each ray which intersected an object, multiple ray-plane intersection tests are done, one for each of the desired sensor planes at different depths. Each of these tests finds out the exact hit point of the ray with the output plane, and subsequently the pixel number where the ray hits the plane, and the color value is added to the correct pixel locations at each plane. We also check that only the rays within the aperture mask ($R[p]$) of the plane are considered for plane p.

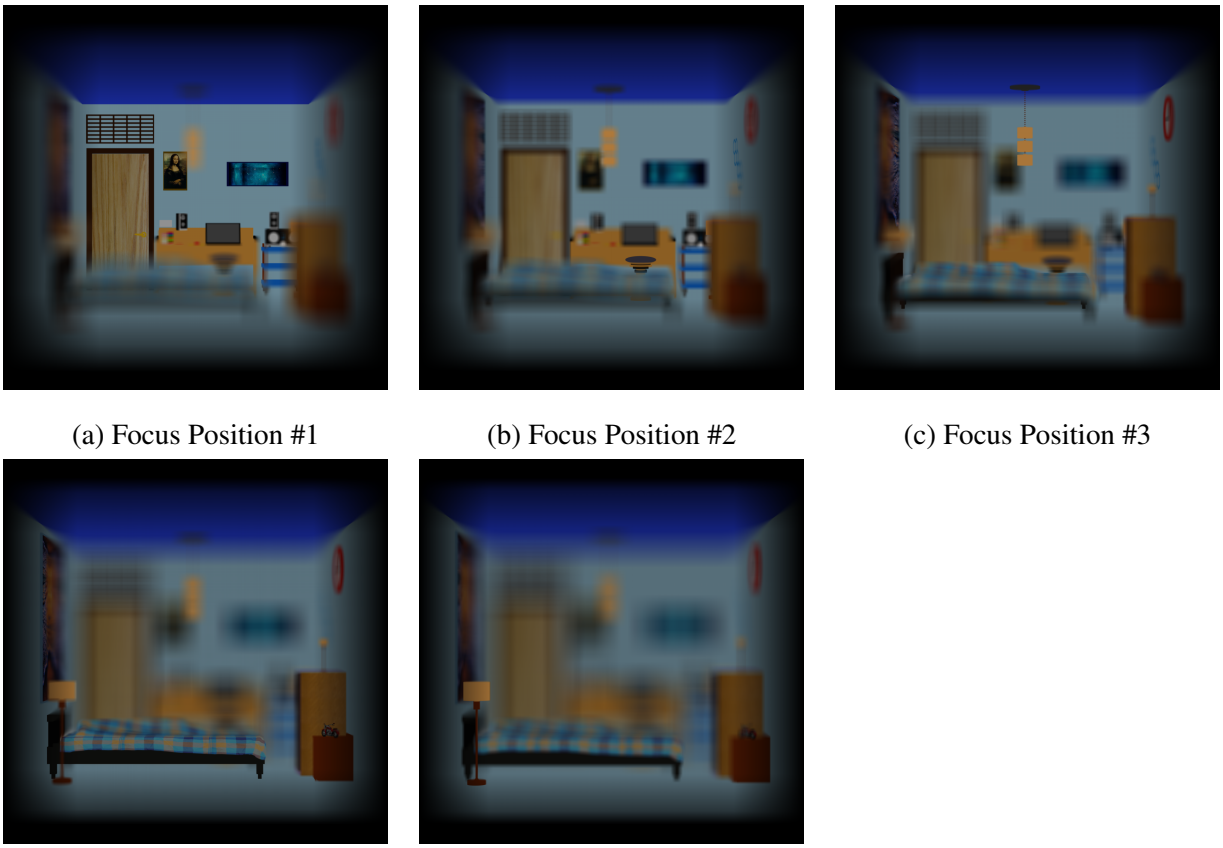
Once the three kernels are finished for all the M^2N^2 rays in the light field space, the rays which hit the scene have added their color component at all the desired sensor plane locations. The image is then normalized by a constant factor of N^2 (= total #rays / sensor plane size) to scale pixel values back between 0 and 255, and then is written to a separate binary image file. This is the output images of SLFT at the desired focus and aperture setting.

We can create focal and aperture stacks by applying kernel #3 (for image generation) iteratively over multiple planes at not much additional cost. This sets our method apart from other methods that trace only a single output image.

4.1.1 Results

Figure 4.2 shows an example of a focal stack with a bedroom scene, and figures 4.3 and 4.4 show two aperture stacks. In figure 4.3, decreasing aperture of the main lens from left to right means less number of rays going through to the object space, resulting in dimming of the image. But an aperture decrease also leads to increasing depth of field, i.e. a wider band on the Z-axis where objects are in focus. This is illustrated in figure 4.4, where the normalization of the sensor plane is based on the aperture opening of the main lens, leading to all images being of the same brightness. We can create focal stacks by

changing the Z-value of the sensor plane, and aperture stacks by varying the f-number of the main lens and choosing a aperture shape, for example square, circle, diamond etc.



(a) Focus Position #1 (b) Focus Position #2 (c) Focus Position #3

(d) Focus Position #4 (e) Focus Position #5

Figure 4.2 Focal stack of bedroom scene generated using SLFT. The figures show shifting of focus from the back of the bedroom to the front in going from (a) to (e). Generated using light slab sampling algorithm.

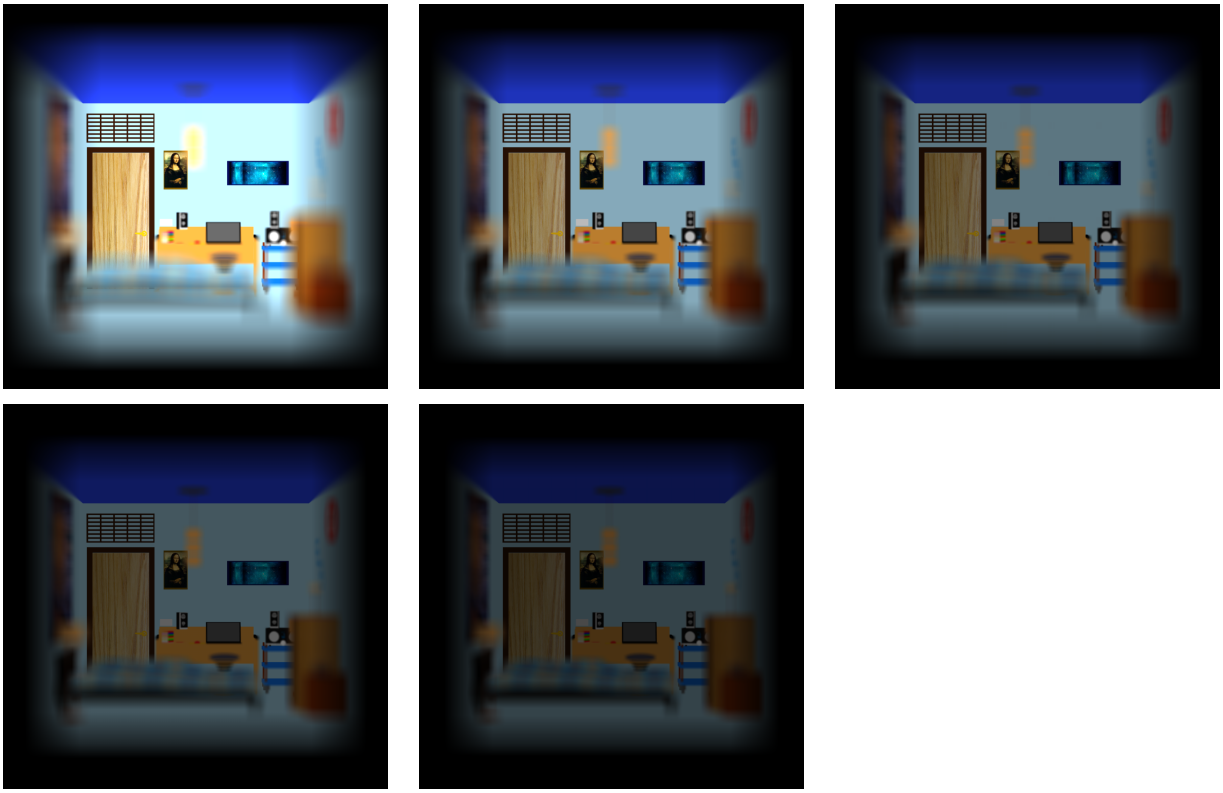
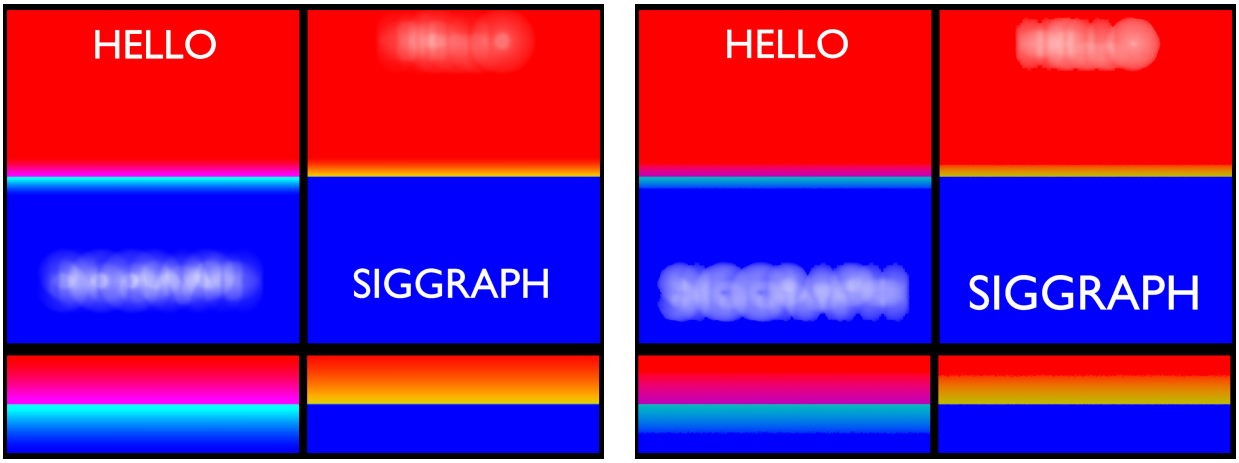


Figure 4.3 Aperture stack of bedroom scene generated using SLFT. The figures show decreasing aperture size going from left to right, resulting in dimming of the image.

Figure 4.4 Aperture stack of bedroom scene generated using SLFT. The figures show decreasing aperture size going from left to right, but the dimming is compensated for, demonstrating increasing depth of field.



(a) Our result

(b) POVRay's result

Figure 4.5 Our light field ray tracing compared with a starndard rendering framework. Figure (a) has 2 renderings by our method at different focus depths, and figure (b) has the corresponding images generated by POVRay.

4.1.2 Comparison with other ray tracers

Figure 4.5 shows a comparison between our result a rendering of the same scene with a similar setup in POVRay. The scene consists of a blue foreground plane and an edge aligned red-green background plane. The top row shows two near and far focused images created by our method and the same images generated using POVRay. The bottom row shows zoomed in versions of the top row. The blocky artifacts in the text segments and the jagged transition from blue to cyan and yellow to red in the POVRay images confirm the superiority of our method. Although the images generated here may not be of physically realistic scenes themselves, but the effects observed can translate to any scene having similar elements.

4.2 The Sub-aperture Representation

Algorithm 2 explains how we can directly trace the 4D sub-aperture representation of light fields, given any 3D scene. The core geometry loading, BVH creation, flattening and traversal algorithms remain the same, as do the fundamental three kernels described in algorithm 1. The major difference is in the ordering and creation of the rays to trace. Using figure 3.3 as reference, we first calculate the photo-sensor array (PSA) dimensions, which are based on the micro-lens array (MLA) dimensions, and on the fact that PSA lies at the focus of the microlenses. The main lens is kept at a distance greater than the hyperfocal distance of the microlenses. As the rays are refracted twice going from the PSA to the world space, we start the rays from MLA, and backtrace it to a PSA point. The ray is mapped from a MLA point to a world point, and all the rays going towards the same world point are traced together.

The sampling of the main lens has a 1:1 mapping with the sampling of the PSA, and the color returned by each ray refracted by the main lens is back-traced to one PSA pixel.

Once all the steps are done, a full light field photograph of the scene is obtained, and the image of any specific focus or aperture setting can be generated using digital refocusing techniques. The digital refocusing methods will be explained in detail in chapter 6.

Algorithm 2 SLFT - Sub-aperture Tracing

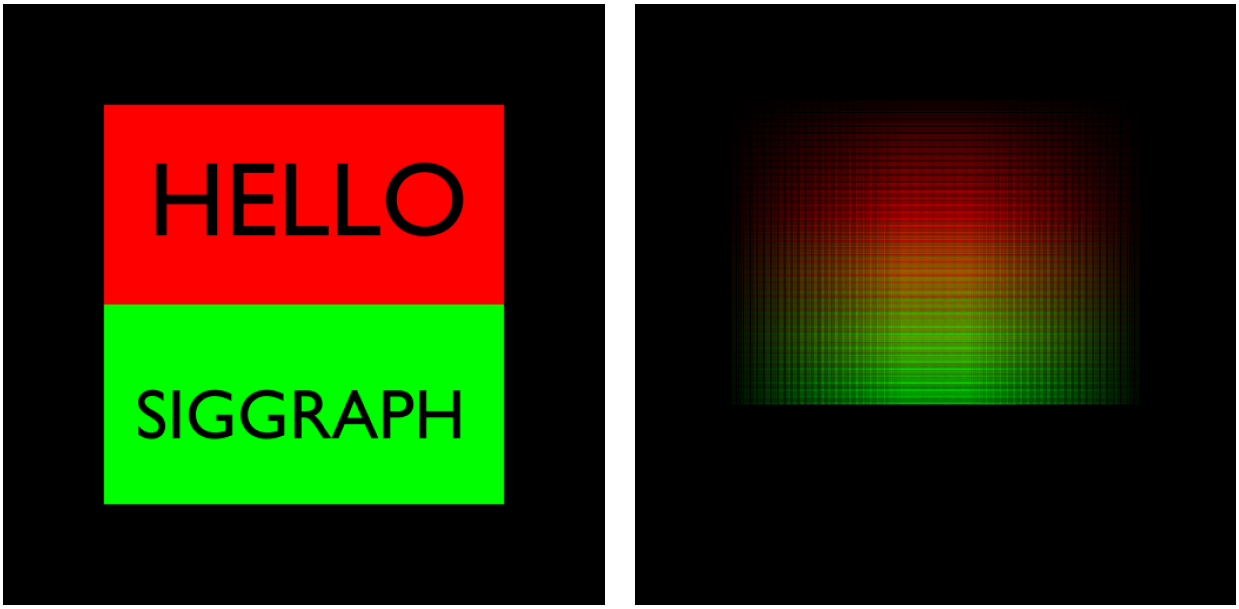
```
scene_config ← ParseSceneParams()
scene_data ← ReadSceneFile(scene_config)
B ← MakeSceneBVH(scene_data)
R ←PSA rays in aperture bounds
S ← InitializeLFSensor()
Initialize GPU Structures
for Each microlens in the MLA do
    ray_set ← GetRefractedRaysfromML(scene_data, B, R)        ▷ Get Main Lens Rays to Trace
    rays ← TracePrimaryRays(scene_data, B, ray_set)
    cols ← ComputeColors(scene_data, rays)
    S ← UpdateSensorPlane(S, ray_set, cols)
Normalize sensor pixel values
Output sensor plane to image
```

4.2.1 Results

An example of two full light field photographs is given in figure 4.6. The images are of 8192x8192 resolution, with 512x512 spatial resolution, and 16x16 directional resolution. For demonstration purposes, we take a fully planar scene, half of which is red, and the other half green, with text written on the planes to show the focusing effects more clearly. Figure 4.6 (a) shows the photograph when the whole scene is kept in an in-focus setting, as also shown in figure 3.3. Figure 4.6 (b) shows the planar scene pushed away from its focus pair, and thus is fully out-of-focus.

4.3 Analysis of the two LF representations

To programmatically verify our theory of equivalence between the light slab and sub-aperture representations, we coded both these methods and verified the results to be very similar. We observed that points of error lie in the sub-aperture representation's shift-sum refocusing logic. Both approaches are limited by the sampling resolution of the light field, but the slicing logic of the light slab (as explained in section 3.1) doesn't approximate in any way while generating the output image. Hence, for the purposes of synthetic rendering, light slab representation gives a more accurate result, given some extra computational budget. It is able to capture all of the aspects of the sub-aperture representation (including creation



(a) Plane in focus

(b) Plane out of focus

Figure 4.6 Two full light field photographs of the same scene at different depths, as generated by our framework.

of sub-aperture images), and has a minimal memory footprint. Furthermore, the sub-aperture representation also suffers from another shortcoming related to object size magnification. The position of the sensor behind the lens determines the magnification of captured objects. Changing the focus position thereby causes a change in magnification: sometimes manifesting as a zooming effect. The sub-aperture representation cannot account for this, because all images are captured for one focus position. The light slab parameterization is physically accurate, and magnification is naturally visible in the target images.

Our tracing algorithm for light slabs runs on-the-fly, which means that it picks up a [u,v] point, and completes tracing and sensor updates, and resets temporary buffers before picking up a new [u,v], which results in a minimal storage overhead. Storing a full light slab is a challenge, and may be required if the goal is post-capture focus and aperture manipulations. With sub-aperture images generated from light slabs, we can store the light field representation, but only up to a limited angular resolution. To break through this barrier, we need a compressed representation of the light field space. In the following chapter, we present a simple and efficient technique to solve the problem of light field storage for high angular resolutions of the light slab.

Chapter 5

Storing Light Fields

In this chapter, we present an efficient technique for storing light fields with very high angular resolutions, to generate images of desired focus and aperture settings post-tracing. Some form of compression is required because storing all the M^2N^2 color values in raw form is not possible due to storage space limitations. For example, a value of $M = 1024$ and $N = 64$, gives us a total of $1024^2 * 64^2 = 4$ gigarays, each with an RGB value of 3 bytes. Thus we would need a storage space of 12GB for raw color values. For an even higher resolution light field (for eg. $N = 1024$), we have a 1 teraray light field, i.e. a 3TB space required, which is impractical to be kept in memory. If we compress the raw images generated by the tracing pass using an image compression codec like JPEG, we can reduce our memory footprint by ˜20-50x, which still takes about 100GB or more. This is not sufficient for storing light fields of such high densities. We need more compression for a viable light field storage algorithm.

5.1 Visualization of the light slab as video

To make the complex problem of light slab storage simpler, we chose video compression for data storage. Figure 5.1 shows a visualization of the light slab representation, showing the portion of the scene as viewed by each sample on the [u, v] plane. We observe that for high spatial resolutions, if we view what each point of the [u,v] plane sees in the world space as an image in itself, the sections of the world space viewed by each of the points on the [u, v] plane is very small. The data between the neighboring samples also has significant overlap as the shift in perspective from a single point on [u, v] to the next is very small. This allows us to view each of these sections as a sequence of frames in a video, which means that we can use a standard video compression algorithms to store the raw color values of the full 4D light field traced into the scene. Video compression utilizes both standard image compression algorithms to decrease storage, as well as the lack of variance in frames of the video for inter-frame compression using key-frames. For our setup, we choose the H.264/MPEG-4 AVC standard, as it is one of the best video codecs in terms of compression quality and size. Also it is compatible with most of the devices today, and is extremely efficient for video content with little motion.

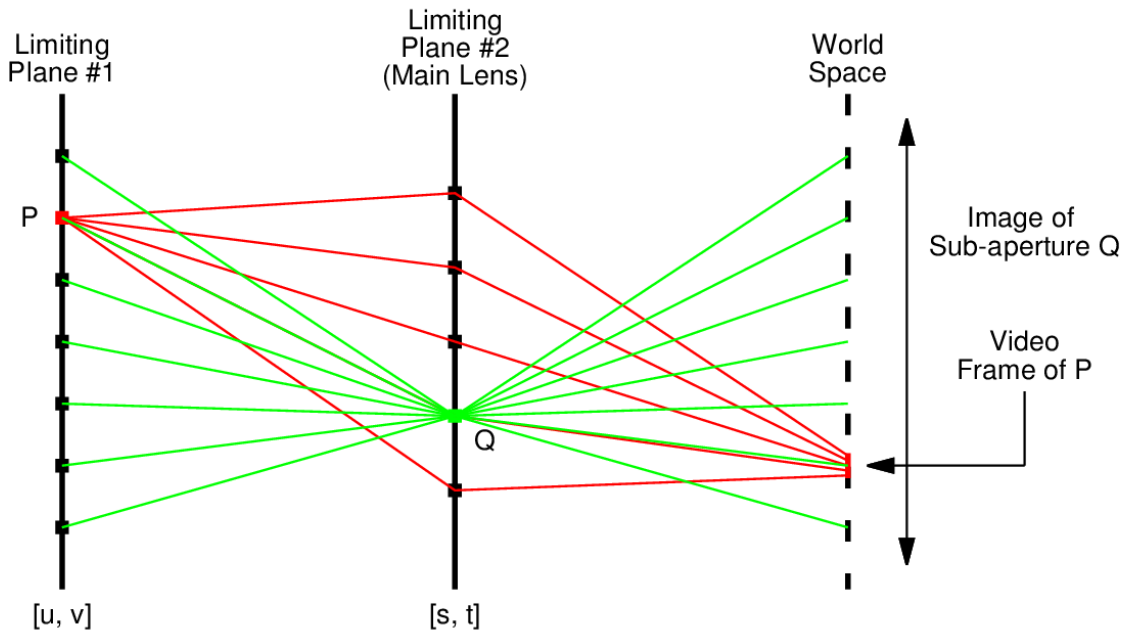


Figure 5.1 Ray diagram for Visualisation of video frames of light slab storage. Sampling the [u, v] plane gives small sections of the world space as frames of the video, and sampling the [s,t] planes gives sub-aperture images as video frames. High spatial resolutions lead to small perspective shifts, and efficient video encoding. Best viewed in color.

Our approach for implementing video compression for light field data storage, apart from reducing complexity, provides extensibililty for application of advanced video codecs with superior coding and decoding schemes.

5.2 SLFT with Video Storage - Algorithm

For light field storage, we use our SLFT framework with the same light slab parameterization (Section 4.1) and scene setup. In algorithm 3, the major difference is that before the ray tracing process begins, a video object is initialized with the H.264 video codec. In each iteration of the 2D loop, for each point (i, j) on the [u, v] plane, the first two GPU kernels described above are executed (steps 7 to 9). Instead of creating a sensor plane, we take the current color buffer, which has all the world information that is seen by point (i, j), and store it as a frame in the video (step 12). At the end of the 2D loop, we have created a H.264 video containing M^2 frames (one for each point on the [u, v] plane), each of $N * N$ pixels (the size of the [s, t] plane) in a MKV container. Algorithm 3 summarizes this approach.

The next step is generating output frames at a certain focal depth. As we have exactly the same information that we had in the light slab implementation (i.e. the colors that each ray gets after getting traced into the scene) in a compressed video format, we decode the video to retrieve RGB frames. Then we initialize sensor plane(s) on the GPU, and update it, giving it 1 video frame per iteration. After all the M^2N^2 rays are traced and the output planes are normalized, we have the output image with the scene focused at any specific focal depth, similar to the output described in section 4.1. We present this approach as an extension of our SLFT. The video approach allows us to sample the full light field as a linearized version of a 2D stack of 2D images. This algorithm can also generate more than one output image at a time, as we have similar data in each iteration as section 4.1, albeit in a different form. This provides us with the additional flexibility of just needing to run the second step (i.e. output generation) again. Algorithm 4 summarizes this.

We use OpenCV's Video API for video encoding and decoding, which makes use of x264 software's open source library libx264 with FFMpeg.

5.3 Visualizing the New SLFT Pipeline

In figure 5.2, we present a visualization of the new SLFT pipeline, complete with the addition of our light field storage technique (initially presented in [26]). Steps 1 to 3 involve creating the scene model and tracing M^2N^2 rays towards the scene, as described in chapter 4. The input to step 1 is a scene geometry file with 3D triangle and texture information. SLFT processes this file and populates the memory buffers, and creates the hierarchical BVH acceleration structure (Step 2). Step 3 initializes the GPU kernels with rays in the light slab representation and traces the rays through the 3D space. In step 4, the color buffers returned from step 3 are stored in a compressed, ordered video representation, which is our representation of the traced light field. The algorithm for implementation of the above steps

Algorithm 3 SLFT - Tracing a Light Slab with video storage

```
scene_config ← ParseSceneParams()
scene_data ← ReadSceneFile(scene_config)
B ← MakeSceneBVH(scene_data)
R ← (s, t) rays in aperture bounds
Initialize GPU Structures
V ←Initialize H.264 MKV Video Container
for all points on the (u, v) plane do
    rays ← TracePrimaryRays(scene_data, B, R)
    cols ← ComputeColors(scene_data, rays)
    Normalize color values                ▷ Add all data for one u, v point as one frame in the video.
    AddColorBufferToVideo(V, cols, scene_config)
Close Video Container and Write to Disk
```

Algorithm 4 SLFT - Generating Sensor Planes from Video

```
V ← GetVideoFromFile()
R ← (s, t) rays in aperture bounds
Initialize GPU Structures
while VideoFile not empty do
    F ← GetFrame(V)
    cols ← GetColorsinBounds(F, R)
    P ← CreateSensorPlane(scene_config, cols)
Normalize sensor pixel values
Output sensor plane P to image
```

is described in section 5.2. To generate the images of the scene at specific depth and aperture setting, Steps 5 and 6 are required. Step 5 reads the video structure to retrieve frames. This is equivalent to sampling a 2D slice from the 4D light field volume. Step 6 uses the extracted frames, along with the sensor plane specification to get the pixel values of interest from the frames.

It is important to note that steps 1 to 4 are computationally expensive, but have to be done only once for a scene configuration. Steps 5 and 6 can be repeated with the video structure, and are inexpensive to compute.

Our pipeline has these two parts as separate scripts. When starting the image generation process without a video representation, steps 1 to 4 are mandatory to generate the video file of sub-aperture/sub-array images, as they are required by the steps 5 and 6 to generate images at defined focus and aperture settings. In our implementation, we created our own file input format which allows the user to describe multiple sensor planes in one go (complete with the definition of the depth and aperture value for each), along with the location of the video file. Up to 100 planes can be specified in a single run of algorithm 4, and one output image is created per plane. The output images have an option to be stored as raw PPM images, or as compressed PNG files.

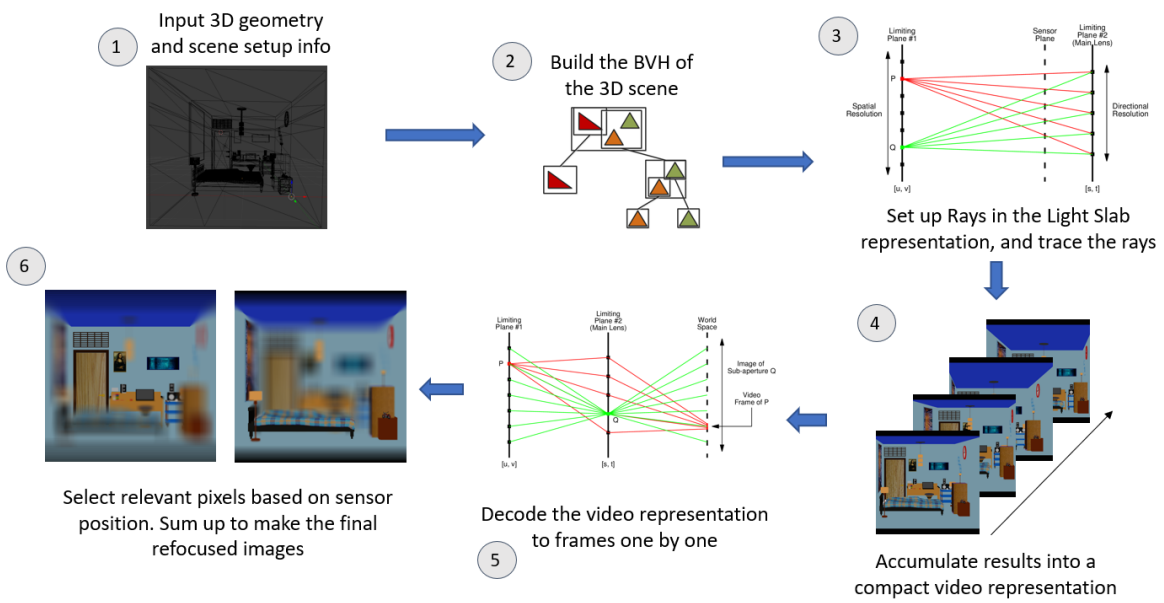


Figure 5.2 Diagrammatic representation of the SLFT pipeline with storage.

Chapter 6

Digital Refocusing of Light Fields

Digital refocusing is a technique that has been used to generate images focused to different depths/distances from a camera, after a single camera shot is taken. The technique is based on light field rendering, and exploits the fact that a image is a 2D integral projection of a 4D light field. In digital refocusing, a stored form of the light field representation is used to create images of a scene at any given focus and aperture setting. In this chapter, we demonstrate two approaches to refocus images – slicing-based refocusing and shift-sum refocusing.

6.1 Slicing-Based Refocusing

This approach is based on the light slab representation, and requires the ray data to be in memory before the image can be created. The name "slicing based refocusing" comes from Levoy and Hanrahan's reference of generating new views being similar to slicing the light field at a particular depth.

Figure 6.1 shows a light slab setup with rays going towards the world space from the [u, v] plane. The red and blue rays originate from one point in the [u, v] plane to converge to a small region in the world space, whereas the green rays converge on a [s, t] point, going all over the world space. A sensor plane S is also shown. For each ray traced towards the scene, we must first find out which object was intersected, and calculate it's RGB color value (as explained in algorithm 4.1). For each of those rays with valid color data, a ray-plane intersection test is done with the sensor plane S. At this stage, we must also check that the ray passes through the defined aperture of the sensor plane. The exact hit point of the ray with the output plane, and subsequently the pixel number where the ray hits the plane is calculated. The color value is then added to the correct pixel location of the plane.

This process can be repeated whenever needed if we have the stored video representation, and in each run of this process we can have as many sensor locations as required. The algorithm for generating refocused images from the stored 4D light field is explained in section 4.

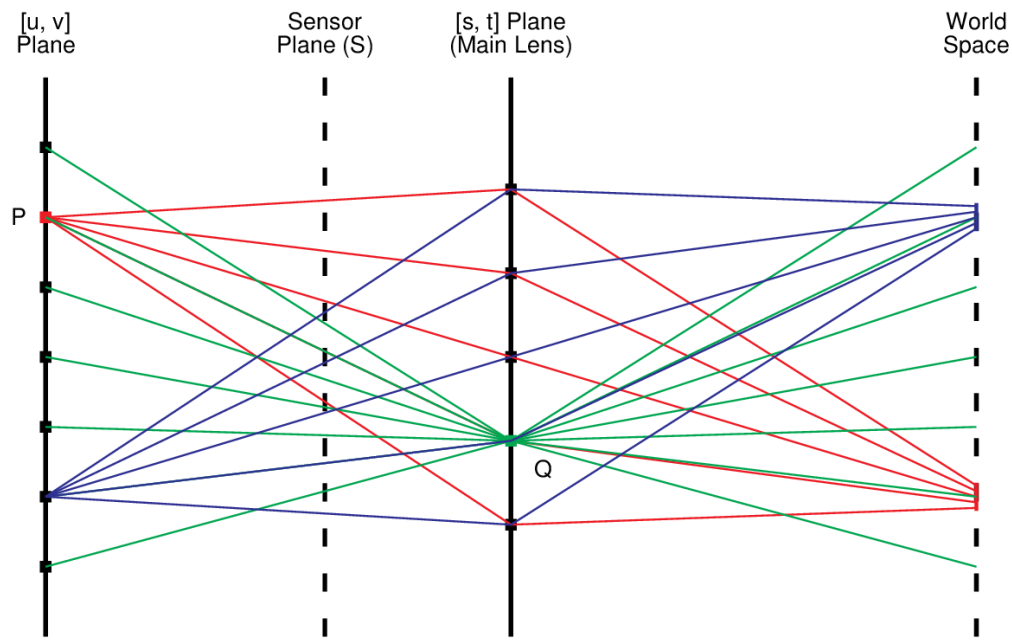


Figure 6.1 Slicing-Based Refocusing.

6.2 Shift-Sum Refocusing

This approach is based on the sub-aperture representation. In shift-sum refocusing, we use the light field photograph (section 3.2.1), made up of small grids of pixels, called sub-array images. Each of these images has the information of a small area of the world space, and selecting the same pixel from each such grid of the light field photograph forms the view of the world space, as seen from a specific angle (i.e. a sub-aperture image). Once we have rendered the light field photograph using either of algorithms 1 or 2, a synthetic photography equation is used to resample the light field function and generate a synthetic focal plane (**(author?)** [40]). The equation is as follows:

$$\overline{E}(s', t') = \iint L\left(u', v', u' + \frac{s' - u'}{\alpha}, v' + \frac{t' - v'}{\alpha}\right) du'\, dv' \tag{6.1}$$

Equation 6.1 represents the irradiance value at a point (s', t') on the synthetic image plane. Evaluating it for all points on the image plane would give us the refocused image. Conceptually, this reduces to shifting and adding the sub-aperture images according to the chosen focal plane.

Figure 6.2 shows how the light field recorded on the photosensor pixels is sampled and processed. Each grid box depicts a bundle of rays coming from a small region of the world space to one pixel on the photosensor. The intercept of the ray with the microlens plane and the lens plane determined its (x, u) position on the ray-space diagram. The figure depicts two colored sample boxes (blue and gray),

angular resolutions. The comparison of results for different scene geometries are presented in Tables 7.1, 7.2 and 7.3. The spatial resolution for all images generated in this thesis is set at $M = 512$.

Table 7.1 compares the PSNR and SSIM values for variation in angular resolution (N) from a minimum of 16 to a maximum of 512. It also presents the variation in time taken to run algorithm 1 for the same variation in angular resolution. The comparison is presented for all the three different scenes.

We see high PSNR values for the scenes, going above 50 even from $N = 128$ for the Bike and Cabin scenes, demonstrating high quality image reconstruction.

Table 7.2 compares the compressed video sizes and time taken for the generating the same with variation in angular resolution (N) in the range of 16 to 512. This is done for all the three different scene geometries.

We also observe from the table that for simpler scenes, the algorithm is able to compress images to a very high ratio, even more than 1500x in some cases. Also, for scenes with more detail, video sizes still stay at a reasonable size (less than 2GB).

Table 7.3 depicts the effect of H.264 video compression on PSNRs for varying angular resolution. The comparison is made between images computed from video decoding and sensor sampling with our on-the-fly SLFT results. Here t is the runtime of video decoding and output plane generation code. These studies and the similar comparisons are presented for the bike scene, cabin scene and the bedroom scene.

7.3 Observations and Analysis

This section details the observations from extensive studies carried out to validate and demonstrate the efficacy of our SLFT framework. The section analyses the results obtained to establish the application of the studies presented in this thesis for studies on ray tracing of synthetic light fields. The detailed studies have been carried out through several runs of presented algorithms and the effect on each of the parameters due to variation in angular resolution gave been observed.

Table 7.1 details the comparison of PSNRs due to varying angular resolution in the light slab, for three different scenes of varying geometries. It can be observed from the table that with increasing angular resolution, the PSNR value goes on increasing. This establishes the efficacy of our framework SLFT in terms of achieving high angular resolution. This is evident for all three types of scenes, i.e., from simple bike to the cabin and finally the complex bedroom scene. The time taken increases as the resolution becomes higher as well as with increasing complexity of the scene geometry.

Table 7.2 shows the variation of time for video generation, the size of video generated as well as the compression ratios achieved at different angular resolutions. It is evident from the table that for increasing resolution requirements, the size of the video as well as the time taken for video generation increases. Compression ratios increase with increasing angular resolutions. A similar trend is seen for each of the three geometries under comparison.

Table 7.3 shows PSNR values of results generated from the stored 4D light field as a video compared to its on-the-fly counterpart. The values show a constant quality reconstruction of the full stack of images at different focal depths across all angular resolutions. This shows that quality keeps increasing hand in hand with images generated without any storage, providing video compression as a viable alternative to generate images of any focus and aperture setting post-trace, with no significant loss in quality.

The comparison of results tabulated in table 7.1, 7.2 and 7.3 for three different scene geometries of three different complexities show that SLFT can perform well for scenes of all complexities, in terms of light field compression as well as reconstruction quality and has only a very small setup time. As we had expected, the heaviest portion of the algorithm is the tracing and color computation portions, as having N^2M^2 rays to trace is the biggest bottleneck of the light field approach. Also, we see both the quality metrics (PSNR and SSIM) increase rapidly in the first few rows of table 7.1, but slow down as we start having enough rays to capture the visual information. Analyzing table 7.2 we note that compression ratios increase with angular resolution, due to step size for the video frames, decreasing across the image world space, leading to more uniformity in each video frame (as world space area covered reduces). Along with this, the compression reduces somewhat for complex scenes, as chances of non-uniformity increase, but still stays high enough for the approach to not lose viability. An observation here is that we should get high compression rates even with very big scenes if they have spread-out, small objects. Table 7.3 reaffirms our confidence in our choice of compression algorithm, showing steady results across all scenes and all resolutions.

If a user wishes to create various images at specific depth and aperture values using SLFT, from the results in tables 7.1 to 7.3, we find that an angular resolution $N = 128$ is ideally suited for a reasonable compute and time budget. However, our framework enables arbitrary scaling of the resolution as desired by the user, for both spatial and angular resolutions. For specific scenes, depending on visual complexity, further tweaking with compression parameters should also help in controlling the size of the video.

Angular Resolution	Scene 1: Bike (10K tris)		
	Time Taken	Avg. PSNR	Avg. SSIM
16	175s	38.396	0.9498
32	197s	42.515	0.9741
64	218s	46.684	0.9896
128	279s	50.566	0.9954
256	578s	53.415	0.9961
512	1598s	∞	1.0

Angular Resolution	Scene 2: Cabin (15K tris)		
	Time Taken	Avg. PSNR	Avg. SSIM
16	216s	38.404	0.9238
32	245s	43.112	0.9710
64	264s	46.834	0.9835
128	406s	50.219	0.9919
256	994s	52.180	0.9931
512	2789s	∞	1.0

Angular Resolution	Scene 3: Bedroom (207K tris)		
	Time Taken	Avg. PSNR	Avg. SSIM
16	331s	33.424	0.9140
32	379s	37.393	0.9571
64	476s	42.012	0.9806
128	742s	46.238	0.9919
256	1656s	48.521	0.9930
512	4230s	∞	1.0

Table 7.1 Comparison of PSNRs due to varying angular resolution in the light slab. The PSNR and SSIM values are computed by taking the highest angular resolution as reference (N=512). Time taken is the runtime of the complete algorithm 1. The high PSNR values suggest good quality reconstruction, increasing with angular resolution.

Angular Resolution (Expected Size)	Scene 1: Bike (10K tris)		
	Time Taken	Video Size (MB)	Compression Ratio
16 (768MB)	4s	1.59	483.1x
32 (3GB)	9s	5.18	592.8x
64 (12GB)	30s	17.08	719.5x
128 (48GB)	118s	64.01	767.9x
256 (192GB)	473s	235.04	836.5x
512 (768GB)	1965s	470.05	1673.1x

Angular Resolution (Expected Size)	Scene 2: Cabin (15K tris)		
	Time Taken	Video Size (MB)	Compression Ratio
16 (768MB)	6s	3.78	203.2x
32 (3GB)	11s	11.98	256.4x
64 (12GB)	32s	41.51	296.0x
128 (48GB)	120s	146.26	336.1x
256 (192GB)	472s	520.07	378.1x
512 (768GB)	2130s	1195.22	658.0x

Angular Resolution (Expected Size)	Scene 3: Bedroom (207K tris)		
	Time Taken	Video Size (MB)	Compression Ratio
16 (768MB)	13s	3.83	200.4x
32 (3GB)	19s	13.24	232.1x
64 (12GB)	44s	47.40	259.3x
128 (48GB)	143s	178.50	275.4x
256 (192GB)	547s	655.64	300.0x
512 (768GB)	2321s	1887.62	416.6x

Table 7.2 Compressed video sizes and times with varying angular resolution in the light slab. The table shows increasing video compression rates with increasing angular resolution. Time taken is the runtime of the video generation code.

Angular Resolution	Scene 1: Bike (10K tris)		
	Time Taken	Avg. PSNR	Avg. SSIM
16	12s	41.098	0.9853
32	30s	41.456	0.9864
64	102s	41.774	0.9871
128	383s	42.002	0.9876
256	1498	42.096	0.9877
512	5403s	42.655	0.9880

Angular Resolution	Scene 2: Cabin (15K tris)		
	Time Taken	Avg. PSNR	Avg. SSIM
16	17s	44.756	0.9961
32	39s	45.125	0.9967
64	121s	45.268	0.9970
128	490s	45.315	0.9971
256	1893s	45.370	0.9972
512	6793s	45.402	0.9973

Angular Resolution	Scene 3: Bedroom (207K tris)		
	Time Taken	Avg. PSNR	Avg. SSIM
16	15s	41.516	0.9959
32	34s	41.888	0.9966
64	111s	42.064	0.9970
128	415s	42.157	0.9972
256	1400s	42.241	0.9974
512	6109s	42.253	0.9975

Table 7.3 Comparison of PSNRs with varying angular resolution due to H.264 video compression. Time taken is the runtime of the video decoding and output plane generation code. The table shows that a constant quality reconstruction is achieved across all angular resolutions. As image quality of reconstructed results increases with increasing angular resolution, therefore video compression provides a viable alternative to generate images of any focus and aperture setting post-trace, with no significant loss in quality.

Chapter 8

Conclusions

Light fields are known to capture the radiance from rays in all directions, providing the angular information lost in conventional 2D photography. Light field imaging has emerged as a technology that allows to capture richer visual information from the world space, offering powerful capabilities for scene understanding, along with depth sensing and post-capture refocusing. They are slated to play a major role in VR, AR (Augmented Reality) and video stabilization, material classification and so on.The high-dimensionality of light fields, however pose challenging requirements for data capture, data compression etc.

This thesis provides an accurate and robust ray tracing framework SLFT that uses light slab implementation for generation of multi view sub-aperture images as in light field camera through ray tracing. One can use SLFT [26] for efficient digital refocusing in synthetic light fields. SLFT also effectively addresses the inherent challenging requirement of light field storage preserving the spatial and angular resolution within acceptable limits.

Our framework leverages on the quality and simplicity of the light slab representation. We provide equivalence with sub-aperture images formed by a standard light field camera for synthetic light fields. We demonstrate accurate post-tracing manipulations of focus and aperture settings from ray-traced synthetic light fields of 3D scenes.The analysis presented in this paper allows ample flexibility to the user to play around with basic parameters to generate optimal images for any desired application. Further, this framework is extensible for adding rendering of advanced photo realistic effects in the pipeline, along with the implementation of advanced compression codecs, and superior encoding and decoding schemes in the future.

8.1 Future Work

There is a lot of research going on to make ray tracing happen in real time. NVIDIA has recently released it's Turing series of GPUs, which provide dedicated ray tracing hardware, claim to boost ray tracing throughput by an order of magnitude. Creating a framework like ours custom-tuned to work with hardware-accelerated ray tracing can yield huge improvements to the run time of our light field

generation algorithm. More research will be required for that. With these recent developments and upcoming generations of GPU hardware, raytracing still presents much promise for exciting research in the years to come.

Also, work on better GPU encoders and decoders for next-generation video codecs like H.265/HEVC will boost the turnaround times of image retrieval, along with increasing the image quality. Further work can also be done on adding rendering support for advanced photorealistic effects in the pipeline.

VR provides an interesting use case for light fields, as changes in viewpoint inside the light field volume can be handled very quickly if the ray data is stored in a easily accessible, compressed format. Integration of light field generation schemes like ours, along with the image retrieval algorithms, into VR applications can be an interesting area to explore.

Related Publications

Udyan Khurana, Parikshit Sakurikar and P.J. Narayanan. SLFT: A physically accurate framework for Tracing Synthetic Light Fields. In Proceedings of the 11th Indian Conference on Computer Vision, Graphics and Image Processing (ICVGIP), December 2018.

Bibliography

[1] Depth of field illustrations. https://photographylife.com/what-is-depth-of-field.

[2] Lytro cameras. https://www.lytro.com.

[3] E. H. Adelson and J. R. Bergen. The plenoptic function and the elements of early vision. In *Computational Models of Visual Processing*, pages 3–20. MIT Press, 1991.

[4] E. H. Adelson and J. Y. A. Wang. Single lens stereo with a plenoptic camera. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (2):99–106, 1992.

[5] A. Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), pages 37–45, New York, NY, USA, 1968. ACM. doi: 10.1145/1468075.1468082. URL http://doi.acm.org/10.1145/1468075.1468082.

[6] S. A. Benton. Survey of holographic stereograms. volume 0367, 1983. doi: 10.1117/12.934296. URL https://doi.org/10.1117/12.934296.

[7] J. F. Blinn. Models of light reflection for computer synthesized pictures. *SIGGRAPH Comput. Graph.*, 11(2):192–198, July 1977. ISSN 0097-8930. doi: 10.1145/965141.563893. URL http://doi.acm.org/10.1145/965141.563893.

[8] R. C. Bolles, H. H. Baker, and D. H. Marimont. Epipolar-plane image analysis: An approach to determining structure from motion. *IJCV*, 1(1):7–55, Mar 1987. doi: 10.1007/BF00128525. URL https://doi.org/10.1007/BF00128525.

[9] D. G. Dansereau, O. Pizarro, and S. B. Williams. Decoding, calibration and rectification for lenselet-based plenoptic cameras. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1027–1034, 2013.

[10] M. Faraday. Thoughts on ray vibrations. *Philosophical Magazine*, 28(188), 1846.

[11] T. Feng and H.-Y. Shum. An optical analysis of light field rendering. pages 23–25, 02 2002.

[12] T. Georgiev, G. Chunev, and A. Lumsdaine. Superresolution with the focused plenoptic camera. volume 7873, pages 232 – 244. International Society for Optics and Photonics, SPIE, 2011. doi: 10.1117/12.872666. URL https://doi.org/10.1117/12.872666.

[13] A. Gershun. The light field. *J. Math. and Physics*, (18):51–151, 1936.

[14] A. S. Glassner, editor. *An Introduction to Ray Tracing*. Academic Press Ltd., London, UK, UK, 1989. ISBN 0-12-286160-4.

[15] A. S. Glassner. *Principles of Digital Image Synthesis*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994. ISBN 1558602763.

[16] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 43–54, New York, NY, USA, 1996. ACM. ISBN 0-89791-746-4. doi: 10.1145/237170.237200. URL http://doi.acm.org/10.1145/237170.237200.

[17] F. Hawary, C. Guillemot, D. Thoreau, and G. Boisson. Scalable light field compression scheme using sparse reconstruction and restoration. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3250–3254, Sep. 2017. doi: 10.1109/ICIP.2017.8296883.

[18] P. S. Heckbert and P. Hanrahan. Beam tracing polygonal objects. *SIGGRAPH Comput. Graph.*, 18 (3):119–127, Jan. 1984. ISSN 0097-8930. doi: 10.1145/964965.808588. URL http://doi.acm.org/10.1145/964965.808588.

[19] Q. Hou, X. Sun, K. Zhou, C. Lauterbach, and D. Manocha. Memory-scalable gpu spatial hierarchy construction. *IEEE Transactions on Visualization and Computer Graphics*, 17(4):466–474, April 2011. doi: 10.1109/TVCG.2010.88.

[20] A. Isaksen, L. McMillan, and S. J. Gortler. Dynamically reparameterized light fields. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 297–306, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. ISBN 1-58113-208-5. doi: 10.1145/344779.344929. URL http://dx.doi.org/10.1145/344779.344929.

[21] H. E. Ives. Parallax panoramagrams made with a large diameter lens. *J. Opt. Soc. Amer.*, (20): 332–342, 1930.

[22] T. Ize, P. Shirley, and S. Parker. Grid creation strategies for efficient ray tracing. In *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, RT '07, pages 27–32, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 978-1-4244-1629-5. doi: 10.1109/RT.2007.4342587. URL https://doi.org/10.1109/RT.2007.4342587.

[23] B. Javidi and F. Okano. *Three-dimensional television, video, and display technologies*. Springer Science & Business Media, 2002.

[24] X. Jiang, M. Le Pendu, and C. Guillemot. Light field compression using depth image based view synthesis. In *2017 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, pages 19–24, July 2017. doi: 10.1109/ICMEW.2017.8026313.

[25] T. L. Kay and J. T. Kajiya. Ray tracing complex scenes. *SIGGRAPH Comput. Graph.*, 20(4): 269–278, Aug. 1986. ISSN 0097-8930. doi: 10.1145/15886.15916. URL `http://doi.acm.org/10.1145/15886.15916`.

[26] U. Khurana, P. Sakurikar, and P. J. Narayanan. SLFT: A physically accurate framework for Tracing Synthetic Light Fields. In *11th Indian Conference on Computer Vision, Graphics and Image Processing (ICVGIP)*, 2018. doi: 10.1145/3293353.3293417.

[27] C. Kolb, D. Mitchell, and P. Hanrahan. A realistic camera model for computer graphics. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 317–324, New York, NY, USA, 1995. ACM. ISBN 0-89791-701-4. doi: 10.1145/218380.218463. URL `http://doi.acm.org/10.1145/218380.218463`.

[28] P. T. Kovcs, Z. Nagy, A. Barsi, V. K. Adhikarla, and R. Bregovi. Overview of the applicability of h.264/mvc for real-time light-field applications. In *2014 3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON)*, pages 1–4, July 2014. doi: 10.1109/3DTV.2014.6874744.

[29] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha. Fast BVH Construction on GPUs. *Computer Graphics Forum*, 2009. ISSN 1467-8659. doi: 10.1111/j.1467-8659.2009.01377.x.

[30] M. Levoy. Light fields and computational imaging. *Computer*, 39(8):46–55, Aug. 2006. ISSN 0018-9162. doi: 10.1109/MC.2006.270. URL `https://doi.org/10.1109/MC.2006.270`.

[31] M. Levoy and P. Hanrahan. Light field rendering. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 31–42, New York, NY, USA, 1996. ACM. ISBN 0-89791-746-4. doi: 10.1145/237170.237199. URL `http://doi.acm.org/10.1145/237170.237199`.

[32] Z. Lin and H.-Y. Shum. A geometric analysis of light field rendering. *Int. J. Comput. Vision*, 58(2):121–138, July 2004. ISSN 0920-5691. doi: 10.1023/B:VISI.0000015916.91741.27. URL `https://doi.org/10.1023/B:VISI.0000015916.91741.27`.

[33] G. Lippmann. La photographie intégrale. *Comptes-Rendus, Académie des Sciences*, (146):446–551, 1908.

[34] D. J. MacDonald and K. S. Booth. Heuristics for ray tracing using space subdivision. *Vis. Comput.*, 6(3):153–166, May 1990. ISSN 0178-2789. doi: 10.1007/BF01911006. URL `http://dx.doi.org/10.1007/BF01911006`.

[35] M. Magnor and B. Girod. Data compression for light-field rendering. *IEEE Trans. Cir. and Sys. for Video Technol.*, 10(3):338–343, Apr. 2000. ISSN 1051-8215. doi: 10.1109/76.836278. URL `http://dx.doi.org/10.1109/76.836278`.

[36] P. Moon and D. E. Spencer. The photic field. *Cambridge, MA, MIT Press, 1981. 265 p.*, 1981.

[37] O. Nemi, S. Rimac-Drlje, and M. Vranje. Multiview video coding extension of the h.264/avc standard. In *Proceedings ELMAR-2010*, pages 73–76, Sep. 2010.

[38] I. Newton. *Opticks or, A Treatise of the Reflexions, Refractions, Inflexions and Colours of Light*. 1730.

[39] R. Ng. *Digital light field photography*. Stanford University, 2006.

[40] R. Ng, M. Levoy, M. Brédif, G. Duval, M. Horowitz, P. Hanrahan, et al. Light field photography with a hand-held plenoptic camera. *Computer Science Technical Report CSTR*, 2(11):1–11, 2005.

[41] T. Okoshi. *Three-dimensional imaging techniques*. Elsevier, 2012.

[42] M. Potmesil and I. Chakravarty. Synthetic image generation with a lens and aperture camera model. *ACM Trans. Graph.*, 1(2):85–108, Apr. 1982. ISSN 0730-0301. doi: 10.1145/357299.357300. URL `http://doi.acm.org/10.1145/357299.357300`.

[43] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan. Ray tracing on programmable graphics hardware. *ACM Trans. Graph.*, 21(3):703–712, July 2002. ISSN 0730-0301. doi: 10.1145/566654.566640. URL `http://doi.acm.org/10.1145/566654.566640`.

[44] M. Ren, R. Liu, H. Hong, J. Ren, and G. Xiao. Fast object detection in light field imaging by integrating deep learning with defocusing. *Applied Sciences (Switzerland)*, 7, 12 2017. doi: 10.3390/app7121309.

[45] M.-G. Retzlaff, J. Hanika, J. Beyerer, and C. Dachsbacher. Physically based computer graphics for realistic image formation to simulate optical measurement systems. *Journal of Sensors and Sensor Systems*, 6(1):171–184, 2017. doi: 10.5194/jsss-6-171-2017.

[46] M. Rizkallah, T. Maugey, C. Yaacoub, and C. Guillemot. Impact of light field compression on focus stack and extended focus images. In *2016 24th European Signal Processing Conference (EUSIPCO)*, pages 898–902, Aug 2016. doi: 10.1109/EUSIPCO.2016.7760378.

[47] P. Sakurikar, I. Mehta, V. N. Balasubramanian, and P. J. Narayanan. Refocusgan: Scene refocusing using a single image. In *Computer Vision – ECCV 2018*, pages 519–535, Cham, 2018. Springer International Publishing. ISBN 978-3-030-01225-0.

[48] P. Sakurikar, I. Mehta, and P. J. Narayanan. Defocus magnification using conditional adversarial networks. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1337–1346, Jan 2019. doi: 10.1109/WACV.2019.00147.

[49] M. Stich, H. Friedrich, and A. Dietrich. Spatial splits in bounding volume hierarchies. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, pages 7–13, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-603-8. doi: 10.1145/1572769.1572771. URL `http://doi.acm.org/10.1145/1572769.1572771`.

[50] V. Van Duong, T. Nguyen Canh, and B. Jeon. Light field image coding for efficient refocusing. In *2018 IEEE 12th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pages 74–78, Sep. 2018. doi: 10.1109/MCSoC2018.2018.00024.

[51] I. Wald. On fast construction of sah-based bounding volume hierarchies. In *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, RT '07, pages 33–40, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 978-1-4244-1629-5. doi: 10.1109/RT.2007.4342588. URL `https://doi.org/10.1109/RT.2007.4342588`.

[52] T. Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, June 1980. ISSN 0001-0782. doi: 10.1145/358876.358882. URL `http://doi.acm.org/10.1145/358876.358882`.

[53] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the h.264/avc video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, July 2003. doi: 10.1109/TCSVT.2003.815165.

[54] B. Wilburn, N. Joshi, V. Vaish, E.-V. Talvala, E. Antunez, A. Barth, A. Adams, M. Horowitz, and M. Levoy. High performance imaging using large camera arrays. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 765–776. ACM, 2005.

[55] K. Zhou, Q. Hou, R. Wang, and B. Guo. Real-time kd-tree construction on graphics hardware. *ACM Trans. Graph.*, 27(5):126:1–126:11, Dec. 2008. ISSN 0730-0301. doi: 10.1145/1409060.1409079. URL `http://doi.acm.org/10.1145/1409060.1409079`.