Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit Nr. 69

Simulation and Rendering of Light Field Data in the Mitsuba Renderer

Florian Schinkel

Course of Study: Informatik

Examiner: Jun.-Prof. Dr.-Ing. Martin Fuchs

Supervisor: Dipl.-Ing. Alexander Wender

Commenced: 2015-09-28

Completed: 2016-05-06

CR-Classification: I.3.0, I.3.3, I.3.4, I.3.7, I.4.1

Abstract

Light fields offer the opportunity to change properties of pictures, even after they were shot. There is no possibility of recording all four necessary dimensions of a light field with a 4D camera, but many different techniques for generating light fields from normal 2D pictures exist. Comparing the quality and usability of these techniques is however difficult, because they often only rely on prototypes or are complex and costly to reproduce. Virtual simulations of theses techniques offer an easy and fast method for such comparisons. The rendering of light fields can be done with normal render software, but which render method is best suited for it, is unclear.

This thesis wants to answer the question if the Mitsuba renderer is capable of simulating the acquisition of light fields and rendering light field data. In several simulations, Mitsuba will be used to create light fields from multiple test scenes. In addition, virtual light field data will be rendered with Mitsuba in different ways. Each time it will be evaluated, which of Mitsubas rendering methods are well suited for the task and which are not. In each case the required time and image quality of the results will be assessed. Because Mitsuba does not support light fields by default, a custom software framework was implemented for simulations and tests. The framework allows to use Mitsuba for light field acquisition and adds the functionality to render light field data to Mitsuba.

The thesis will show that Mitsubas methods of rendering are absolutely capable of creating synthetic light fields or rendering them. Depending on the use case however, the choice for the right rendering technique differs. For example, some methods are not capable of creating light fields of specific scenes, while other methods can not be used for rendering.

Kurzfassung

Lichtfelder ermöglichen es Eigenschaften von Bildern auch nach der Aufnahme zu ändern. Es gibt zwar keine Möglichkeit die vier notwendigen Dimensionen eines Lichtfeldes direkt mit einer 4D Kamera aufzunehmen, aber es gibt zahlreiche und sehr unterschiedliche Techniken um Lichtfelder aus normalen 2D Bildern zu generieren. Diese Aufnahmetechniken miteinander auf Qualität oder Nutzbarkeit zu vergleichen, gestaltet sich allerdings schwierig, da sie größtenteils nur auf Prototypen basieren oder sehr aufwendig und teuer zu reproduzieren sind. Virtuelle Simulation solcher Techniken bietet eine einfache und schnelle Möglichkeit für solche Vergleiche. Das Rendern von Lichtfeldern kann durch normale Rendersoftware erfolgen, aber hier stellt sich die Frage, welche Rendermethoden am besten dafür geeignet sind.

Diese Arbeit soll klären, ob der Mitsuba Renderer dazu genutzt werden kann, um Lichtfeldaufnahmen zu simulieren und Lichtfelddaten zu rendern. Dabei wird Mitsuba in mehreren Simulationen dazu benutzt werden, um Lichtfelder von verschiedenen Testszenen zu erzeugen. Außerdem werden anschließend virtuell erzeugte Lichtfelder mit Hilfe von Mitsuba auf mehrere Arten gerendert werden. Dabei wird jeweils evaluiert, welche Rendermethoden von Mitsuba sich gut oder gar nicht für die Aufnahme oder das Rendern eignen. Dabei wird sowohl die Laufzeit als auch die erzeugte Bildqualität bewertet werden. Da Mitsuba den Umgang mit Lichtfeldern nicht standardmäßig unterstützt, wurde speziell ein Software Framework erstellt, um die Simulationen und Tests durchzuführen. Das Framework ermöglicht die Lichtfeldaufnahme mit Mitsuba und erweitert Mitsuba um die Funktion Lichtfelddaten zu rendern.

Es wird sich zeigen, dass sich die Rendermethoden von Mitsuba im allgemeinen durchaus dafür eignen Lichtfelder synthetisch aufzunehmen oder zu rendern. Welche Rendermethode dabei allerdings zu bevorzugen ist, unterscheidet sich von Anwendungsfall zu Anwendungsfall. Einige Methoden sind zum Beispiel nicht dazu geeignet Lichtfelder von bestimmten Szenen aufzunehmen, während andere nicht zum Rendern genutzt werden können.

Danksagung

Ich möchte zuerst Christin dafür danken, dass sie immer an meiner Seite war, mich immer unterstützt hat und immer an mich geglaubt hat. Ohne sie hätte ich diese Arbeit nicht geschafft.

Ein sehr großes Dankeschön gebührt auch meinen Eltern, dafür, dass sie mich all die Jahre ertragen und mir damit mein Studium erst ermöglicht haben.

Zuletzt gehört natürlich auch Martin Fuchs und Alexander Wender gedankt, da beide während der gesamten sechs Monate Bearbeitungszeit stets hilfsbereit waren und mir mit Rat und Tat zur Seite standen.

Contents

List of Figures

List of Tables

1 Introduction

Today digital photography is the most common way to take photographs of the real world. Digital 2D sensors take light, which shines trough a cameras lens, accumulate it and convert it to a digital number value for every sensor pixel. The resulting photograph is a representation of an array of those captured pixel-values. It is clear, that this representation is only a rough estimate of the complete light transport that happens outside of the camera. The so called plenoptic function describes, that the complete light transport real world scenes can be described as a 7D function (more details of the plenoptic version will follow in chapter **??**)[AB91]. Because normal cameras project the 7D light transport onto a 2D plane, much information about the world surrounding the camera gets lost.

The approach of light fields, introduced by [GGSC96] and [LH96], tries to capture more of the available light information than normal pictures. Light fields reduce the 7D plenoptic function to a 4D function by defining certain assumptions. Unfortunately, the direct acquisition of light fields is also impossible, because only 2D image sensors exist. But it is possible to calculate 4D light fields from 2D image data. There are many different ways of doing that and they all have different properties and differentiate in complexity and cost. Often, normal 2D digital cameras in combination with additional, custom-made lens arrays, filters or other utilities get used for the capturing. All acquisition techniques than use some kind of multiplexing to form a light field from the available 2D pixel values. While light fields do not capture the whole seven dimensions of the plenoptic function, they offer more possibilities for post-processing than normal 2D pictures do. For example they allow for the viewpoint or focus of a picture to be changed, after it was taken. More on light fields, their acquisition and uses will be presented in chapter 2.

Of course the 4D light field data has to be recalculated into new 2D images to view them. In general creating 2D images from abstract data structures with software is called rendering. Common rendering techniques can also be used to render new 2D images from 4D light field data. More on the basics of rendering and how it can be used to render light fields can be found in chapter 3.

1.1 Motivation

The possibilities for capturing light fields are manifold and different, but a comparison between them is often not feasible or even not possible, because they often rely on very complex or cost intensive custom made prototypes and therefore cant be recreated easily. An way to make comparisons between different acquisition techniques possible is synthetic simulation. Here the light fields are not captured in the real world scenarios with camera hardware and additional utilities, but instead with the aid of rendering: Virtual cameras are used to capture light field data from virtual 3D scenes. With this approach no real world prototypes or utilities have to be bought or built and the different techniques can be compared easily and fast under different conditions. To compare the results of the simulations, the resulting light fields of course have to be rendered.

So to conduct simulations of light field acquisition, a rendering software that is capable of producing light field data and rendering such data is needed (ideally a single application that can handle both tasks). Such renderers are very complex and there are many different ways to implement them. So to programming a custom renderer be time consuming and expensive.

Luckily there is a render software available, that can be tuned to work with light fields: Mitsuba. Mitsuba offers a lot of different and easy to use state-of-the-art rendering techniques, is open source, well documented and can easily be customized by third-party programmers. Because of this properties, Mitsuba seems like a suitable renderer for simulating different light field acquisition techniques, as well for rendering light fields.

1.2 Objectives

The main goal of this thesis is to examine how well Mitsuba to generate and render light fields. In a first step a synthetic simulation for light field acquisition will be created to compare two different acquisition techniques. Mitsuba will be used for this simulation scenarios. Its different render techniques will then be compared and evaluated on how well they are suited for such simulations.

In a second steps Mitsuba will then be used to render light field data and its different techniques will again be compared. For the rendering part it was chosen to only use synthetically created light field data, because no suitable real world light field data was available to the author. But the chosen rendering algorithm should also be applicable to real world data.

Because Mitsuba does not support light fields out of the box, its customizability and expandability have to be exploited to use it for light field simulation and rendering. For that, a custom software solution, that uses Mistuba will be created.

1.3 Outline

In chapter 2 the theoretical basis and practical implementations of light fields are introduced. This also includes what advantages they offer over normal photography, their applications, as well as their benefits and shortcomings. The next chapter, chapter 3, is about the basics of rendering and rendering light fields. Mitsuba and all its methods of rendering will get introduced there as well. After that, the custom software framework which was created to make Mitsuba capable of simulating light field acquisition and rendering, will get introduced in chapter 4, along with all its capabilities, conditions and limitations. Chapter 5 covers the conducted simulations for light field acquisition. The structures and conditions of them, as well as their results will be presented there. The tests, that where conducted to examine the capabilities of Mitsuba to render light fields will be presented in chapter 6. A summary and outlook on future work in chapter 7 and a conclusion in chapter 8 will finally conclude this thesis.

2 Light Field Basics

The topic of this chapter are the fundamentals of light fields. First the plenoptic function, on which light fields are based on, will be introduced. Subsequently it will be defined what light fields are, followed by an overview over possible acquisition techniques. Finally their use cases and their problems will be discussed.

2.1 The Plenoptic Function

The plenoptic function was first described by Adelson and Bergen in 1991 [AB91]. This function completely describes the distribution of light in space. It is based on the assumption, that light can be interpreted as an finite number of individual light rays, that travel trough space. Every ray has a distinct direction and intensity. The plenoptic function can be written as:

(2.1) $P(\theta, \phi, \lambda, t, V_x, V_y, V_z)$, where

- θ and ϕ are the angles for a ray, with respect to an optic axes.
- λ is the wavelength, which in camera systems translates to the color of the ray.
- t is time.
- V_x, V_y and V_z are x,y and z coordinates in space.

It contains seven independent dimensions: For every position in space (V_x, V_y, V_z) it defines all rays, coming from every angle (θ, ϕ), with every possible wavelength (λ), at any point in time. If now a camera is placed at position V_{x1}, V_{y1}, V_{z1}, the plenoptic function can be evaluated for that position and all rays, that reach the camera can get used to form that cameras image. The taken picture then is representative of all light rays, that were arriving at the cameras position on that explicit point in time the picture was taken. If the complete plenoptic function for a scene is determined and saved, a viewer could later traverse trough a virtual representation of that scene freely, without needing information about the scenes geometry. With this 7D representation, not only

the camera position, but also the time or other properties of the camera(for example field of view) could be changed. But unfortunately it is impossible to record complete 7D plenoptic functions for scenes. Even if it was possible, the recorded data would be too big to save and use efficiently.

Normal photography just captures an 2D representation of the complete 7D plenoptic function, with the two parameters being the coordinates on the image sensor (for digital photography pixels represent this coordinates). Each pixel gets assigned a value, that is representative of the intensity of all the light rays, that were hitting it over the exposure time of the picture. Information about the direction of the rays or their wavelength gets lost. For this reason, properties of the image can not be changed after it was shot.

2.2 Light Fields

Light fields are a 4D slice of the complete 7D plenoptic function of a scene. They do not offer all capabilities of the plenoptic function, but their acquisition and usage is realizable for real world scenarios. Because they record more information about a scenes light transport, they offer many possibilities to change output pictures, even after they were taken (see chapter 2.4 for more use cases).

But what exactly are light fields?

Lightfields were introduced in two papers, "The Lumigraph"([GGSC96]) and "Light Field Rendering" ([LH96]), both released separately in 1996. As mentioned, they are a 4D slice of the complete 7D plenoptic function.

Three assumptions get made to reduce the seven dimensions and thereby making the acquisition and usage of light fields possible: The first assumption is, that only static scenes with constant illumination are captured. Therefor the time-dimension can be ignored. The wavelength can get ignored as well, if color only gets represented as three discrete color channels. With theses assumptions, the seven dimension were reduced to five dimensions: The position (V_{x1}, V_{y1}, V_{z1}) and the direction (θ, ϕ) of every single ray. An additional assumption reduces that five dimension to four: Scenes are free of occluders and shot in a transparent medium, like air or vacuum. Under that assumption the intensity of rays does not change in space, so rays have the same intensity, no matter where in space they are sampled. This way one dimension from the position can be omitted and a ray can be fully described with four dimensions.

To assure efficient calculations, control over the captured area of rays and uniform sampling, Levoy et al. proposed to parametrise the four dimensions as "light slabs" [LH96]. An example for the light slab representation can be seen in Fig.2.1.

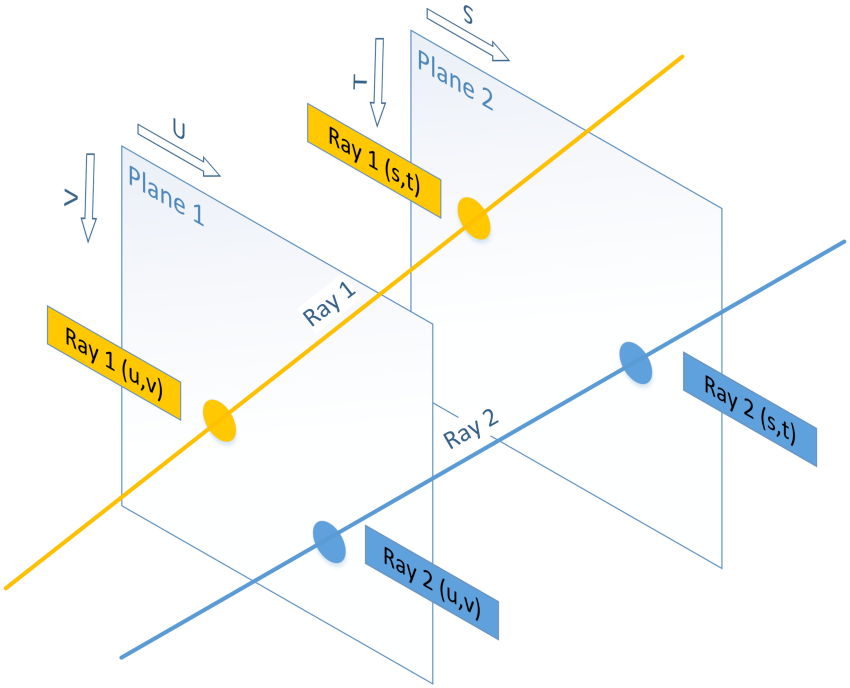


Figure 2.1: Light slab representation of two rays. The respective coordinates of the intersection points with Plane 1 and Plane 2 get used to parametrise both rays.

Two parallel planes get defined in space. All light rays of the scene intersect those planes at two points. A Ray then can be represented by the u,v,s,t-coordinates of the two intersection points. This way every rays origin and direction can be identified and saved individually. This u,v,s,t-parametrization still gets used today. The light field of a scene contains all possible (or as many as possible) light rays in this 4D representation.

2.3 Acquisition of Light Fields

For scenes that meet the mentioned assumptions, a 4D light field can be recorded. But because cameras are only capable of capturing 2D images, ways for creating light fields with 2D cameras have to be used. The main issue is, that light fields need information about every single light ray in a scene and normal 2D cameras are not capable of recording single rays. To make single ray acquisition possible, either the process of recording the image, or the camera itself has to be altered, so that the rays can be recalculated from the taken picture. There are many different techniques for doing that. They all use some kind of multiplexing or redundant image information to calculate the

light slabs of the light field. They can be categorized into four different categories (after [WIL+12]):

2.3.1 Multi Device Capturing

Multiple cameras (with identical properties) get combined to an array [WJV+05], [YEBM02]. An example for such an array can be seen in Fig.2.2. All cameras in the

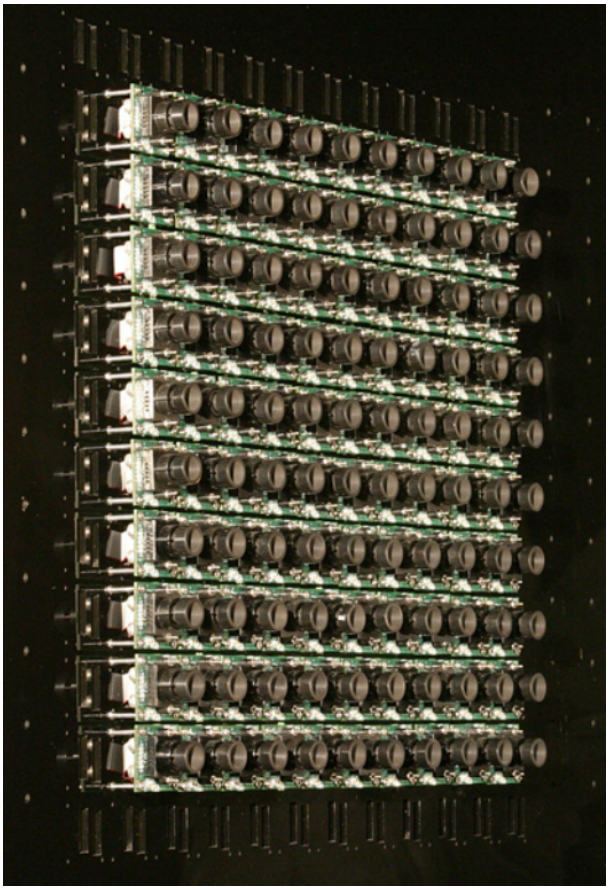


Figure 2.2: An example for a camera array with 100 video cameras.
(Image source: [WJV+05])

array take a picture simultaneously and all pictures combined form the light field. The separate cameras record the scene from multiple different positions and cover different parts of the scene. In such setups, the intrinsic and extrinsic properties of the cameras are known, so calculating light slab coordinates for every image pixel is easy. The first

plane (u,v-plane) is where the camera array is located[1] and the second plane is located in the scene.

The problem with this approach is the space and cost factor: For sufficient light field density, the number of cameras and their resolution has to be high.

2.3.2 Time Sequential Imaging

Time sequential imaging only needs one camera and is therefore more cost and space efficient than multi device approaches. To generate a light field, a single camera is used to record multiple shots of a static scene, each from a different position. In most approaches this is done by traversing the camera around the scene[LH96],[DHT+00]. The traversion can either be done manually, or with the help of mechanical devices. Because the extrinsic parameters of the camera has to be known for the computation of light slabs, the positions of the camera has to be tracked [2]. This is done by placing trackers in the scene. A software can easily recognize the trackers in the scene and use their recorded positions to calculate the exact camera position, from which the individual pictures were taken [3]. An example for a scene containing trackers can be seen in Fig.2.3.

There is also an approach, that does not move the camera around, but still captures scenes from different angles: Ihrke et al. proposed tacking s static camera to capture a scene which contains a trackable mirror [ISG+08]. The mirror is then moved to different positions in the scene and photographed to acquire pictures from different view angles, which of course can form a light field.

2.3.3 Integral Imaging

Multi device capturing, as well as time sequential imaging can be quite expensive and time consuming. Integral imaging introduces the possibility to record light fields with only a single camera and a single exposure. For that, the camera itself gets altered by changing the internal properties of the sensor or the lens. Usually this get accomplished

[1] The positions of the individual cameras within the camera array is known.

[2] When using time sequential light field acquisition in virtual simulations this step can be skipped, because the cameras positions are known.

[3] The tracking and calculating of camera positions is a part of computer vision and not subject of this thesis. Recognizing certain patterns in pictures is called "pattern recognition". Once patterns are found, computer vision techniques can find correspondences in the pictures to calculate 3D camera positions. For mor details on computer vision, please refer to specialized literature, like [CPW10] or [Sze10]

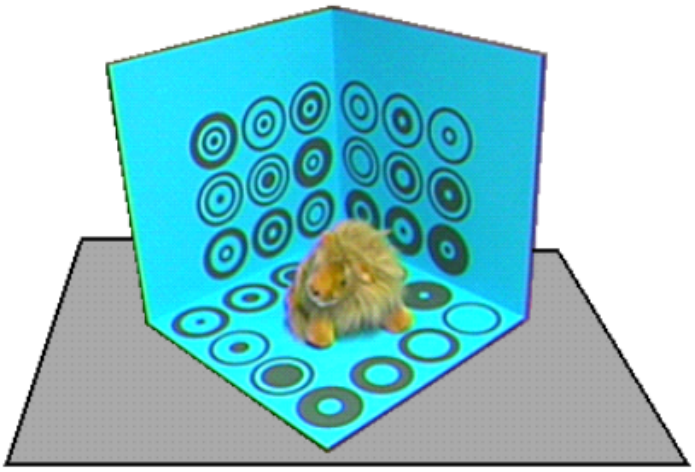


Figure 2.3: An example scene for light field acquisition with time sequential imaging. The scenes contains a lion puppet, which is placed in a sky blue box. The box is marked with multiple round markers.
(Image source [GGSC96])

by putting additional optical elements in front of the sensor. This way incoming light rays get subdivided and a single sensor can capture multiple different sub pictures. Often an array of micro lenses is placed directly in front of the image sensor to achieve that. There are multiple appraoches for that method, see [AW92], [BF12], [GL10], [NLB+05]. They differ in where the lenses are positioned and what kind of lenses are used. But they all share the same idea: In an ideal case, the light that enters the camera gets divided into separate rays, which each only interacts with on single sensor pixel. An example for that can be seen in Fig.2.4.

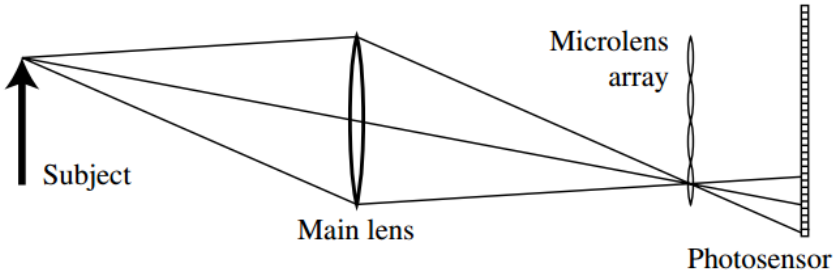


Figure 2.4: Schematic for an camera with a lens array in front of the sensor.
(Image source [NLB+05]).

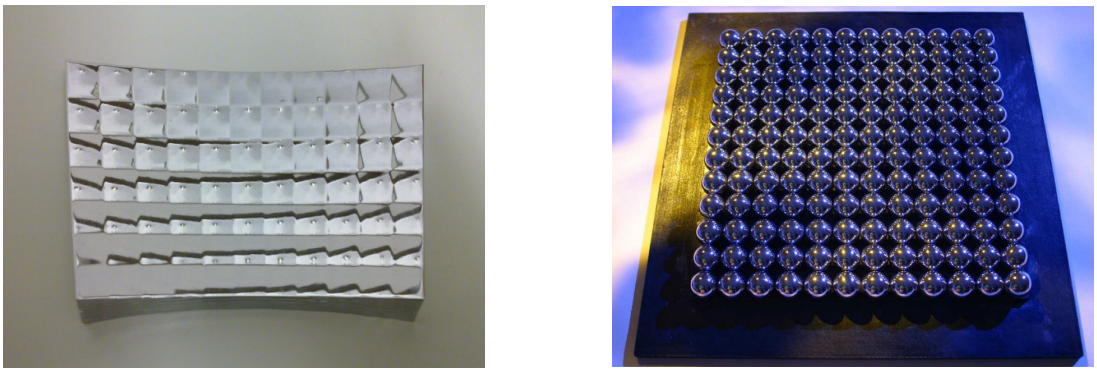
The whole sensor is divided into multiple sub pictures, with each representing different parts of the light field. An algorithm can later rearrange the pixels of the single picture to form the separate sub images.

The main problem with integral imaging is that image resolution of the resulting light fields is small. There is always a tradeoff between spatial resolution and angular resolution [GZC+06]. If x different camera positions are desired, the image sensor of the camera (with resolution y) has to be split in x sub pictures and each light field picture only can have a maximal resolution of $\frac{y}{x}$.

2.3.4 Single Sensor Plenoptic Multiplexing

This class of approaches also uses one single picture of a scene to create a light field. But unlike integral imaging, no additional elements are inserted into the camera. Light field are recorded by placing certain objects in front of the camera. So the scene is not captured directly, but indirectly. The sensor of the camera is also divided into sub pictures, but no alterations to the camera have to be performed. For example masks or filters can be placed in front of the camera [AVR10], [GL10].

A special method of single sensor plenoptic multiplexing is the usage of mirrors to simulate different viewpoints. The scene is photographed indirectly trough a mirror ball ([TARV10]) or mirror array ([FKR12]). The array can either be formed from mirror spheres ([UWH+03]), flat mirrors or slightly curved ones. Examples for mirror arrays can be seen in Fig.2.5.



(a) **(b)**

Figure 2.5: Two examples for mirror arrays:
a) Shows a mirror array made from slightly curved mirror facets.
(Image source [FKR12])
b) This array is constructed from small mirror balls.
(Image source [UWH+03])

The individual mirror pieces show the mirrored scene from different angles. The spatial and angular resolution are dependent on the number of mirror facets, their size and the angle in which they are aligned to each other.

2.3.5 Comparability

All introduced acquisition techniques are very different and have different strengths and weaknesses. But which technique is the best? Unfortunally, the comparison between them is difficult. Reasons for that are (after [LLC07]):

Most approaches do not use of the shelf hardware to capture light fields. Often special prototypes are constructed. Many of the papers int eh fields do not provide enough information about how their prototypes were constructed. Therefor an exact recreation is impossible. For example, detailed information about the used cameras is often omitted.

Even if the prototypes are explained in enough detail, building them can be quite expensive (for example multiple cameras in a array in [WJV+05]). As a result, building multiple prototypes and comparing them is often not feasible.

For that reasons (among others) the real world comparison of different techniques is not possible. A fast, easy and cheap alternative is to simulate the acquisition techniques virtually. The real world conditions get recreated and light fields are recorded in an virtual environment.

2.4 The Many Uses of Light Fields

As stated light fields capture more information about the total light transport of a scene than normal 2D pictures. This additional information can be used to allow for multiple post-processing possibilities. They can be used in many different fields, like for example image based rendering, integrated photography and computer vision.

Some of the most common use cases of light fields are:

- Changing the properties of the a scenes camera (for example focus or aptarture) [NLB+05],[VRA+07], [YEBM02], [WJV+05].
- They can be used for 3D reconstruction [AW92],[DHT+00]. Objects, that were captured in a light field can later be rendered as 3D models.
- Image based rendering ([SCK08], [LH96], [BBM+01]): Light fields can be used when rendering 3D scenes. All global illumination effects of a scene can be captured in real world light fields, without requiring any information about the scenes geometry. This light fields can later be used for rendering a virtual representation of the captured scene. For complex scenes, the usage of light fields can be quicker than traditional render techniques. They require less computing power and allow

for easy to render interactions with the scene (change of viewpoint). Because real world pictures can be used, the scene can be rendered photo realistic. In that context, light fields also can be used to capture and represent complex light sources [GGHS03], [HKSS98].

- Image artifacts like scattering, glare or blur can be reduced with the help of light fields[TAHL07], [VRA+07].

2.5 Problems with Light Fields

For all opportunities, that light fields offer, they also come with some shortcomings, which prevent them from being used outside of scientific research[4]:

2.5.1 Cost

Especially for multi device capturing the cost of buying multiple cameras is high. But micro lens arrays, used for integral imaging or mirror arrays can also be expensive, because they have to be custom made.

2.5.2 Data Sizes

Especially if time sequential or multi device capturing methods are used, the data size of the resulting light fields can be multiple gigabytes or terabytes in data size, because a large number of high resolution images is needed to build a dense light field. The storage, transfer and rendering of this data sets can be difficult without proper compression. Normal compression methods are not capable of compressing light fields to small size and within small time frames. Special compression methods that exploit the high redundancy within light field data have to be used. For example the approach of block wise bitpacking can be used to reduce light fields size dramatically [Sie15]. But even compressed the data size of a dense light field is often too big to store in computer memory(RAM) at once, which makes the usage not feasible.

[4]They are real world light field cameras available for purchase. Examples are the Lytro (https://www.lytro.com/) or the Raytrix (https://www.raytrix.de/). But no light field camera has reached wide spread popularity or usage.

2.5.3 Resolution

As stated often a tradeoff between spatial resolution and angular resolution has to be found for multiplexing techniques, that only require one image [GZC+06]. Even for image sensors with a high resolution (mega pixels), the resulting light field images only can reach a fraction. Rendering such low res light fields afterwards leads to poor picture quality and artifacts.

3 Rendering Basics

In this chapter the basic concepts of rendering 2D images in Computer Graphics will be introduced. Rendering is a broad and active field of research in Computer Science and Computer Graphics, so within the limits of this thesis it is impossible to talk about all its aspects in detail. The focus of this chapter will mainly lay on the most important and most basic topics. They are needed to understand the work that is later introduced in chapter 5 and chapter 6 in regards of simulating the acquisition and rendering of light fields.

At the beginning of this chapter the foundation of all of rendering in Computer Graphics will be introduced: The rendering equation.

3.1 Computer Graphics and the Rendering Equation

In Computer Graphics the main task, which has to be solved, is the creation of a 2D image from a description of a 3D scene. Most times the goal is to create realistic looking pictures, which later can not be distinguished from a real world photograph. Sometimes the creation of artistic looking images is desired (for example in cartoon animation movies) but in this work the focus will lay on creating realistic looking images. The way from a 3D representation of a scene to a 2D image is not trivial or simple, that is why many different approaches to solve this task are possible. But all approaches have something in common:

Interactions between the scenes objects, the light sources and the camera have to be calculated (or approximated) to simulate real world lightning conditions. Important topics are occlusion (which objects are visible from the camera?), shadowing (which parts of the scene are directly or indirectly lit?), perspective properties of the camera and many more. The simplest way to achieve this goal is to track the light which travels trough a scene, interacts with the different objects and finds its way into the camera.

There are two major ways to do this: Object order rendering and image order rendering.

Object order rendering uses the scenes objects to determine the pixel value of the rendered picture. Roughly speaking it iterates over the scenes objects, determines which object is visible at each pixel and than assigns the color value of the seen object to that pixel. It can be very fast, but only provides a rough estimation of real world illumination. Therefor it is not well suited for creating realistic looking images. Object order techniques is often used in frame rate dependent applications like video games or interactive 3D simulations. In those use cases the frame rate is more important than realistic simulation of lightning. An example for object order rendering is the scanline algorithm [WREE67].

On the contrary image order rendering is very slow, but capable of creating photo realistic images. It tries to emulate real world illumination physically correct. In general, image order rendering iterates over every image pixel to calculate their color values. An popular example of image based rendering is Ray Tracing. There are many different types of Ray Tracing, but they all come from the same idea: Shot light rays from the camera into a 3D scene, trace the way of the rays, intersect them with the objects of the scene and calculate the amount of light that gets reflected from that point into the camera. A more detailed explanation about Ray Tracing and a introduction in different Ray Tracing types will follow in chapter 3.5.

The main focus of this thesis will be on image order rendering, because to simulate real world light fields, realistic looking renderings are needed , which are hard to achieve for object order rendering.

The most fundamental concept of light transport in a 3D scene is the rendering equation [ICG86],[Kaj86].

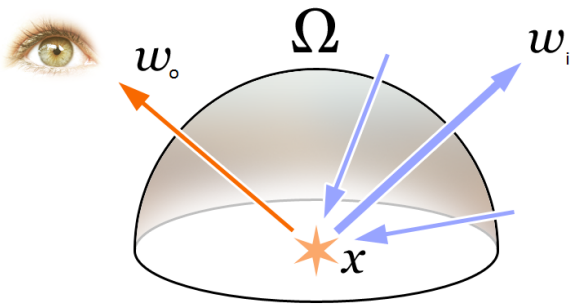


Figure 3.1: Visualization of the rendering equation and its parameters.
(Image source https://en.wikipedia.org/wiki/Rendering_equation)

The equation, which is visualized in Fig.3.1, can be written as:

$$L_0(x, \omega_0, \lambda, t) = L_e(x, \omega_0, \lambda, t) + \int_\Omega f_r(x, \omega_i, \omega_o, \lambda, t) L_i(x, \omega_i, \lambda, t)(\omega_i n) d\omega_i \tag{3.1}$$

where (excerpted):

- x is a point in space.
- ω_o is the direction of the outgoing light.
- λ is a specific wavelength of light.
- t is time.
- $L_e(x, \omega_0, \lambda, t)$ is the emitted light, that gets directly emitted from the surface.
- $f_r(x, \omega_i, \omega_o, \lambda, t)$ is the bidirectional reflectance distribution function of the surface at position x. More details on this will follow in chapter 3.2.
- $L_i(x, \omega_i, \lambda, t)$ is the incoming light.
- Ω Hemisphere around the surface normal n which contains all possible values for ω_i.

In short, the equation describes the amount of light that is emitted from a point x in the direction ω_o. To calculate that amount of outgoing light, all the incoming light from all possible directions of Ω have to be integrated over. If this equation is solved for every surface point in the scene and for every light ray entering the camera, the rendering of the scene would be complete and photo-realistic. Image order rendering tries to do exactly that. But it is easy to see, that it is computationally not possible to solve the equation completely for all directions and all surface points of a scene. The integral part of the equation would be to big, because the $f_r(x, \omega_i, \omega_o, \lambda, t)$ and $L_i(x, \omega_i, \lambda, t)$ terms would have to be solved for too many points. That is why simplifications and approximations have to be used to render a 3D scene efficiently. There are different integrators, which have the purpose of solving the integral of the rendering equation efficiently (see chapter 3.5.1 for examples). To understand these integrators completely, other fundamentals have to be introduced first: Reflectance models (chapter 3.2), sampling and reconstruction (chapter 3.4) and the Monte Carlo integration (chapter 3.3).

3.2 Reflection Models

A big part of the rendering equation is the term $f_r(x, \omega_i, \omega_o, \lambda, t)$ which represents the properties of the surface the point x lies on. The reflected light of a surface is of course dependent on the material of the surface. A Christmas tree ball reflects light different than a ball made out of wood. Surfaces can for example differ in glossyness, reflectance, shininess or roughness. Some examples for different reflection properties of materials can be seen in Fig.3.2.

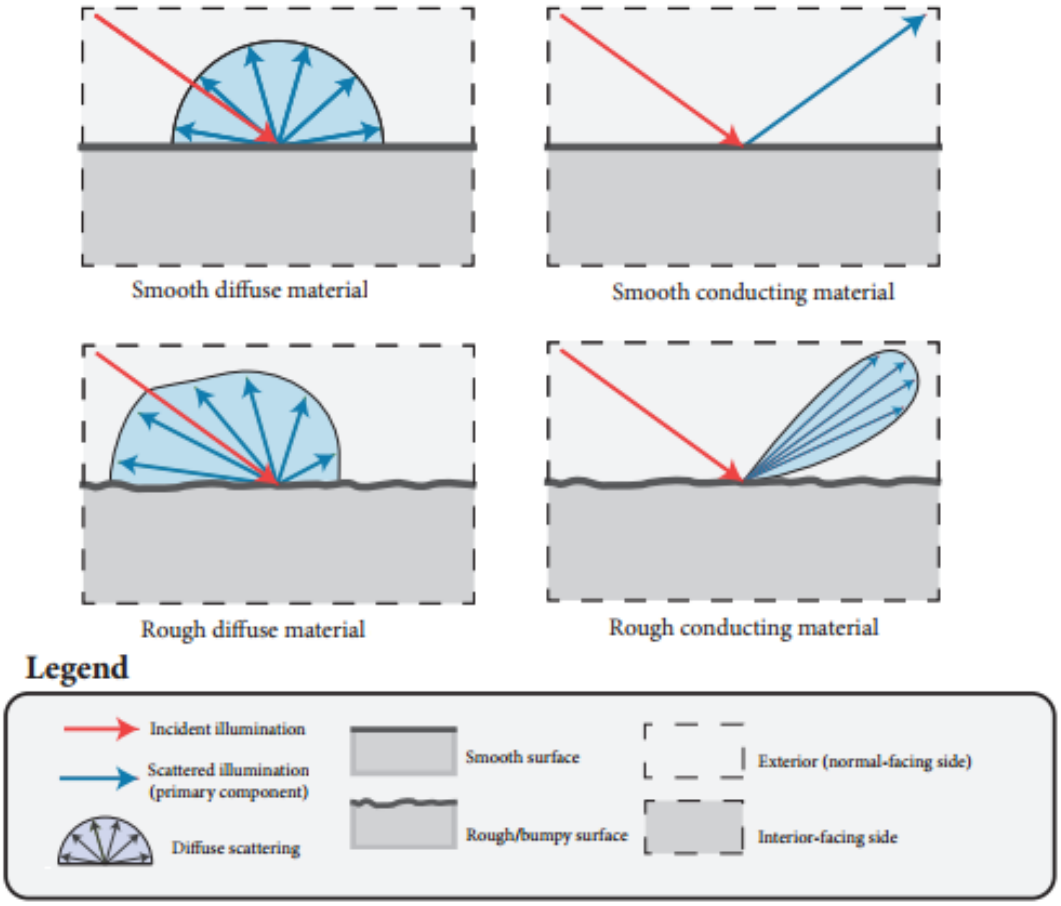


Figure 3.2: Overview for different surface types. These examples are some of the material-types available in the Mitsuba renderer. It shows how incoming light (red arrow) gets reflected on different surfaces. A conductor for example reflects the light completely and unchanged in a direction, while a rough conductor reflects it diffusely in a certain area.
This figure is a changed version of a picture, that was taken from the Mitsuba documentation [Jak14].

This different properties can be described by functions, which are called bidirectional reflectance distribution functions (BRDFs). They describe the behavior of incoming and outgoing light of a given surface, how it is reflected, in which direction in which direction it is reflected, if light comes in from a specific angle.

There are many different types of BRDFs, which describe different types of reflection. Using combinations of many different models, one can precisely define (or approximate) the properties of many materials. When defining 3D scenes with different objects in it, it is possible to give any object a different BRDF, to simulate different materials. BRDFs

can even simulate subsurface scattering(SSS)[HK93]. SSS describes materials, that not only reflect light, but manipulate incoming light under their surface, like human skin or marble. Figure 3.3 illustrates the light transport for SSS.

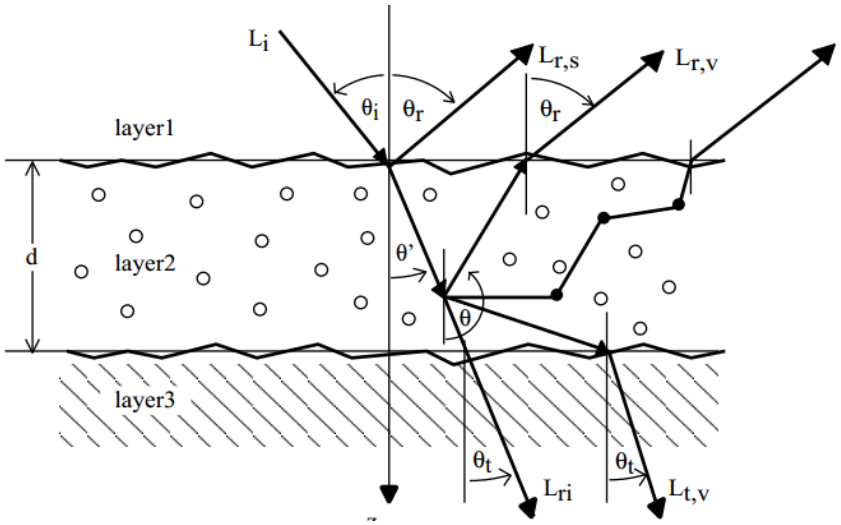


Figure 3.3: 2D Example for subsurface light transport. An incoming radiance (L_i) with an incoming angle of θ gets scattered in layer2 and creates multiple outgoing radiances ($L_{r,s}, L_{r,v}, L_{ri}, L_{t,v}$) on both sides of the surface.
(Image source [HK93])

There are multiple ways of generating the individual forms of those functions for specific materials. They can for example be measured in a laboratory, simulated virtually or calculated from different physically equations that mimic real world light transport.

3.3 Mote Carlo Integration

As stated before solving the whole integral of the rendering equation is too expensive to do. A fast and simple way of solving large integrals numerically is the so called Monte Carlo integration [MRR+53], [VG95]. The main formula of Monte Carlo integration reads:

$$\int_a^b g(x)dx \approx \frac{b-a}{N}\sum_{i=1}^{N} g(x_i) \tag{3.2}$$

The basic idea of Monte Carlo techniques is: Rather than integrating the whole domain of the integral, the result can be approximated by sampling just a couple of points of the domain and summing them up. The samples get chosen randomly or pseudo-randomly.

Of course the correctness of the approximation strongly depends on the number of samples N. A big drawback of Monte Carlo techniques is, however, their convergence. For many integrals a large amount of samples is required to give a "good solution"(one with a small divergence from the perfect solution).

Luckily, there are methods of improving the convergence. Examples for such methods are importance sampling and Russian roulette. All this techniques work by manipulating the random factor of the Monte Carlo integration by taking only "important" or "comprehensive" samples in consideration. Russian roulette for example "eliminates" samples, which are expensive to evaluate, but have a rather small contribution to the overall result. Importance sampling uses density functions to guaranty that mostly preferred samples get drawn, while unimportant samples are less likely to be drawn [OA00].

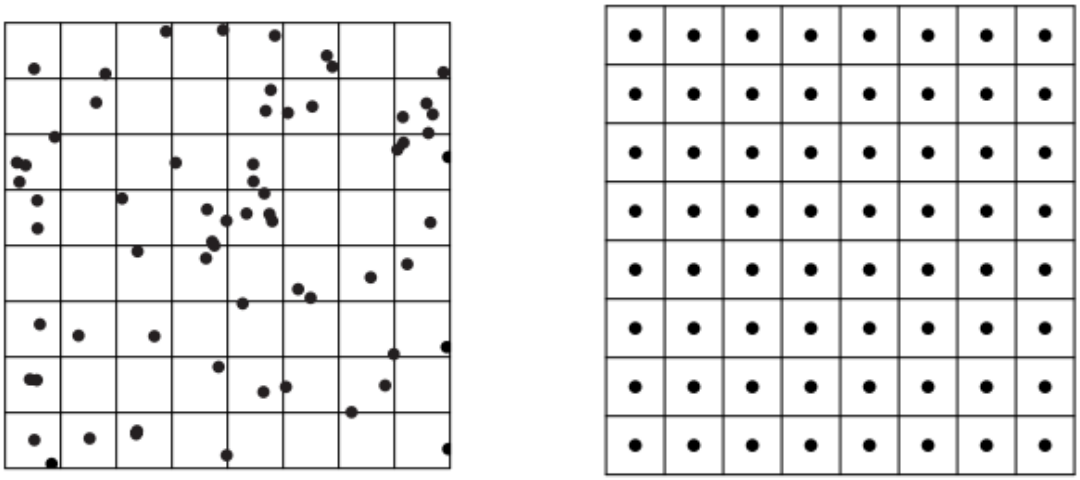
In combination with those methods Monte Carlo integration can be used to approximate the solution of rendering equations integral fast and efficient.

3.4 Sampling and Reconstruction

Now that the cost of solving the integral of the rendering equation is reduced, there is still one problem: Solving the rendering equation for every point in the scene is still impossible, because scenes contain a very large number of points. So reducing the number of considered points in a scene is required. Sampling is a way of accomplishing that [Bra65], [Gla14].

Basically, sampling describes the transformation from a continuous signal into a discrete signal. In case of rendering, the continuous signal is the scene itself and the desired discrete signal is a image with discrete values on specific locations (the image pixels). Producing the discrete color-values per pixel is performed by so called samplers. Basically the samplers choose where exactly samples get drawn. There are many different approaches possible, examples are: random sampling and stratified sampling.

Figure 3.4 shows two examples for sampling patterns. The problem with random sampling is easy to spot: Some pixels do not have samples drawn inside them. Stratified sampling has only one sample per pixel, which can cause problems (aliasing) with thin lines in the scene. A good sampler would be a combination of both approaches. It uses multiple samples per pixel, but the samples are not strictly aligned in an uniform fashion. There are multiple possibilities to achieve that, an example is the "low discrepency sampler", proposed in [KK02]. Which sampler results in the best picture quality, dependents on the scene and the used integrator.



(a) **(b)**

Figure 3.4: Two fundamental sampling patterns.
(Image source [PH10])

After the samples for each pixel have been drawn, the next step is to combine these samples into a concrete pixel value. This step is called reconstruction [Bra65], [Gla14], [MN88]. A filter function gets used to combine all samples within a certain area to a pixel value. Many different filter functions can be used, the most common examples are:

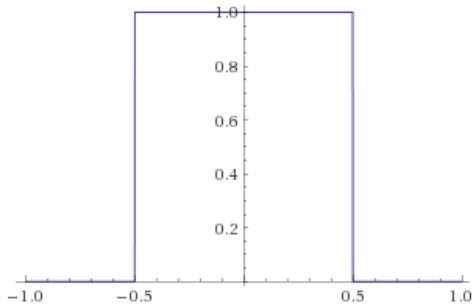
The box function: $b(x) = \begin{cases} 1 & \text{if} |x| < 0.5 \\ \text{otherwise} \end{cases}$

The triangle function: $t(x) = \begin{cases} 1 - |x| & \text{if} |x| < 1 \\ 0 & \text{otherwise} \end{cases}$

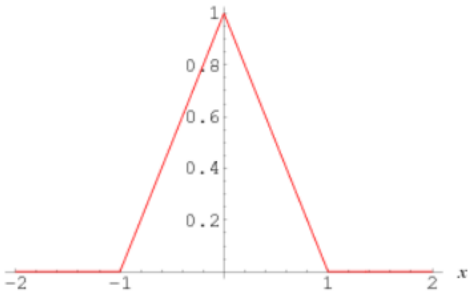
The Gaussian filter function: $g_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{-\frac{t^2}{2\sigma^2}}$, where σ controls the width of the bell curve.

Plots for those functions can be seen in Fig.3.5.

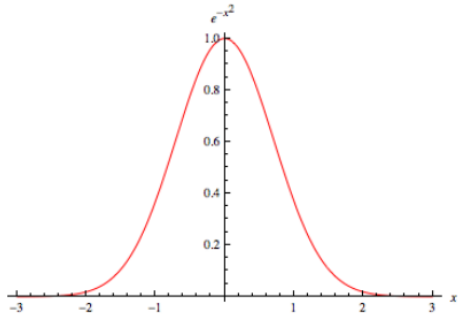
The drawn samples within a pixel can of course simply get summed up to form the pixel value (Boxfilter) or they can get summed up weighted. The Gaussian and the triangle filter for example weight samples in the middle of a pixel higher than samples on the edge of the pixel. The different techniques for reconstruction can eliminate or create artifacts, like aliasing ([Cro77]) or ringing (see Fig.3.7 and Fig.3.6 for examples of those effects).



(a) Box Filter



(b) Triangle Filter



(c) Gaussian Filter

Figure 3.5: Plots of the three basic filter functions. The plots were created with Wolfram Alpha (https://www.wolframalpha.com/).

With sufficient sampling and reconstruction the number of points for which the rendering equation has to be solved is reduced, while good image quality is still achievable. How good the image looks, is of course mainly dependent on the number of drawn samples per pixel.

3.5 The Mitsuba Renderer

In the following chapter the Mitsuba renderer, which was created by Jakob Wenzel and is heavylly based on the book "Physically Based Rendering"[PH10] will get introduced. It uses many different (mainly image order) state-of-the-art rendering techniques to render pictures from virtual 3D scenes. Mitsuba comes with many different predefined modules to describe and render scenes: Light sources, integrators, materials, objects, textures, camera models and many more are included for usage. All modules are implemented with performance in mind and are highly controllable by the user trough parameters. It also offers options for distributing render jobs over multiple machines and cores across networks or clusters to improve performance. The user can interact with all those modules trough a GUI (as seen in Fig.3.8) or trough a command line interface.

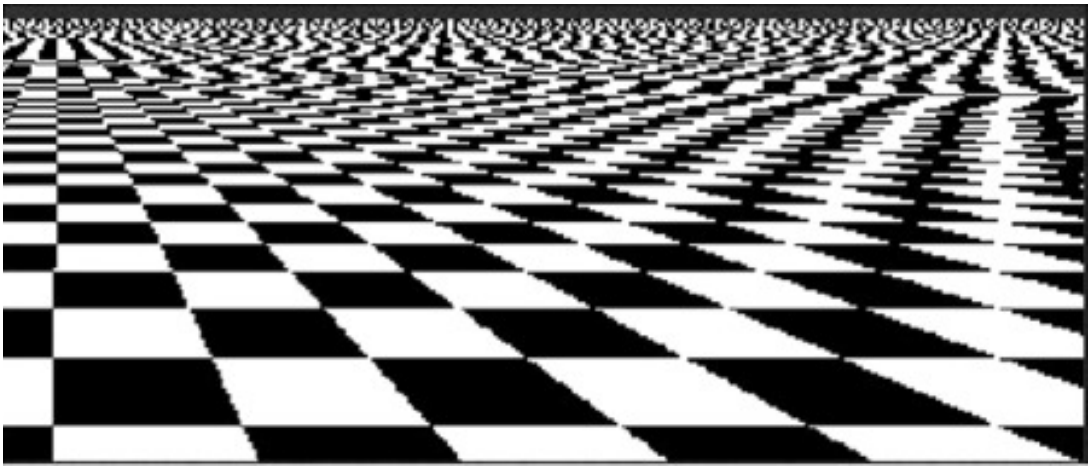


Figure 3.6: Example for aliasing on a checkerboard. The edges of the squares are jagged.
(Image source [KA04])

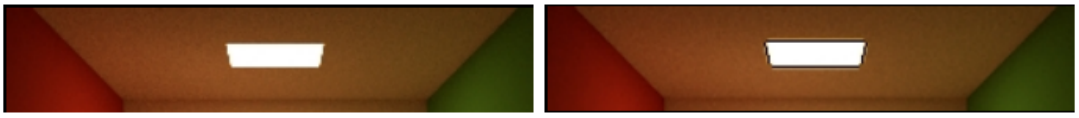


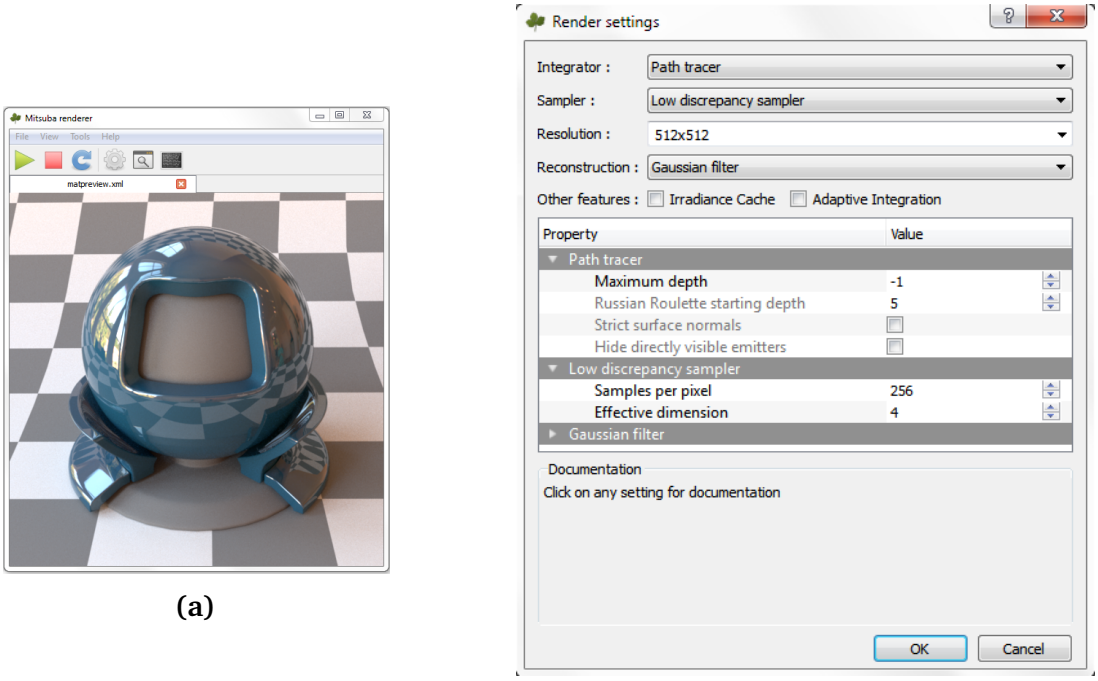
Figure 3.7: Example for the ringing effect: The pictures show the light source in the Cornell Box (see chapter 5.1.1). The left picture shows the light source without ringing, the right picture with ringing. The light source is surrounded by a black frame. This effect can occur in extreme changes in brightness.
(Image source [Jak14])

It is also open-source and highly expandable, so existing modules can get altered or new plugins can easily implemented and deployed. There are three major ways a programmer can customize or extend Mitsubas modules: The C++-Interface, the Python-Interface and the XML scene format.

The C++-Programming interface allows programmers to create custom implementations of nearly all of Mitsubas module types. Examples are emitters, films, integrators, mediums, reconstruction filters, samplers, sensors, texture-types and shapes. He can either modify existing ones or create new plugins from the ground up to add new functionality to Mitsuba.

With the Python interface render jobs and scene files can be controlled. The user can write scripts to create or alter scenes, set parameters and start render jobs.

Mitsuba uses its own XML format to describe and define scenes. The numerous predefined plugins of Mitsuba can easily be inserted into a scene by simply modifying its XML



(a) **(b)**

Figure 3.8: The main window of the Mitsuba GUI (a) and the render settings menu (b).

file. The user has full control over the scenes properties, including all of its objects, the camera, the emitters and many more. Custom made plugins can also get integrated into scenes as well. A simple example for the XML description of a scene and its rendering can be seen in Fig.3.9.

Its high customizablility and powerful tool set are the main reasons Mitsuba was chosen to implement the simulation of the acquisition and rendering of light fields. But because Mitsuba offers a high number of different rendering techniques, it is important to compare them to find out which are suited for light field acquisition or rendering and which are not.

To better understand these techniques, all of Mitsubas predefined integrators will get introduced in the following subsection.

More information about Mitsuba and all its properties and functions can be found in its very detailed documentation [Jak14].

```
<?xml version="1.0" encoding="utf-8"?>

<scene version="0.5.0">
<shape name="testFlaeche" id="testFlaeche" type="rectangle">
        <transform name="toWorld">
            <scale value="5.0"/>
            <rotate x="0.00" y="1.00" z="0.00" angle="180.00"/>
            <translate x="99.9963" y="0.00" z="1.00"/>
        </transform>

        <bsdf type="diffuse">
            <texture name="reflectance" type="bitmap">
                <string name="filename" value="lena.jpg"/>
                <string name="filterType" value="ewa"/>
            </texture>
        </bsdf>
    </shape>

</scene>
```

(a)

(b)

Figure 3.9: An example for a Mitsuba XML scene description file (a) and the corresponding rendering (b). The scene defines a rectangle, its toWorld transformation and its surface properties (A texture from a bitmap file, containing the "lena" picture).

3.5.1 The Different Rendering Techniques of Mitsuba

Mitsuba offers many different integrators, samplers and reconstruction filters to render realistic looking pictures. As mentioned, the task of integrators is to solve the integral part of the rendering equation. Straight forward solving of the integral is computational not possible, because the number of equations would be too high. Integrators deploy different strategies to reduce the number of needed equations.

In this chapter the available Mitsuba integrators will be described. These integrators are very different, they all have different strengths and weaknesses, but are all state-of-the-art approaches. In later chapters of this thesis those techniques will be compared on how suitable they are for acquiring or rendering light fields.

Ambient Occlusion

Ambient Occlusion is one of a few integrators of Mitsuba, that does not generate realistic looking renderings. It is however very simple and fast. Every object in the scene gets light from uniform illumination from all directions. The only effect that can be simulated that way is what objects are visible and where shadows and occlusion occurs (hence the name). It is often used to calculate the shadows between adjoined objects and edges, for example at corners or cracks (for example in videogames). As seen in Fig.3.10 the Ambient Occlusion integrator creates only uni color pictures.

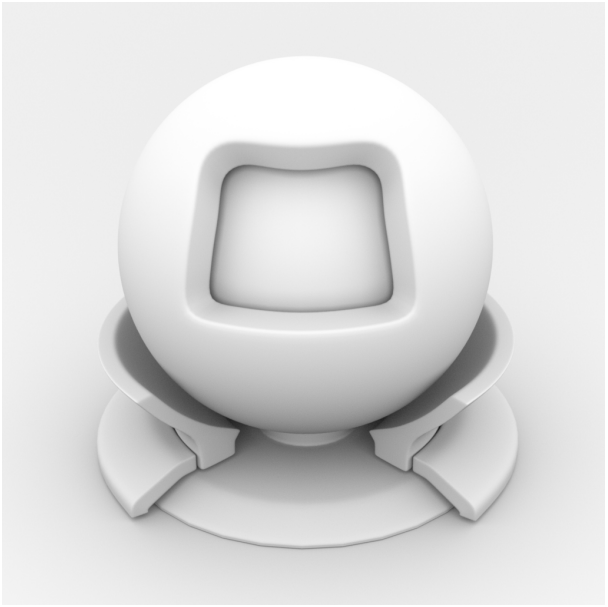


Figure 3.10: Rendering of the Matpreview example scene with Ambient Occlusion.

Direct Illumination

Direct Illumination is also a very fast and simple integrator. For every pixel multiple BSDF and emitter samples get drawn and then combined via different heuristics. The user can control how many BSDF and how many emitter samples the integrator will use. A high number of BSDF samples will result in high quality at glossy objects, while every other object will not benefit from them. A high number of emitter samples leads to poor quality with glossy objects, but works good for every other type of object. So the best choice of those parameters is depended on the scene.

The speed an quality of the Direct Illumination integrator comes at a price tough: It can not handle indirect illumination between a scenes objects, as seen in Fig.3.11, where the reflection (indirect illumination) from the checkerboard in the glossy surface is missing.

Path Tracing

Ray Tracing is one of the most standard integration techniques in image order rendering and Path Tracing is another form of Ray Tracing. While in Ray Tracing the rays are finite, the paths in Path Tracing can theoretically be infinite. But they are both based on the same simple idea: Solve the rendering equation directly by shooting rays from the image pixels into the scene and trace their path. For every intersection between a

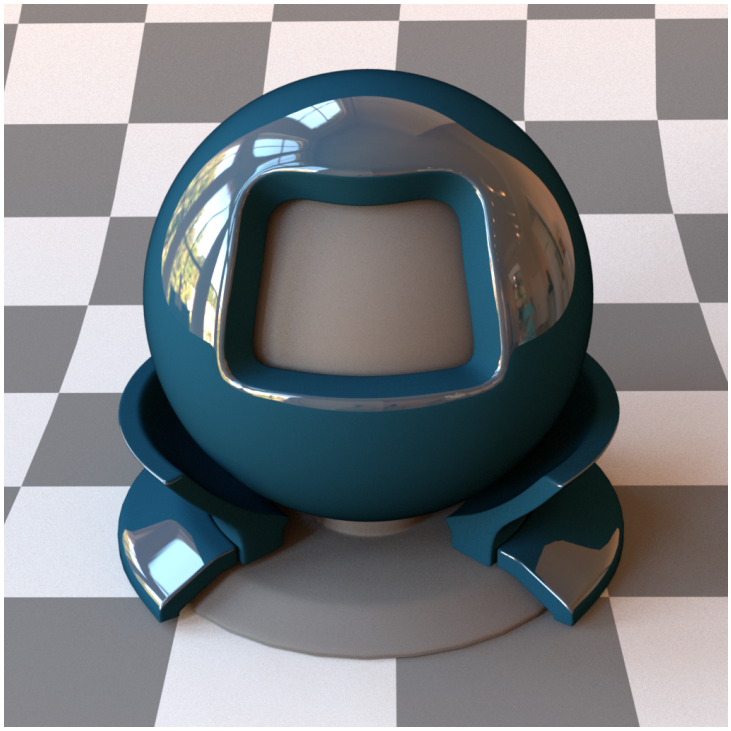


Figure 3.11: Rendering of the Matpreview example scene with Direct Illumination. The indirect illumination from the checkerboard is missing.

ray and a scene object solve or approximate the rendering equation for that point (see chapter 3.3). The ray gets reflected in a (semi-)random direction and the new path is also traced. This gets repeated until a light source gets hit, or an other end criteria is met, as illustrated in Fig.3.12.

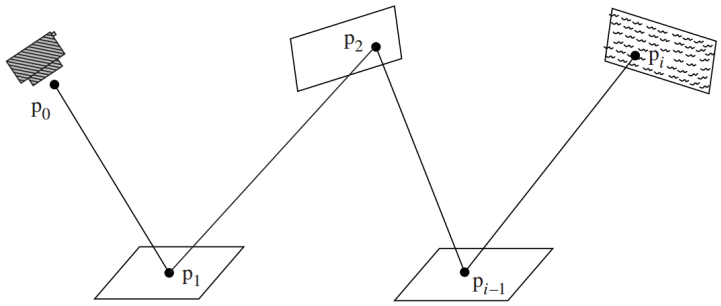


Figure 3.12: Illustration on how path tracing works. The ray gets shot from a point in the camera (p_0) into the scene. It intersects objects at the points p_1 - p_{i-1} and finally hit a lightsource at point p_i.
(Image source [PH10])

When a light source gets hit by a ray, this ray gets a color value assigned which is based on where the light source was hit and what path the ray had to take to get to the source. To create realistic looking pictures multiple rays have to be sampled for every pixel. Problems with Path Tracing occur when light sources in the scene are hard to reach. Then for many rays no color value can be calculated or paths have to be traced very long to reach an emitter (which is really bad for the performance). This problem also occurs with caustics, where the ray can "get lost" in an object. Path Tracing is the base for many other techniques that improve upon its basic concept, mostly by improving the choice of the reflected path at intersection points for example to reach light sources more quickly or more reliable.

Volumetric Path Tracing [LW96]

Volumetric Path Tracing (as the name implies) is a extension of normal Path Tracing. It can handle participating media like for example fire, smoke or fog, which can not be done with a standard Path Tracer. This is achieved by allowing light rays to be scattered in a medium. On simple surfaces and objects Volumetric Path Tracing acts exactly like normal Path Tracing.

Mitsuba comes with a simple and an extended version of a Volumetric Path Tracer. The simple method (in contrast to the extended version) does not use importance sampling and is therefor faster, but can cause poor image quality with glossy objects and caustics.

Adjoint Particle Tracer

Instead of tracing rays from the camera to emitters, the Adjoint Particle Tracer takes the opposite approach: It shots single particles from the emitter into the scene and tries to find a path into the camera. This approach is only beneficial in very special cases and the Mitsuba documentation discourages the use of the Adjoint Particle Tracer.

Photon Mapper [Jen04]

The Photon Mapper is a multi-pass approach, which means it is divided into two separate steps. Pass one shots photons from the emitters into the scene and tracks where they land. If enough photons where shot, three different "photon maps" can be built, one for every illumination type (direct, caustic and indirect). This maps save how many photons landed on every point in the scene. Those maps are viewpoint independent, so they can be reused for different camera positions. In the second step the photon map gets used to

render the picture. There are different possibilities for this, Mitsuba uses recursive Ray Tracing. In short, paths of rays get traced and all photons along the path get collected and used to calculate the color-value of the sample.

The problem with this approach is that the output pictures can look very blotchy, unless a very large number of photons was used to create the photon maps. But if too many photons get used the rendering can look blurry.

Progressive and Stochastic Progressive Photon Mapper [HOJ08], [HJ09]

Progressive Photon Mapping is a progressive version of the Photon Mapper, where photon shooting- and gathering-steps are alternated. After every gathering step a small number of photons get erased from the photon maps, to save system memory (photons maps for a large number of photons can get big). A special property of such progressive approaches is that the image quality can alter (preferably improve) over time. So "good" renderings can be achieved with a short runtime, while "better" results can always be achieved by letting the progressive photon mapper run longer. Of course the convergence of the results quality should be high. The stochastic progressive photon mapper improves the convergence of the normal progressive approach in certain situations, for example for scenes with motion blur or glossy reflections.

Virtual Point Light(VPL) [Kel97]

This approach is a multi-pass technique like the Photon Mapper. In a first step many different virtual light sources get created in the scene. The positions of those VPLs are based on the scenes emitters and indirect illumination. After these VPLs get created, the scene gets rendered one time for each VPL, each time with only that specific VPL as lightsource. Finally the separate rendered images get combined into the final rendering result. An interesting property of this approach is that the separate renderings for every VPL are independent from one another, so they can be calculated in parallel on a graphics card. Glossy objects can cause problems for this technique and often artifacts on the corners of objects occur.

Bidirectional Path Tracer [LW93]

Bidirectional Path Tracing is a straight up improvement strategy for normal Path Tracing. It solves the problem, that the standard approach can not handle hard to reach light sources. This is achieved by not only tracing rays in one direction, but in two directions: One ray gets shot and traced from the camera, just like in normal Path Tracing. But this

time an additional path gets created from the light source as well. The renderer then tries to combine those two paths as often as possible without occlusions. This way, paths to light sources can be found faster and more reliable, even if they are hard to reach. Additionally the found paths are weighted based on how useful they are for the final result for that pixel (multiple importance sampling).

Usually the bidirectional variant of Path Tracing creates less noise than the standard algorithm, but is computationally more heavy (by factor 3-4). It also can not be parallelized well, because multiple importance sampling always needs the whole picture and can not separate the picture in different tiles, which could be distributed over multiple cores.

Primary Sample Space Metrolpolit Ligth Transport(PSSMLT) [KS01]

PSSMLT is an algorithm in the class of Markov Chain Monte Carlo(MCMC) methods. It is built as a layer on top of normal Path Tracing or Bidirectional Path Tracing. With the help of Markov chains it permutes found paths of those integrators in sample space to create new paths. For an example see Fig.3.13.

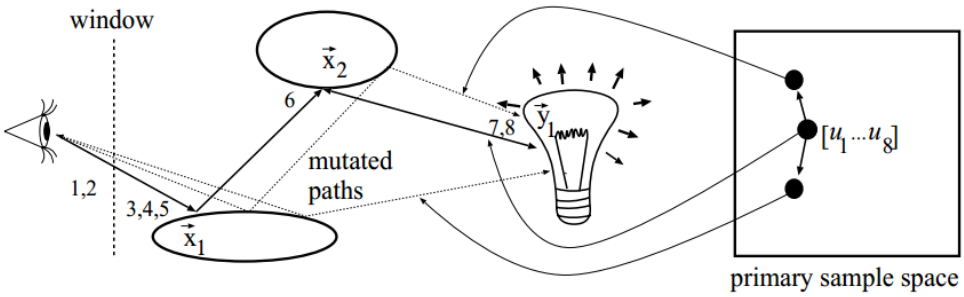


Figure 3.13: Example for the permutation in PSSMLT. Random numbers ($u_1 ... u_8$) get chosen to find a path from the eye to the light source. u_1 and u_2 define the point where the ray intersects the window. Tracing that ray, it hits an object at $\vec{x}_1$. There u_3, u_4 and u_5 get used to determine in which direction the ray is reflected. At the next intersection point $\vec{x}_2$ the ray gets reflected too and finally hits the light source in $\vec{y}$. The complete path can be defined with the random numbers $u_1 ... u_8$.
The Markov chains permutation creates mutated paths by slightly changing these random numbers. This can result in better paths, that reach the light source faster (or at all). (In this simple example no better path can be found.)
(Image source [KS01])

The MC-permutation is carried out in a way, that more "relevant" paths are found. This way difficult scenes can get rendered faster and with less noise. For simple to render scenes however PSSMLT does not offer any benefits and can also perform worse, because of the additional computational overhead needed to permute the light paths.

Path Space Metrolpolis Light Transport(PSMLT) [VG97]

PSMLT is very similar to PSSMLT. It also uses Markov chains to permute light paths in the scene to find "good" ones. But unlike PSSMLT it does not permute paths that where created with other Path tracing methods, but instead creates its own path type and directly works on them. This way the algorithm has more direct control over the light paths. In the Mitsuba documentation it is stated however, that the ways the paths are calculated and changed are very complex, so many unforeseen problems could occur while rendering [Jak14].

Energy Redistribution Path Tracing [CTE05]

Energy Redistribution Path Tracing is another MCMC-method. Path Tracing get combined with perturbation strategies of Metropolis Light Transport. It uses two passes to render a image: The first step produces paths with a normal Bidirectional Path Tracer, the so-called seeds. In the second step every seed gets permuted with a Markov chain to improve them. Unlike with PSMLT and PSSMLT the seeds do not get mutated only in sample space, but also along the path that they take. That way every sample and the whole picture can, in theory, achieve better quality than those other MLT methods. If the Bidirectional Path Tracer produces a good result, Energy Redistribution Path Tracing can be used to make the results even better.

3.6 Light Field Rendering

As discussed in chapter 2.2, light fields consist of many rays, that represent a recorded scene. To render the scene, the rays of the light field have to be recalculated into pixel values. The more dense the light field is for a sampled point, the more accurate the rendering will be for that position. All rays that represent the sample have to be chosen and combined to form the sample value. Multiple ways to do that are possible. The method used in this thesis will get introduced in chapter 4.2. Later, in chapter 6, the Mitsubas rendering techniques will be evaluated, on how well they are suited for this task.

In this chapter the basic concepts of rendering images where discussed, as well as the Mitsuba renderer and all its rendering techniques. The next chapter will introduce the software framework, which was created on top of Mitsuba for light field acquisition and rendering of light field data.

4 Custom Software Framework Using Mitsuba

As stated, one goal of this thesis is to simulate the acquisition and rendering of light fields. In chapters 5 and 6 Mitsuba, which was introduced in chapter 3.5 will be used to execute this simulations. Unfortunately, Mitsuba does not support light fields out of the box. To use Mitsuba for this purposes, custom software solutions on top of Mitsuba, had to be created. This was possible because Mitsuba is open-source, customizable, modular and offers many interfaces for programmers. For the virtual acquisition of light fields a custom framework with GUI was created and for rendering light fields a purpose-made emitter module was created. In this chapter both the framework and emitter-plugin will be introduced.

4.1 Simulation of Light field Acquisition

When simulating the acquisition of light fields, it is required, that the simulations are highly controllable trough parameters. To achieve that, a GUI was created, which allows the user to control nearly every parameter of the simulations, as well as their output. He has the choice between two separate simulation modes: Recording the light field via different camera positions (see chapter 2.3.2) or a mirror setup (see chapter 2.3.4). Both simulations have different settings menus, as seen in Fig.4.1 and Fig.4.2.

For both simulation types a XML-file containing the scene informations has to be selected[1].

If wanted, camera parameters like resolution, position, up-vector and lookat-vector from the selected scene file can be overwritten and changed manually in the GUI.

For the first simulation the number of different camera positions, their displacement and the option if they should be arranged in a planar or spherical fashion, can be set up.

[1]Note: This file has to be located in the same folder as the program, so later Mitsuba can find the file

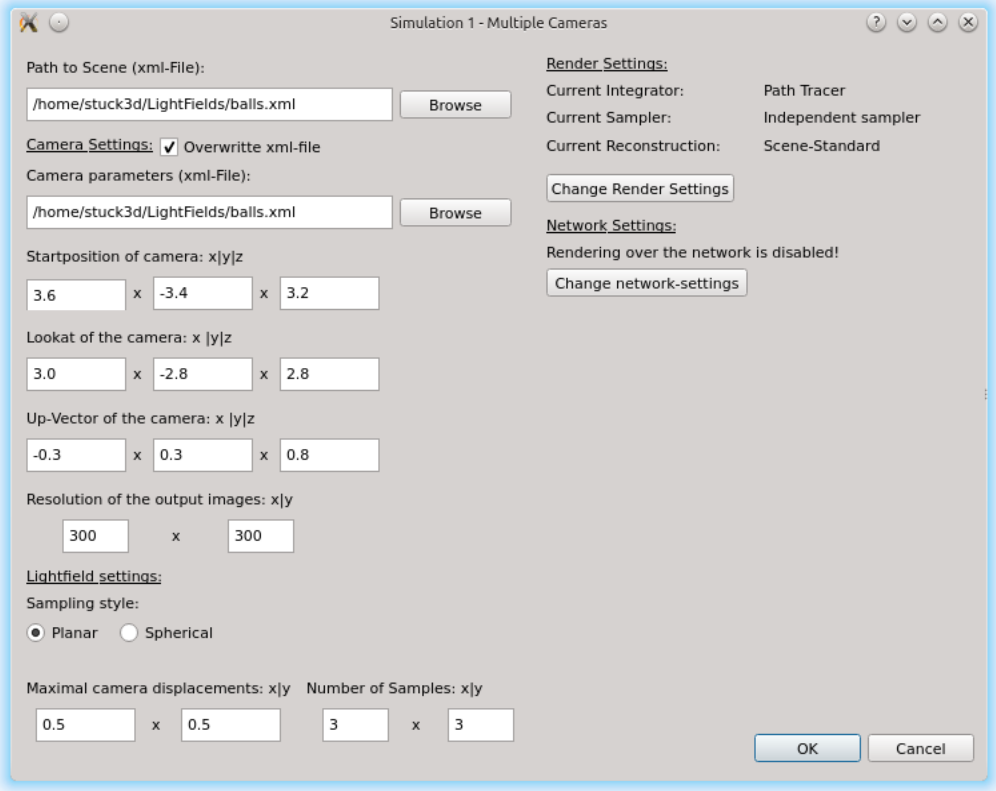


Figure 4.1: The settings menu for the camera position simulation.

For the second simulation type, the mirror parameters can be changed if the mirror is not yet defined in the scenes XML-file. The position of the mirror, the size and number of the single mirror facets and the angle between them can be changed. Later a mirror with the selected properties will be created and inserted into the scene.

For both simulation types the framework allows for full control over all integrator-, sampler- and reconstruction-settings via the rendersettings-submenu (Fig.4.3). The menu is case dependent and changes appearance depending on the chosen settings. In this menu all possible parameters from Mitsuba can be set.

It is also possible to use Mitsubas ability to use multiple network nodes for the simulations. In the network settings menu network nodes can be inserted.

The flow of the simulation can be seen in Fig.4.4. As mentioned, the user interacts with the C++ GUI and changes the settings of the simulation. The settings then are saved in XML files, which are the input for the automatically invoked Python script. The scripts start Mitsuba automatically with the selected render settings and create XML output files

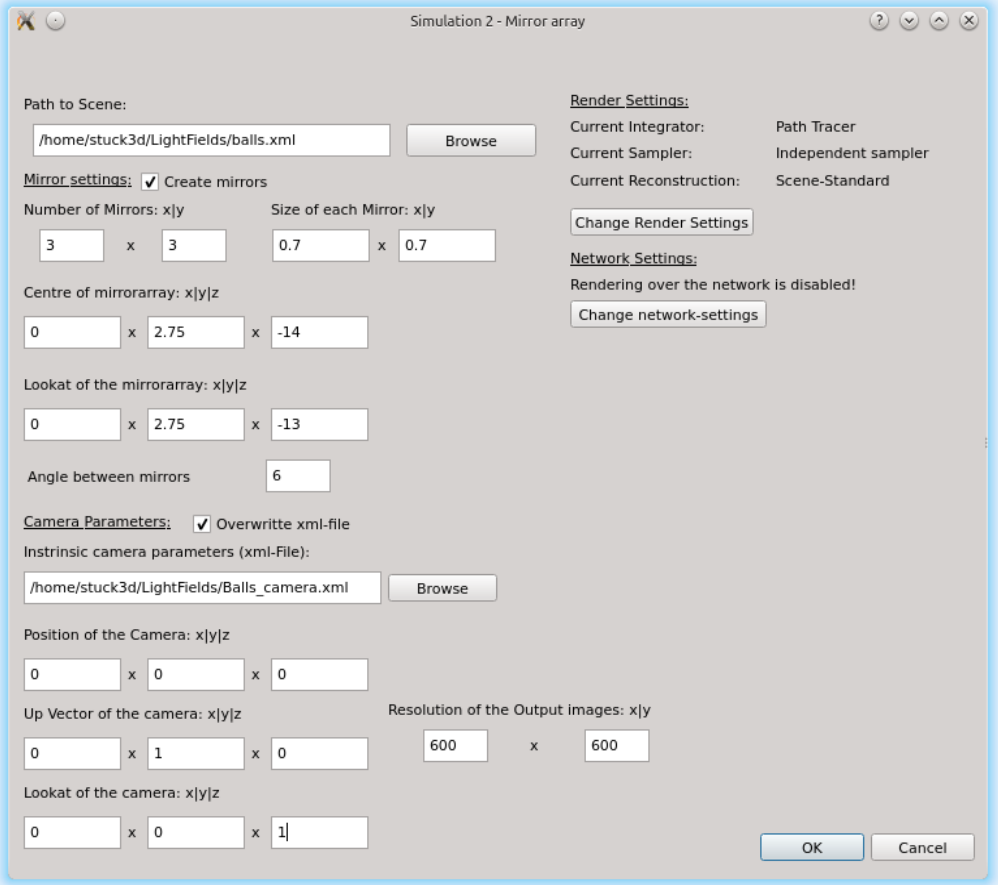


Figure 4.2: The settings menu for the mirror setup simulation.

with information about the render process (for example the render time), the chosen settings and the cameras extrinsic and intrinsic parameters. Alternatively the user can manipulate the settings XML files and invoke the Python scripts manually. That way multiple similar light fields can be created and compared, without using the GUI every time.

4.2 Rendering of Lightfield Data

For rendering light fields, a custom Mitsuba emitter, named "lightfield" was created. The emitter works roughly like the standard area emitter from Mitsuba. A normal area emitter emits diffuse light from a arbitrary shape into the scene. The custom emitter can be added to every object. It uses the synthetically created light fields from the camera

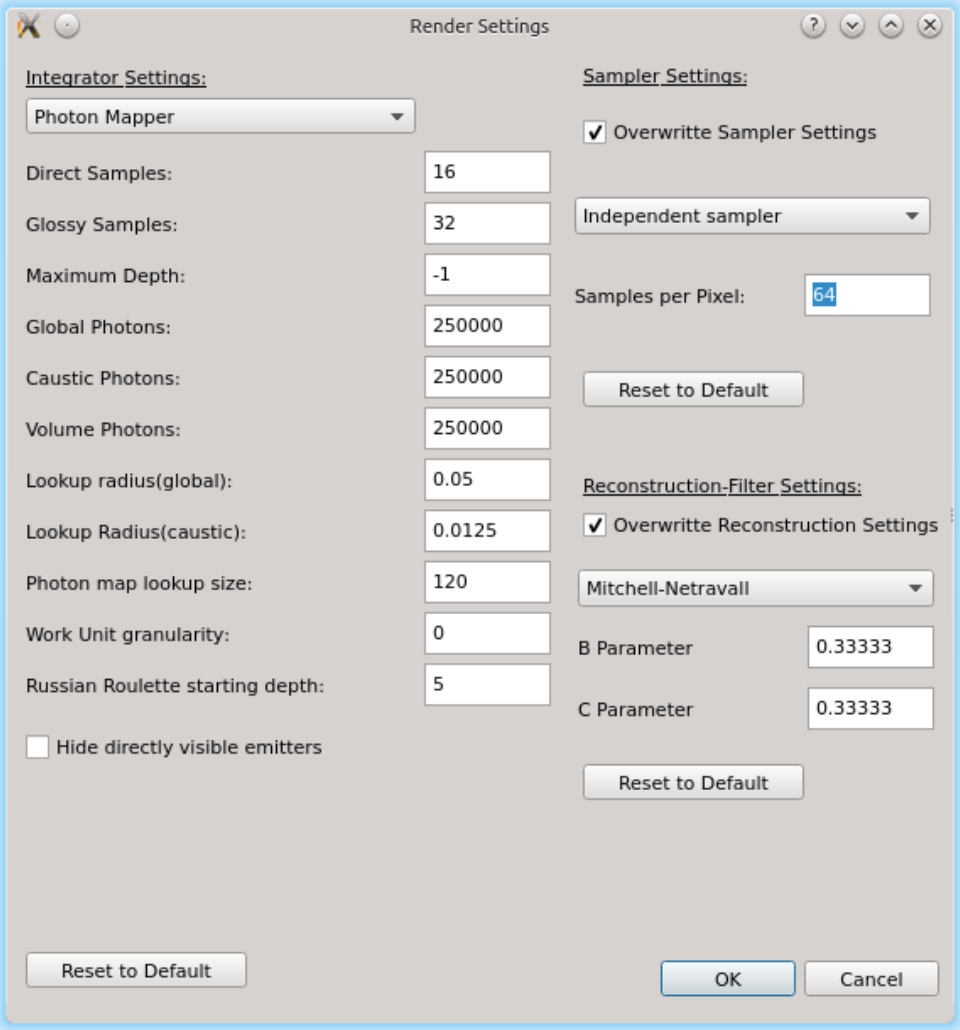


Figure 4.3: The render settings menu.

simulation framework as input. The reason only synthetic light fields were used is that no easy to use real world light fields were available at the time of the thesis creation. The emitters only parameter is the data path to the folder containing all the XML and image files that where generated by the simulation. For an example on how the emitter can be used in scenes, see Fig.4.5.

The emitter loads all pictures from the given path and for every pixel calculates the corresponding ray, that was projected onto it. After that, every ray is intersected with two parallel planes, to get their u,v,s,t-parameters (see chapter 4.2). Subsequently these parameters are saved in a kdTree. KdTrees are a special datastructure, that (among other

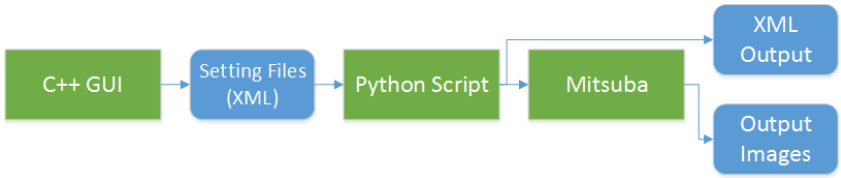


Figure 4.4: The main flow of the simulation framework.

```
<scene version="0.5.0">

<shape name="testFlaeche" id="testFlaeche" type="rectangle">
        <transform name="toWorld">
            <scale value="3.0"/>
            <rotate x="0.00" y="1.00" z="0.00" angle="180.00"/>
            <translate x="0.00" y="0.00" z="10.00"/>
        </transform>
        <emitter type="lightfield">
            <string name="filename" value="/Lightfield/"/>
        </emitter>

</shape>
</scene>
```

Figure 4.5: An Example scene containing a rectangle with a light field emitter.

things) allow fast searching of nearest neighbors in multi dimensional spaces [Ben75]. When the scene gets rendered later on, every sample ray, that reaches the emitter, gets its u,v,s,t coordinates calculated as well. Subsequently the nearest neighbor for that sample (the ray with the most similar u,v,s,t coordinates) gets searched in the kdTree, to get the sampled radiance. An 2D example for that process is shown in Fig.4.6.

For that approach to work properly, the two planes have to be parallel to each other. Also the scenes camera has to point in the same direction as the cameras from the light field. For that reason two assumptions have to be met, when using the light field emitter class in a scene:

1. The camera of the scene is positioned on the x,y-plane, so the z-component of its position has to be zero.
2. The camera looks in positive z-direction.

Because of this two assumptions, no camera rotation is supported.

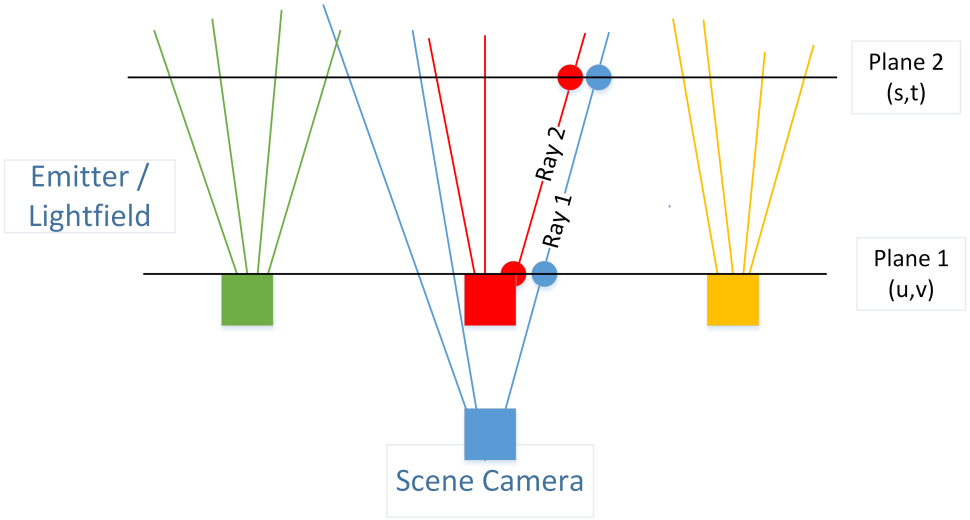


Figure 4.6: A simplified 2D schematic of a scene containing a light field emitter. The blue rays are the rays, that are coming from the scenes camera. One example for those rays is be "Ray 1". The blue dots are the intersection points from "Ray 1" with the predefined planes. The coordinates of that points form the u,v,s,t coordinates for "Ray 1". The light field contains all the green, red and orange rays. The nearest neighbor for "Ray 1" is the red ray "Ray 2", because their intersection points on the planes (and therefor their u,v,s,t coordinates) are closest together. So the returned radiance for "Ray 1" is the radiance, that was originally recorded for "Ray 2". This procedure is repeated for every camera sample.

4.3 Requirements and Limitations

The software framework requires multiple third party programs to be installed and setup correctly. A table of all required packets can be seen in table 4.1.

The software was only tested on a Linux system (Ubuntu 14.04), but should also work on Windows and MacOs systems. Depending on the resolution and the number of pictures, it can take multiple minutes to load a scene that contains a light field emitter into Mitsuba. This is because for every pixel of every picture the light slab has to be computed.

Unfortunately libAnn, which is used by the lightfield emitter is not thread-safe, so different rendering cores can not access the same kdTree at the same time, without complications. So when rendering a scene, containing a lightfield emitter, Mitsuba has to be setup to only use one rendering core.

Packet	Description
Python	Required for running the Python Scripts.
C++	The GUI was written in C++.
Qt	The GUI was created using the Qt-framework.
Mitsuba	Mitsuba and the Mitsuba development tools must be installed correctly
lxml	Handles the parsing of XML for the input and output files of the Python scripts
OpenCV	Handles the loading of image files into the emitter. Also used for the calculation of the rays.
libAnn	Is responsible for the creation and organization of the kdTree of u,v,s,t coordinates in the emitter.
Boost	The Boost lib is used for XML-parsing in the emitter class.

Table 4.1: Requirements for the simulation framework.

5 Simulation of Light Field Acquisition using Mitsuba

In this chapter Mitsubas capabilities to simulate the acquisition of light fields will be examined. The custom software framework, that was introduced in the previous chapter will be used to perform two different simulations (with multiple camera positions and a mirror setup) on four different test scenes to generate virtual light fields. Subsequently the performance and suitability of Mitsubas different integrators will be compared and discussed.

5.1 Simulation Procedure and Restrictions

For the simulations to create meaningful results, first the execution and their restrictions have to be defined. In the following section the test scenes, the used integrators, the used hardware, what values were measured for every simulation run and how they were conducted, will be introduced.

5.1.1 Scenes

To guarantee comparability, both simulation types are executed on the same four scenes. The used scenes were chosen to have different properties, to assure that they cover a wide range of possible real world scenarios. The four scenes are:

Matpreview

The first scene chosen was the "Matpreview" example scene from the Mitsuba home page[1]. An example rendering of this easy to render scene, can be seen in Fig.5.1.

[1] http://www.mitsuba-renderer.org/download.html

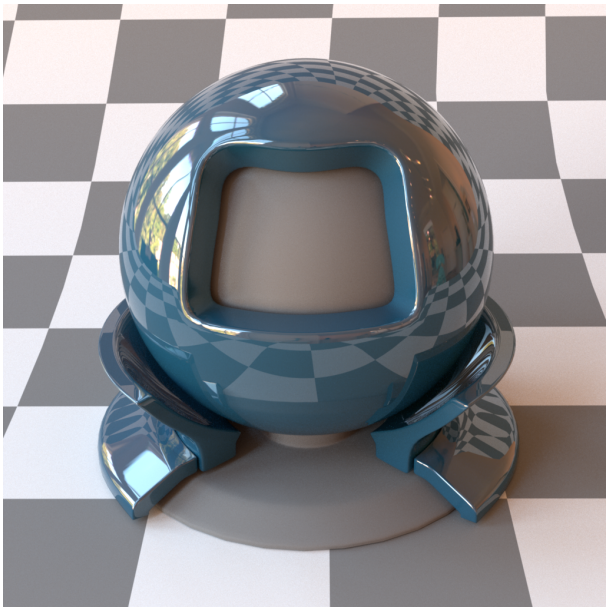


Figure 5.1: Rendering of the Matpreview test scene

It contains only three components:

- A blue ball-like object in the center of the scene. It has a very glossy surface with a cutout in the middle, that shows a matt kernel. The ball is placed on a two parted stand, which have the properties as the kernel and the ball itself.
- A checkerboard in the background of the ball.
- An environment map for illumination.

Cornell Box with Subsurface Scattering

The Cornell Box scene is a popular testing scene for rendering systems and techniques. It consist of a room with a white ceiling and floor, white back wall, red left wall and green right wall. The room is lighted by only a single visible and easy to reach square area light in the ceiling. To test rendering algorithms, varying types of objects are placed into that room. In the variant used here, two different sized blocks, a small one and a bigger one, are placed in the middle of the room. The smaller block is placed in front of the second block and partially occludes it. Both blocks also feature subsurface scattering (compare chapter3.2), to investigate how Mitsubas integrators are suited for scenes containing subsurface scattering.

An example rendering of this scene can be seen in Fig.5.2. This variant of the Cornell Box also was taken from the Mitsuba homepage.

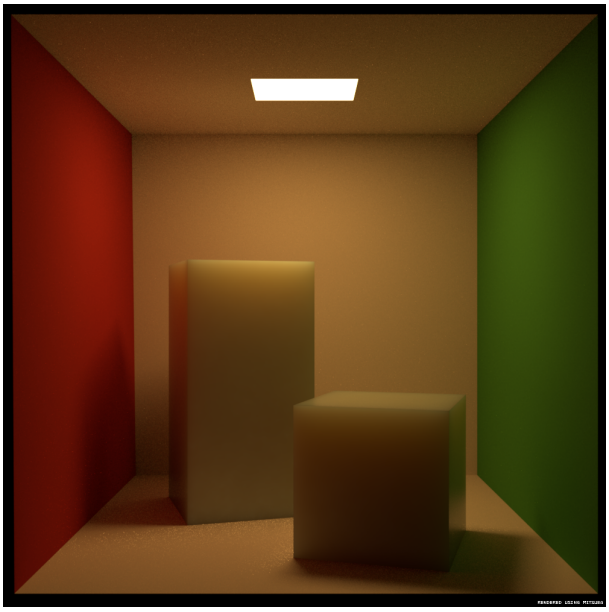


Figure 5.2: Example rendering of the Cornell Box test scene

The reason the Cornell Box is often used to evaluate render techniques, is because it is on the one hand simple, but on the other hand covers three basic and important lightning conditions: There is direct illumination, indirect illumination between the walls and the blocks and the blocks cast shadows on the walls and floor. Because the smaller block stands in front of the second block, the renderer has to handle (simple) occlusions as well.

myBox

The scene "myBox" (that can be seen in Fig.5.4) features a room, which is very similar to the Cornell Box[2]. But this time the back wall is a mirror surface and the light source is placed in a cylinder on the right side of the scene. Two pillars are placed in front of the light source, the right one is made out of glass, the left one is a rough conductor. This scene is very hard to render, for two reasons: Because it is placed inside a cylinder, the light source is very hard to reach. Additionally because one pillar is made out of glass, the integrator has to handle caustics too. This scene was mainly chosen to observe how the integrators perform in such hard conditions.

[2]The scene was created by Martin Fuchs for his lecture "Image Synthesis" at the University of Stuttgart in the year 2015.

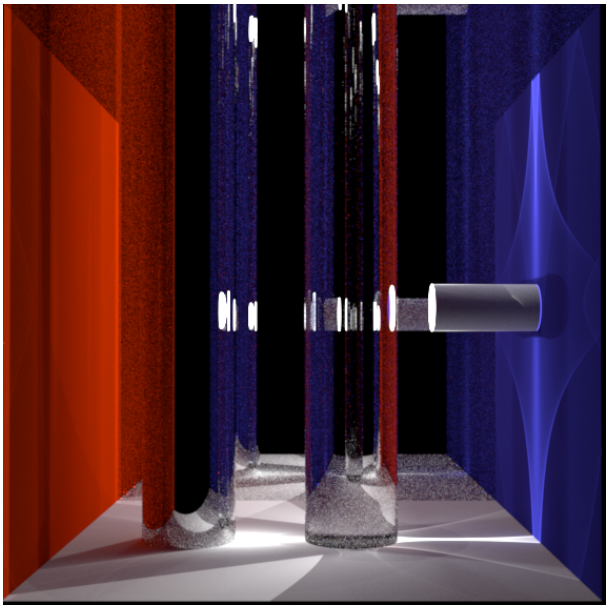


Figure 5.3: Rendering of the myBox test scene.

Balls

The "Balls" test scene consists of many different balls in one room[3]. The balls have different colors and different reflection properties, some of them are made out of glass, some out of plastic and some of them are highly specular. Some of those balls are defined as area emitters, so the scene contains multiple light sources. The scene is like the myBox scene hard to render, because it features a lot of occlusion, caustics, reflections and some areas of the scene are hard to reach from a light source.

This four scenes cover a wide range of possible scene types. Participating media, like smoke, fire or fog was intentionally not included in any scene, because light field acquisition in the real world requires static scenes and therefore those phenomenons get avoided.

5.1.2 Integrators

Four of Mitsubas integrators where chosen not to be included in the simulations.

[3]Parts from this scene are taken from https://mynameismjp.wordpress.com/2015/04/04/mitsuba-quick-start-guide/

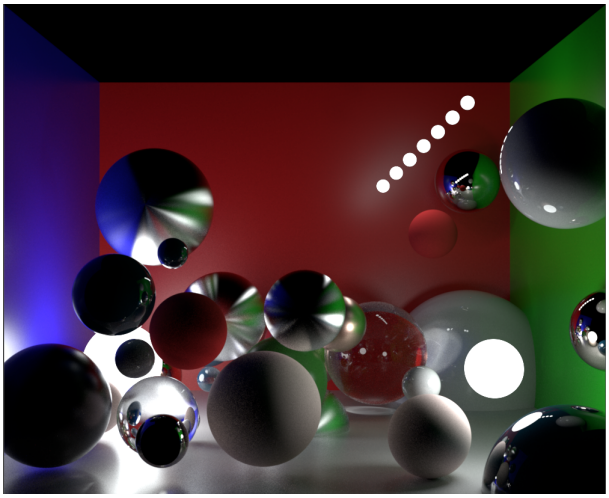


Figure 5.4: Rendering of the Balls test scene.

Ambient Occlusion integrator does not render realistic looking pictures. The goal of the simulations however is to simulate real world acquisition of light fields. That is the reason why Ambient Occlusion was not investigated.

On the test system, on which the simulations were executed, the VPL integrator was not functioning. When trying to use it, Mitsuba would crash. This behavior was independent of the integrator settings or the chosen scene. For this reasons VPL could not be included in the simulation.

Progressive methods in general can produce different results dependent on their run-time and results can differ with repeated runs as well. To produce reliable and comparable results is not possible, because the influence of randomness is too high with such methods. That is why both progressive Photon Mapper methods were excluded from the simulations.

All other Mitsuba integrators, that where introduced in chapter 3.5.1 were tested.

5.1.3 Used Hardware and Software

All the simulations were executed on the private system of the author. The specifications of the test system can be found in Table 5.1. While this system is not high end, its performance is what is to be expected in most single machine PCs today. It is sufficient enough to create applicable simulation results. Of course more powerful hardware or a cluster of PCs can reduce render times or improve render quality radically. But the results of the simulations on such hardware should be comparable.

CPU	Intel Core i5-3570 @ 3.4GHz (4 Cores)
RAM	8192MB DDR3
GPU	NVIDIA GeForce GTX 970
OS	Ubuntu 14.04 (64Bit)
Mitsuba version	0.5.0

Table 5.1: The specifications of the test system.

5.1.4 Measured Values

For every simulation run, multiple factors were recorded to later compare the integrators with each other:

Success
Was the integrator able to successfully render an output or did it just produce a black image or no image at all?

Time
The required render time was measured for every simulation run, mainly to regulate integrator settings and compare techniques.

Quality of the Output
The image quality of the rendered picture was objectively evaluated. "High image quality" in this context means: There are no visible artifacts, there is no noise visible, the picture is sharp, no aliasing occurs and all the lightning effects of the scene are rendered correctly.

Parameters
For every run the applied settings were saved. This includes the settings for the integrators, the sampler, the reconstruction filter, as well as the camera settings (instrinsic and extrinsic parameters) and the light field settings. This was done to make later evaluation easier and allow the usage of the output images as input for the rendering of the light field (see chapter 6).

5.1.5 Procedure

In this subsection the procedure of the simulation are introduced. To make simulation runs comparable, some basic parameters were determined for all of them. For both simulation types, the output image resolution was set to 600x600 pixels. This resolution proved to be a good compromise between render time and image quality.

For the render settings of each simulation run, the default settings from Mitsuba were chosen to build the baseline. If tweaking was necessary, the parameters were adapted accordingly.

For the simulation with different camera positions the following procedure was selected: The simulation was executed three times: The first time with only a single camera position (mainly as reference value), after that four camera positions and finally with nine different positions. Because when creating dense light fields for real world usage, a large number (hundreds or thousands) of individual pictures is required, the rendering time for every picture in a simulation should be relatively short. For that reason the render parameters for every simulation run were set, so they would require between 90 and 180 seconds. If the render time was under 90 seconds, the quality settings for the rendering were raised, if the time exceeded 180 seconds, the settings were lowered. This way it is assured, that the render time for a single picture (and consequently the light field) is not too high, but the images offered enough quality to compare them[4]. To reduce the influence of disparities in the scenes architectures to the render time or image quality, the displacements between the individual camera positions was chosen to be rather small.

For the mirror setup, which only requires one rendering per light field, the following procedure was chosen: A mirror with nine evenly distributed mirror facets was added to the test scenes. It was place at the position, that the scenes camera originally was located at. The scenes camera position was adapted, to show the whole mirror in the rendering. This arrangement was aligned so that the middle piece of the mirror could roughly cover the whole scene. An example for such a mirror arrangement can be seen in Fig.5.5.

In a first step the render settings were chosen to be the same as for the first simulation type for that scene and integrator. Because the mirror setup can virtually create nine camera positions in one rendered image, a second simulation run was executed additionally: The quality settings were raised, so the render time would roughly match the time it took to render nine images with nine different camera positions in the first simulation type.

[4]A much shorter maximum time limit would cause all render techniques to result poor quality images.

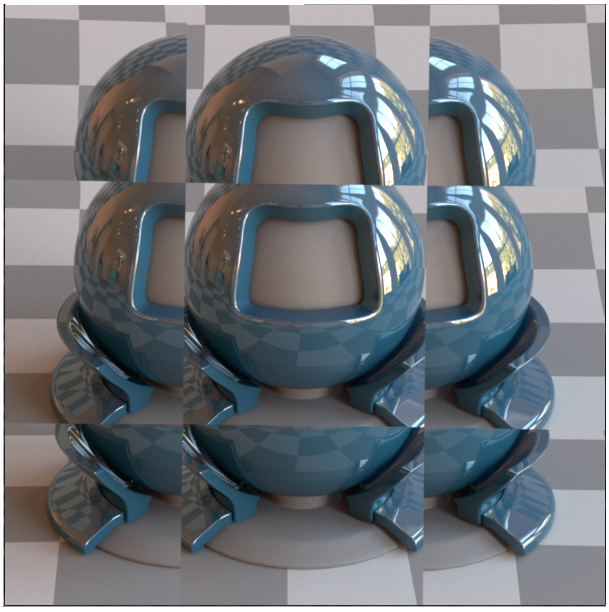


Figure 5.5: Rendering of the Matpreview test scene with a mirror setup.

5.2 Results

In the following subsection the results of the simulations are summarized. The results for the two different simulation types and especially the integrators will be compared with one another.

5.2.1 Performance of the Integrators

The performance of all integrators for all four scenes can be seen in tables Tab.5.2 (for the camera position simulation) and Tab.5.3 (for the mirror setup simulation). In those tables, the image quality of the renderings is categorized in four categories:

- No result (NR): The integrator was not capable to render an image, but crashed or only rendered a black image.
- Artifacts (A): The integrator created curious artifacts or was missing illumination effects of the scene.
- Insufficient quality (1): The output picture had clearly visible quality problems, like heavy noise or fireflies.
- Satisfactory) (2): Besides small quality problems, the image quality was otherwise satisfactory.

- High Quality (3): The image quality of the integrator was very good, or suffered only from very small problems (like slight noise).

Fig.5.6 shows an overlook over examples for those four categories.

Integrator \ Scene	Matpreview	Cornell Box	myBox	Balls
Direct Illumination	A	3	NR	A
Path Tracer(PT)	3	2	NR	1
Vol. PT (Simple)	3	2	NR	1
Vol. PT (Extended)	3	2	NR	1
Adj. Particle Tracer	1	NR	NR	1
Photon Mapper	2	2	NR	1
Bidirectional PT	2	NR	1	1
PSSMLT	2	NR	1	1
PSMLT	2	NR	A	A
Energy Redistridbution PT	2	NR	1	1

Table 5.2: The objective render quality for every integrator in every scene for the simulation with multiple camera positions.

Integrator \ Scene	Matpreview	Cornell Box	myBox	Balls
Direct Illumination	NR	NR	NR	NR
Path Tracer(PT)	3	3	NR	1
Vol. PT (Simple)	3	2	NR	1
Vol. PT (Extended)	3	3	NR	1
Adj. Particle Tracer	NR	NR	NR	NR
Photon Mapper	3	2	NR	1
Bidirectional PT	3	NR	2	1
PSSMLT	3	NR	2	1
PSMLT	3	NR	A	A
Energy Redistridbution PT	1	NR	1	1

Table 5.3: The objective render quality for every integrator in every scene for the mirror setup simulation. The render parameters were set to match the render time of nine pictures with the first simulation.

Multiple conclusion can be drawn from the results presented in Tab.5.2 and Tab.5.3. Multiple integrators are not compatible with subsurface scattering in the current ver-

sion of Mitsuba: The Adjoint Partcile Tracer, the bidirectional Path Tracer, the Energy Redistribution Path Tracer and both MLT methods.

The Adjoint Particle Tracer never resulted in satisfactory images for the camera simulation and was not working at all for the mirror simulation. The Energy Redisitrbution Path Tracer was also never able to perform well. Therefore the usage of these techniques can not be recommended.

Direct Illumination can produce good (and fast) results, but only if scenes don't contain indirect illumination (the Cornell Box is an example for such a scene). For simple scenes containing indirect illumination the normal Path Tracer proved to be a good choice.

Within the time restrictions of the simulations no integrator was capable to render the myBox and Balls scenes with sufficient quality.

The Volumetric Path Tracer methods never improved the image quality of the standard Path Tracer. But because of the overhead, their run time was always worse. For scenes without participating media, the volumetric approaches do not provide any benefit.

5.2.2 Results for the Different Camera Positions Simulation

When setting the number of images higher, no errors or crashes occurred. All integrators, that were successful in rendering one picture were also capable of rendering four or nine pictures.

Also, the the picture quality did not suffer from an increased image number. All pictures taken in a single run were of the same quality. That means all integrators are in general capable of acquiring light fields with multiple camera positions (if the integrator is compatible with the respective scene of course).

For the camera position simulation the time it took to render a single camera position was compared with the time it took to render four or nine pictures. The resulting factors can be found in table Tab.5.4.

	Minimum	Average	Maximum
Factors for four pictures	2.94	3.89	4.65
Factors for nine pictures	7.65	8.92	10.27

Table 5.4: The factors the time for the runs with four or nine pictures was slower, than the runs with a single picture.

The time it took to render four pictures was on average 3.89 times longer than for a single picture, while nine pictures required 8.92 times more time. Small fluctuations in

the numbers can be explained by the slight changes in the scenes, that were introduced by displacing the camera positions or the influence of random elements in the different integrators. A single simulation run with 100 pictures was conducted and the render time was 99,7 times higher than for the single picture case. One can gather from that, that Mitsuba is scaling perfectly with the number of pictures and that a light field constructed from x pictures will always roughly take x times the time it takes to render a single picture[5].

5.2.3 Results of the Second Simulation: Mirror Setup

The Direct Illumination and Adjoint Particle Tracer are not compatible with the mirror setup. The reason Direct Illumination does not work, is that it can not render indirect illumination effects. When watching a scene trough a mirror, all the camera sees is the result of indirect illumination. The Adjoint Particle Tracer traces particles from the light source trough the scene and tries to find a path to the camera. An additional mirror in the scene, makes finding the camera more difficult for the particles. In all executed simulation runs no particle reached the camera, always resulting in black output images. All other integrators worked with the mirror setup.

5.2.4 Comparison of Both Approaches

When running the mirror simulation with the same parameters as for the camera simulation, the image quality was always worse for the mirror setup, an explicit example for that can be seen in Fig.5.7. This behavior was to be expected, because an additional mirror makes scenes more difficult to render.

Raising the quality settings of the integrators for the second step of the mirror simulation, resulted in some cases in improved image quality. In the Matpreview scene both MLT methods, the bidirectional Path Tracer and the Photon Mapper achieved higher image quality. For the Cornell Box scene the Path Tracer and extended Volumetric Path Tracer profited from the additional render time. For the hard to render scenes, however the increased settings were still not high enough to increase image quality significantly.

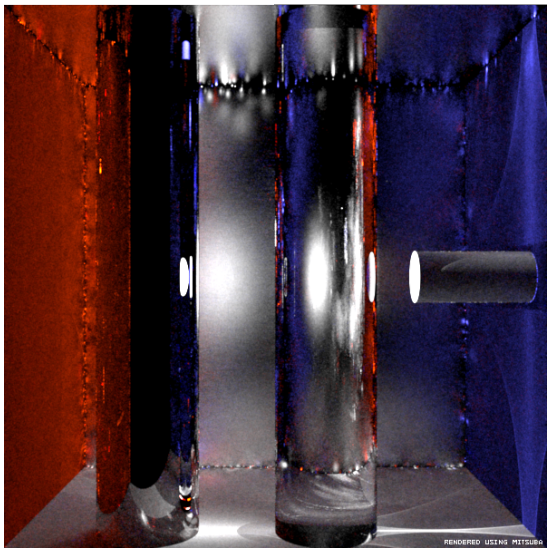
[5] If the camera displacement is not too big to change the rendering cost of the individual pictures too much.

5.2.5 Simulation Parameters

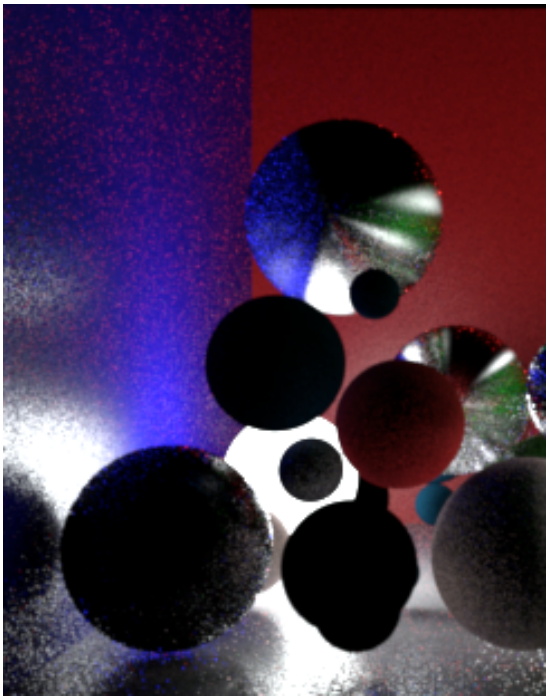
As stated, the Mitsuba default settings were used as baseline settings for every simulation run. These setting proved to be a good compromise between image quality and run time and where mostly unchanged. Primarily the number of samples for the sampler had to be adjusted to adapt the results to the desired conditions.

5.3 Discussion

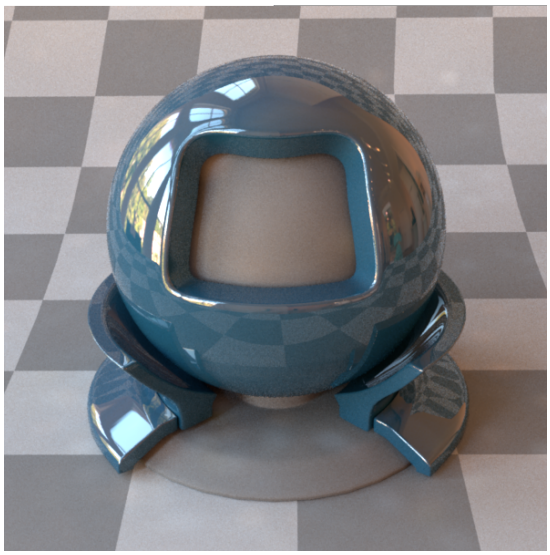
The simulations showed, that for easy scenes, most Mitsuba integrators are suitable for acquiring of light fields. For difficult scenes however, the simulation with Mitsubas methods can not deliver sufficient light field quality, without being too time consuming. So unless very high computational power is available, recording high quality and dense light fields for complex scenes is not (yet) manageable. In such cases, the benefit from virtual simulations could be too small and real world recording of the light field might be the cheaper variant. For small simulations with the goal to compare acquisition techniques for easy examples, Mitsuba can be recommended. Depending on the type of scene at hand, the integrator has to be chosen accordingly, because not all techniques are equally applicable for every type of scene.



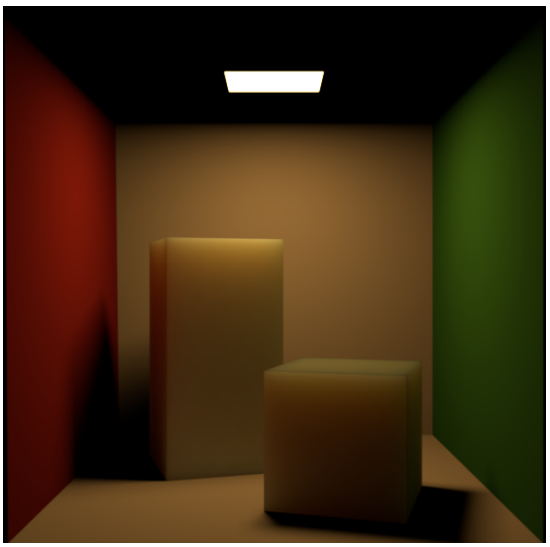
(a)



(b)

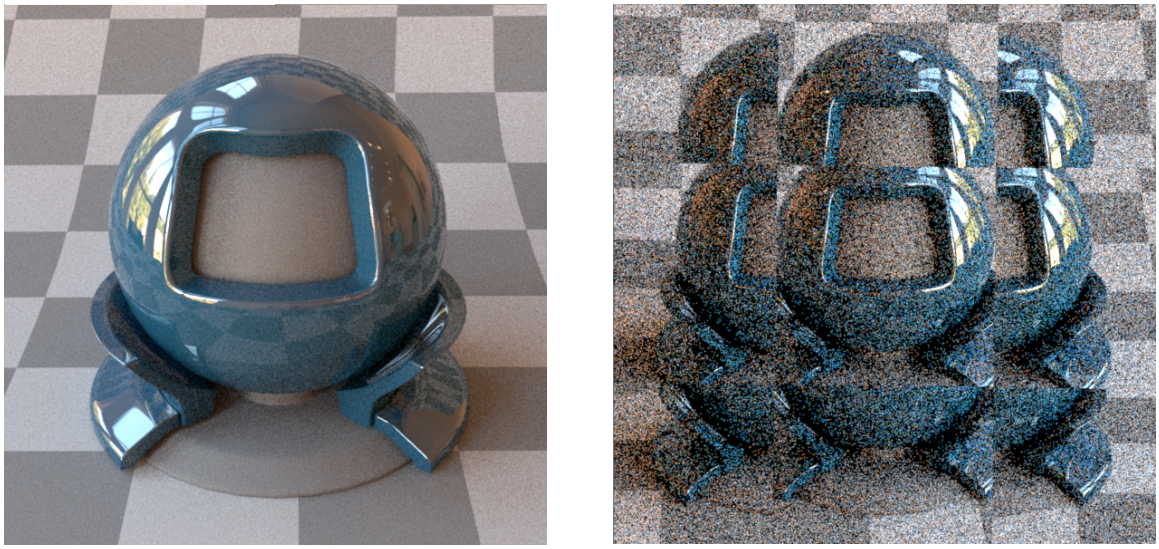


(c)



(d)

Figure 5.6: Example for the four quality categories:
(a) Category A (Artifacts): Rendering of the myBox-scene with the PSMLT integrator. At the corners of the box, heavy artifacts are visible.
(b) Category 1 (Insufficient quality): Rendering of the Balls-scene with the Adjoint Partcle Tracer. Heavy Noise and fireflies are visible.
(c) Category 2 (Satisfactory): Rendering of the Matpreview-scene with the Photon Mapper. The image quality is good, except for the blurry spots on the chessboard and little noise at the right top side of the ball.
(d) Category 3 (High Quality) : Flawless rendering of the Cornell Box-scene with the Direct Illumination-technique. There is no noise or other artifacts visible.



(a) **(b)**

Figure 5.7: Two renderings of the Matpreview-test scene with the Energy Redistribution Path Tracing, both times with the same integrator settings. Picture (a) comes from the first simulation type and (b) from the mirror simulation setup. While the first picture only suffers from slight noise, the second one is clearly affected by heavy noise.

6 Rendering of Light Field Data with Mitsuba

In this chapter the light field emitter, that was introduced in chapter 4.2, will be used to render light field data.

The emitter will be placed in two different simple test scenes. Every time an camera ray hits the emitter it looks up the nearest neighbor of that ray in the rays from the light field. For this task, all of Mitsubas integrators will be used and evaluated.

6.1 Simulation Procedure and Restrictions

It was decided, that no real world light field data was used in this tests. The reason for that is that no suitable light field data was available and the level of control over synthetic light field data is higher. With the help of the simulation framework (from chapter 4.1), a custom light field was generated. An image of this light field can be seen in Fig.6.1.

The scene shows an red ball with diffuse reflectance properties, that hovers between the camera and a checkerboard. This scene meets the assumptions that were defined for the emitter, that the camera is located on the x,y-plane, has no rotation and looks in positive z-direction (see chapter 4.2)[1].

The light field was recorded once by rendering the scene from the standard central camera position, producing a light field containing just one image (mainly as baseline value). Then four, nine, 16 and 25 image light fields were recorded, each with a small maximal camera displacements. Small meaning, that the red ball in the middle of the scene was still the main content of the pictures. Each image of the light field was created with the Path Tracer integrator with high quality settings and a 1000x1000 resolution,

[1] For the following tests, the contents shown in the light field are irrelevant, as long, as it meets these assumptions. This is because all the emitter does is access the image data of 2D pictures. What is shown on these pictures is not important.

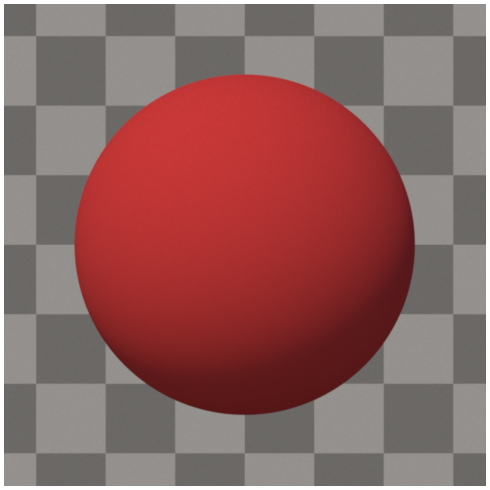


Figure 6.1: Scene for the redBall scene, that was used to generate light fields for the light field emitter.

resulting in very high quality pictures. This way it can be guaranteed, that artifacts later on do not accrue because of a insufficient image quality of the light field.

This light field data was integrated into two very simple test scenes, which will be introduced in the next section.

6.1.1 Scenes

The first scene only features an plane rectangle, that is positioned in front of the camera. The rectangle is defined as an light field emitter. Rays that are shot from the camera hit the emitter directly. This way renderings of the scene should be equal to the original light field recordings.

As a second test case, the same scene was used, but this time a glass ball was placed between the camera and the rectangle. In this scene the camera rays do not hit the emitter directly, but rather get refracted inside the glass ball first. This way it can be investigated if the emitter class and Mitsuba are capable of rendering light fields indirectly (for example trough a lens in front of the camera).

6.1.2 Integrators

Just like in chapter 5, all of Mitsubas integrators were examined, excluding Ambient Occlusion, the Progressive Photon Mapper, the Progressive Stochastic Photon Mapper and VPL.

Ambient Occlusion, because it is not able to generate realistic looking pictures, the two Photon Mapper variants, because they are progressive and VPL, because it was not working on the test system.

6.1.3 Procedure

The tests were performed on the same system, as the simulations in chapter 5.1.3.

Every included integrator was tested in 20 different ways: Once for every number of light field images (one, four, nine, 16 and 25), for the two different scenes and from two different camera positions. The first camera position is in the front center the rectangle. In a second step, the camera was moved slightly, in a way, so that the new camera position was not directly featured in the light field acquisition. This way the camera rays are not exactly the same as the light fields rays, but rather have to be approximated by their neighbors (see chapter 4.2).

Every integrator was tested with its standard settings and 16 samples per pixel. In case that the quality of the rendering result was too low or the rendering times too high, the samples per pixel setting was adapted.

For each integrator and every test run it was recorded, if the rendering was successful, how long it took and what quality the resulting picture had.

6.2 Results

This section will summarize the results of the different test runs. Some integrators unfortunately did not work with the light field in the test runs. Chapter 6.2.1 will discuss this circumstance. The overall image quality of the renderings will be examined in chapter 6.2.2. In chapter 6.2.3, it will be investigated how changing the image number and density of the source light field changed the outcome of the rendering. An important property when rendering is the required render time. The time aspect of the test are explored in chapter 6.2.4. Concluding this section are a look on how changing the camera positions changed the render results (chapter 6.2.5) and what the difference between the two different test scenes was (chapter 6.2.6).

6.2.1 Incompatible Integrators

Unfortunately multiple integrator were not able to render the light fields.

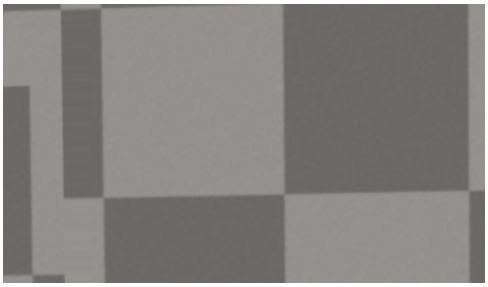
The Bidirectional Path Tracer and the Adjoint Particle Tracer only produced black images. The Primary Sample Space MLT, Path Space MLT and Energy Redistribution Path Tracer all crashed with the error message "Error: The rendering Job did not complete successfully" before the rendering was finished [2].

For that reason these integrators could not be included in the evaluation. In the following sections only the remaining five integrators (Direct Illumination, Path Tracer and the two Volumetric Path Tracer variants) are discussed.

6.2.2 Image Quality

For dense parts of the light field (in most cases the ball in the center), the image quality of the rendering was good, even with only 16 samples per pixel.

But if the light field was rendered for not dense areas, like the edge of the checkerboard, two types of artifacts occurred, which both can be seen in Fig.6.2.



(a) Corruptions in the checkerboard

(b) Streaks in the checkerboard

Figure 6.2: The two different type of image artifacts, that can be found in renderings of non dense areas of the light fields, for example the left edge of the checkerboard when the camera was moved.

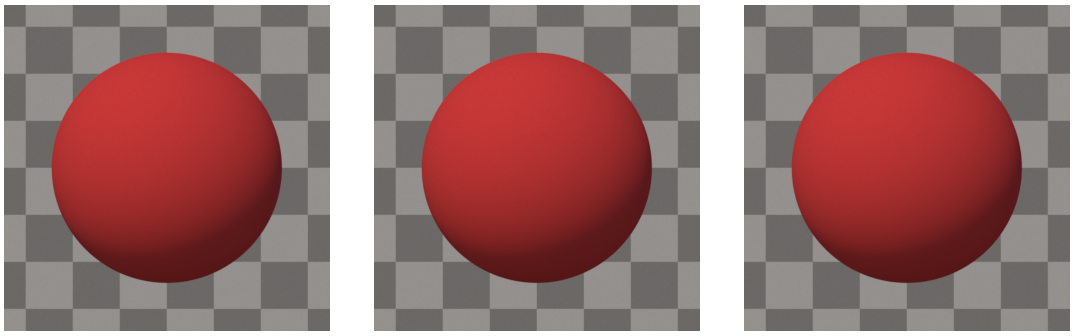
There were no noticeable differences in image quality between the five integrators. If image artifacts occurred, they applied to all integrators in the same way. Other than these streaks or corruptions, which both are caused by the non density of the light field

[2]The reason for that behavior is unknown. A bug inside the light field emitter could be the cause. Unfortunately this problem could not be resolved in time.

at those areas, the image quality was good for all five integrators. The images were sharp and contained no noise or aliasing.

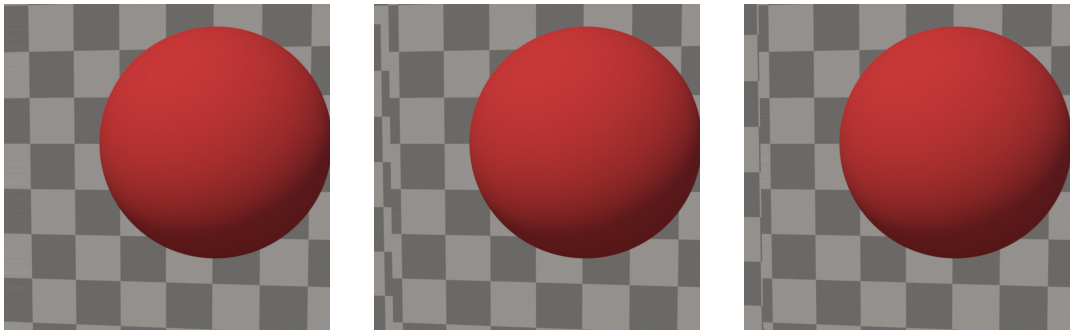
6.2.3 Influence of Number of Light Field Images

The image quality did not change for the cases of one, nine or 25 light field images (see Fig.6.3 and Fig.6.4 for examples), only the types of visible artifacts changed (caused by a different density for the hard to render parts of the light field). The results were again the same for every working integrator.



(a) Light field with one image **(b)** Light field with nine images **(c)** Light field with 25 images

Figure 6.3: The Different results for different image numbers with an unchanged camera position. The resulting images are exactly the same.

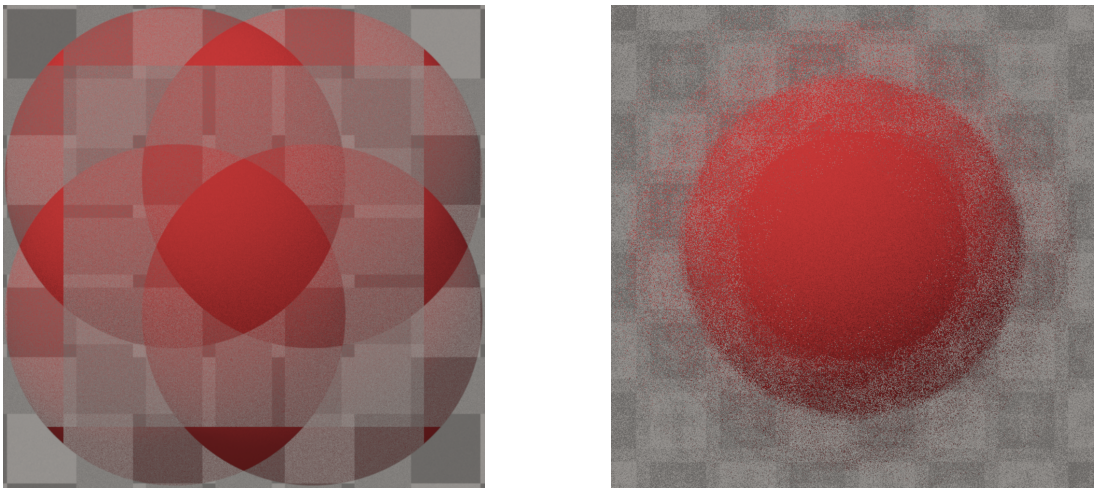


(a) Light field with one image **(b)** Light field with nine images **(c)** Light field with 25 images

Figure 6.4: The Different results for different image numbers for a changed camera position. The quality of the rendered ball was the same, only the artifacts on the left image size changed.

Also the render time did not change in this cases, if the camera was positioned at the standard position (see 6.2.5 for more details on the other case).

Using four or 16 images changed the results dramatically. Heavy image artifacts occurred and the render times were significantly higher, than for the other cases. Changing the integrator did not change this behavior. Two example pictures for the image artifacts can be seen in Fig.6.5.



(a) Light field with four images

(b) Light field with 16 images

Figure 6.5: Heavy artifacts occurred for the cases with 2x2 (a) and 4x4 (b) light field images. Both images were created with the Path Tracer and both times the samples per pixel number was set to 2 (instead of 16), because rendering times were too high. Even for this low number of samples, the rendering time was eight (a) and six (b) minutes.

Both the four image and the 16 image light field are not dense in the middle of the scene, because they feature an even number of image per row. No image is available for the middle position (which represents the standard camera position in the test scene). This is the reason, the resulting images and render times were so bad: In the crucial areas of the picture, which is visible the most (the middle with the red ball) are underrepresented in the light field. Therefore no useful neighbors for the camera rays could be found and the search takes longer. This leads to the assumption, that the nearest neighbor search with libAnn is a bottleneck in the emitter. This suspicion substantiates in the following section, which investigates the render times in more detail.

6.2.4 Render Time

The Direct Illumination, Path Tracer and Volumetric Path Tracer always needed roughly the same time (for an light field with nine pictures and an unchanged camera position 27 and 28 seconds). The Photon Mapper was slightly slower than the other methods, but not by a large margin (for the same example case 30 seconds). For an unchanged camera positions, the render time did not change for an increasing image number.

Lowering the number of samples per pixel causes lower render times, but introduces noise to the picture. A sample amount of 16 proved to be sufficient.

While rendering, it was noticeable that the rendering of areas of the light fields, that were not dense enough required more time to render, than areas with enough samples. Searching the nearest neighbor for such rays requires more effort, which hardens the suspicion, that libAnn causes performance losses for non dense areas.

Note: The time it took to load a scene into Mitsuba was not included in this evaluation, because it is dependent on how much images the loaded light field includes, Mitsuba and the emitter. It does not represent the quality or the capability of the single integrators to render light fields. But keep in mind that loading a scene for many high resolution images can take quite a long time (multiple minutes).

6.2.5 Changing the Camera Position

Changing the camera position slightly did negatively influence the rendering times. This negative effect was amplified when more images were included in the light field. Table 6.1 shows an example for this.

Number of images in the light field	Render time for standard position)	Render time for changed position)
1	27,5s	31s
9	28s	53s
25	28s	90s

Table 6.1: Some example times for the rendering of the Light Field with and without a moved camera position. The render times for the Path Tracer with one, nine and 25 image light fields are shown.

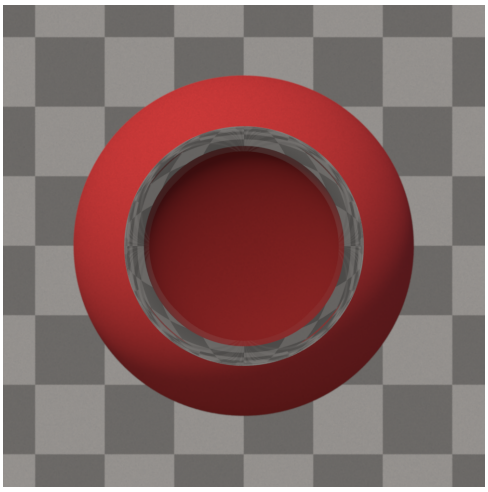
The reason for that is simple: By changing the camera position, areas of the light field that are not dense get visible, therefor increasing the number of hard to calculate pixel values. Increasing the number of the light field pixels increases the size of the kdTree,

which contains all light field rays and it takes more time to find a nearest neighbor in such a big tree. As established, the reason for that behavior is libAnn.

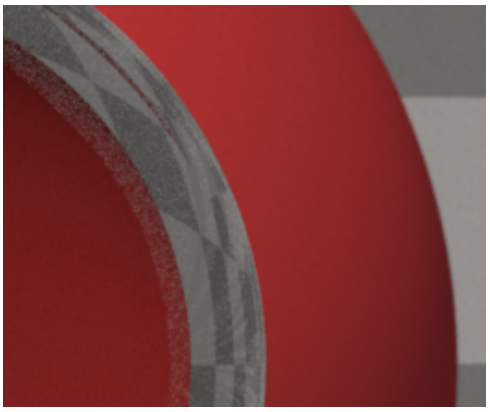
6.2.6 Difference between Scenes

The quality and the render times were exactly the same when introducing an additional glass ball into the scene.

The only difference was, that only the Photon Mapper was able to render the glass ball without noise. This circumstance can be seen in Fig.6.6.



(a) The rendering of the glass ball without noise.



(b) Example for noise in the glass ball.

Figure 6.6: a) The ball scene, that was rendered with the photon mapper and a nine picture light field. There is no noise visible in the glass ball.
b) An enhanced view of the problem that all other integrators had with the scene: Visible noise in the glass ball.

Other than that, the glass ball did not influence the rendering time or quality of the light field.

6.3 Discussion

The tests showed, that the choice of a Mitsuba integrator does not matter (again only focusing on the integrators, that worked with the light field emitter). All working integrators performed nearly the same, resulting in equal render times and image quality.

Only the Photon Mapper was a little bit slower than the four other techniques, but handled caustics in the second scene better.

The limiting factors of the test were the lack of density in the examined light fields and the nearest neighbor search with libAnn. Rendering non dense areas in the light field caused different image artifacts. Unforutunally bigger and more dense light fields could not be tested on the used test system, because the lack of RAM.

The libAnn library, which is used in the light field emitter was the performance bottleneck in the tests. Every time the search for a nearest neighbor of rays was not easy (in non dense areas), the performance decreased dramatically.

7 Summary and Outlook

This chapter will give an quick summary of this works content. The summary is then followed by an overview over possible future work.

As stated in chapter 2, light fields provide many interesting opportunities for the post processing of images, but their acquisition can be difficult and complex and comparison of different acquisition techniques is difficult. Virtual simulations are the best way for such comparisons. Before light fields can be used, they have to be rendered, which is no trivial task. For that reason chapter 3 introduced the fundamentals of rendering in general and the rendering of light fields. The open source renderer Mitsuba was also introduced there, along with his multiple rendering techniques.

Subsequently the main goal of the thesis was to examine if and how Mitsuba can be used for simulating different light field acquisition techniques and rendering. Because Mitsuba doe not support light fields, a custom software framework had to be implemented. This framework and its possibilities were introduced in chapter 4. It contains two parts: A simulation framework for two different light field simulations and an emitter for light field rendering. The usage of these two parts was disused in chapters 5 and 6.

The simulations of light field acquisition were conducted for four different test scenes. They all had different properties and degree of difficulty. Mitsubas integrators were evaluated on how fast they could render the scene, if they could render the scene at all and how high the quality of the resulting images was.

The same procedure was repeated in chapter 6 for light field rendering. All integrators were compared with each other under the same aspects. They were used to render light fields for two different scenes and afterwards the performance of all integrators was compared.

Outlook

In addition to the topices that were examined within the limits of this thesis, some additional issues could get investigated.

There are many open questions to answers for simulation of light field acquisition: In chapter 5 light fields were generated by changing the positions of a scenes camera. The amount the camera was moved was always the same. Future work could examine what the optimal displacement between cameras is, to produce the highest quality light field. The number of different camera position was also very limited in those simulations. An interesting aspect, that was not covered in this thesis is how many individual images are needed to create an sufficient dense light field. The simulations were conducted to reproduce real world light field acquisition. A future thesis or paper could explore if there are differences between synthetic generated light fields and real world versions of the same scene. The light fields in this thesis were only generated with two different techniques, time sequential capturing and with the help of an mirror array. As stated in chapter 2.3 there are many more different methods of acquiring light field data. As subject for a future work, these approaches could also be reproduced and compared in virtual simulations with the help of Mitsuba.

The lightfield emitter that was introduced in chapter 4.2 and applied in chapter 6 was unfortunately defective. In additional work the emitter could be fixed and expanded. For example, it used libAnn for nearest neighbor search, which proved to be a bottleneck for performance. There are other frameworks for nearest neighbor search, so libAnn could be replaced by another application. The emitter also caused crashes with multiple rendering techniques, so that they could not be included in the tests.

All simulations and test of chapter 5 and chapter 6 were only conducted on the personal computer of the author. As a result, the performance and scope of these test was often limited by the used hardware. The same simulations and test could in the future be conducted on high performance hardware, like a high performance cluster. This way the full potential of Mitsuba, its integrators and the used techniques could be explored. For example denser light fields could be generated or rendered or more difficult scenes could be used to create light fields.

8 Conslusion

This thesis showed that Mitsuba can generally be used for light field acquisition and the rendering of light field data.

For easy to render scenes, most of Mitsubas integrators are at least able to create light fields with sufficient high quality images in a reasonable time. This holds up for both of the two tested simulation types. The rendering time is linearly dependent with the number of rendered images. Differences between the individual integrators are strongly dependent on the used scene.

However, difficult to render scenes require too much render time for acceptable image quality on consumer hardware. On such hardware, Mitsuba is only suited for small simulations or quick test cases of light field acquisition. But the established differences between the integrators should also apply to simulations on more powerful systems or clusters.

Mitsuba is well suited for light field rendering. All integrators (that worked with the light field emitter) are able to render adequate dense areas of light fields in a very short time in an flawless quality. No quality differences between the light fields source images and the rendered image were noticeable. The tests also show that performance and image quality is only insufficient in areas where the light fields density is too low. Dense light fields can be rendered fast and in high quality with Mitsuba and there is no meaningful difference in performance or render quality between its integrators.

Index

Bibliography

[AB91] E. H. Adelson, J. R. Bergen. “The Plenoptic Function and the Elements of Early Vision.” In: *Computational Models of Visual Processing*. MIT Press, 1991, pp. 3–20 (cit. on pp. 13, 17).

[AVR10] A. Agrawal, A. Veeraraghavan, R. Raskar. “Reinterpretable Imager: Towards Variable Post-Capture Space, Angle and Time Resolution in Photography.” In: *Computer Graphics Forum* 29.2 (2010), pp. 763–772. URL: http://dx.doi.org/10.1111/j.1467-8659.2009.01646.x (cit. on p. 23).

[AW92] E. H. Adelson, J. Y. A. Wang. “Single Lens Stereo with a Plenoptic Camera.” In: *IEEE Trans. Pattern Anal. Mach. Intell.* 14.2 (Feb. 1992), pp. 99–106. URL: http://dx.doi.org/10.1109/34.121783 (cit. on pp. 22, 24).

[BBM+01] C. Buehler, M. Bosse, L. McMillan, S. Gortler, M. Cohen. “Unstructured Lumigraph Rendering.” In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’01. New York, NY, USA: ACM, 2001, pp. 425–432. URL: http://doi.acm.org/10.1145/383259.383309 (cit. on p. 24).

[Ben75] J. L. Bentley. “Multidimensional Binary Search Trees Used for Associative Searching.” In: *Commun. ACM* 18.9 (Sept. 1975), pp. 509–517. URL: http://doi.acm.org/10.1145/361002.361007 (cit. on p. 49).

[BF12] T. E. Bishop, P. Favaro. “The Light Field Camera: Extended Depth of Field, Aliasing, and Superresolution.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.5 (May 2012), pp. 972–986 (cit. on p. 22).

[Bra65] R. Bracewell. “The fourier transform and its applications.” In: *New York* 5 (1965) (cit. on pp. 32, 33).

[CPW10] C.-h. Chen, L.-F. Pau, P. S.-p. Wang. *Handbook of pattern recognition and computer vision*. Vol. 27. World Scientific, 2010 (cit. on p. 21).

[Cro77] F. C. Crow. “The Aliasing Problem in Computer-generated Shaded Images.” In: *Commun. ACM* 20.11 (Nov. 1977), pp. 799–805. URL: http://doi.acm.org/10.1145/359863.359869 (cit. on p. 33).

[CTE05] D. Cline, J. Talbot, P. Egbert. “Energy Redistribution Path Tracing.” In: *ACM Trans. Graph.* 24.3 (July 2005), pp. 1186–1195. URL: http://doi.acm.org/10.1145/1073204.1073330 (cit. on p. 43).

[DHT+00] P. Debevec, T. Hawkins, C. Tchou, H.-P. Duiker, W. Sarokin, M. Sagar. “Acquiring the Reflectance Field of a Human Face.” In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 145–156. URL: http://dx.doi.org/10.1145/344779.344855 (cit. on pp. 21, 24).

[FKR12] M. Fuchs, M. Kaechele, S. Rusinkiewicz. “Design and Fabrication of Faceted Mirror Arrays for Light Field Capture.” In: *VMV 2012: Vision, Modeling & Visualization*. Eurographics Association, 2012, pp. 1–8. URL: http://dx.doi.org/10.2312/PE/VMV/VMV12/001-008 (cit. on p. 23).

[GGHS03] M. Goesele, X. Granier, W. Heidrich, H.-P. Seidel. “Accurate Light Source Acquisition and Rendering.” In: *ACM SIGGRAPH 2003 Papers*. SIGGRAPH ’03. San Diego, California: ACM, 2003, pp. 621–630. URL: http://doi.acm.org/10.1145/1201775.882316 (cit. on p. 25).

[GGSC96] S. J. Gortler, R. Grzeszczuk, R. Szeliski, M. F. Cohen. “The Lumigraph.” In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’96. New York, NY, USA: ACM, 1996, pp. 43–54. URL: http://doi.acm.org/10.1145/237170.237200 (cit. on pp. 13, 18, 22).

[GL10] T. Georgiev, A. Lumsdaine. “Rich image capture with plenoptic cameras.” In: *Computational Photography (ICCP), 2010 IEEE International Conference on*. Mar. 2010, pp. 1–8 (cit. on pp. 22, 23).

[Gla14] A. S. Glassner. *Principles of Digital Image Synthesis*. Morgan Kaufmann, 2014 (cit. on pp. 32, 33).

[GZC+06] T. Georgeiv, K. C. Zheng, B. Curless, D. Salesin, S. Nayar, C. Intwala. “Spatio-angular Resolution Tradeoffs in Integral Photography.” In: *Proceedings of the 17th Eurographics Conference on Rendering Techniques*. EGSR ’06. Nicosia, Cyprus: Eurographics Association, 2006, pp. 263–272. URL: http://dx.doi.org/10.2312/EGWR/EGSR06/263-272 (cit. on pp. 23, 26).

[HJ09] T. Hachisuka, H. W. Jensen. “Stochastic Progressive Photon Mapping.” In: *ACM SIGGRAPH Asia 2009 Papers*. SIGGRAPH Asia ’09. Yokohama, Japan: ACM, 2009, 141:1–141:8. URL: http://doi.acm.org/10.1145/1661412.1618487 (cit. on p. 41).

[HK93] P. Hanrahan, W. Krueger. “Reflection from Layered Surfaces Due to Subsurface Scattering.” In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’93. Anaheim, CA: ACM, 1993, pp. 165–174. URL: http://doi.acm.org/10.1145/166117.166139 (cit. on p. 31).

[HKSS98] W. Heidrich, J. Kautz, P. Slusallek, H.-P. Seidel. “Canned Lightsources.” English. In: *Rendering Techniques ’98*. Ed. by G. Drettakis, N. Max. Eurographics. Springer Vienna, 1998, pp. 293–300. URL: http://dx.doi.org/10.1007/978-3-7091-6453-2_27 (cit. on p. 25).

[HOJ08] T. Hachisuka, S. Ogaki, H. W. Jensen. “Progressive Photon Mapping.” In: *ACM Trans. Graph.* 27.5 (Dec. 2008), 130:1–130:8. URL: http://doi.acm.org/10.1145/1409060.1409083 (cit. on p. 41).

[ICG86] D. S. Immel, M. F. Cohen, D. P. Greenberg. “A Radiosity Method for Non-diffuse Environments.” In: *SIGGRAPH Comput. Graph.* 20.4 (Aug. 1986), pp. 133–142. URL: http://doi.acm.org/10.1145/15886.15901 (cit. on p. 28).

[ISG+08] I. Ihrke, T. Stich, H. Gottschlich, M. Magnor, H.-P. Seidel. “Fast Incident Light Field Acquisition and Rendering.” In: *Journal of WSCG* (Feb. 2008) (cit. on p. 21).

[Jak14] W. Jakob. *Mitsuba Documentation*. 0.5.0. Feb. 2014 (cit. on pp. 30, 35, 36, 43).

[Jen04] H. W. Jensen. “A Practical Guide to Global Illumination Using Ray Tracing and Photon Mapping.” In: *ACM SIGGRAPH 2004 Course Notes*. SIGGRAPH ’04. Los Angeles, CA: ACM, 2004. URL: http://doi.acm.org/10.1145/1103900.1103920 (cit. on p. 40).

[KA04] K. Khoshelham, A. Azizi. *Kaiser Filter For Antialiasing In Digital Photogrammetry*. 2004 (cit. on p. 35).

[Kaj86] J. T. Kajiya. “The Rendering Equation.” In: *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’86. New York, NY, USA: ACM, 1986, pp. 143–150. URL: http://doi.acm.org/10.1145/15922.15902 (cit. on p. 28).

[Kel97] A. Keller. “Instant Radiosity.” In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 49–56. URL: http://dx.doi.org/10.1145/258734.258769 (cit. on p. 41).

[KK02] T. Kollig, A. Keller. “Efficient multidimensional sampling.” In: *Computer Graphics Forum*. Vol. 21. 3. Wiley Online Library. 2002, pp. 557–563 (cit. on p. 32).

[KS01] C. Kelemen, L. Szirmay-Kalos. *Simple and Robust Mutation Strategy for Metropolis Light Transport Algorithm*. Tech. rep. TR-186-2-01-18. human contact: technical-report@cg.tuwien.ac.at. Favoritenstrasse 9-11/186, A-1040 Vienna, Austria: Institute of Computer Graphics and Algorithms, Vienna University of Technology, July 2001. URL: https://www.cg.tuwien.ac.at/research/publications/2001/Szirmay-2001-METR/ (cit. on p. 42).

[LH96] M. Levoy, P. Hanrahan. "Light Field Rendering." In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. New York, NY, USA: ACM, 1996, pp. 31–42. URL: http://doi.acm.org/10.1145/237170.237199 (cit. on pp. 13, 18, 21, 24).

[LLC07] C.-K. Liang, G. Liu, H. Chen. "Light Field Acquisition using Programmable Aperture Camera." In: *Image Processing, 2007. ICIP 2007. IEEE International Conference on*. Vol. 5. Sept. 2007, (cit. on p. 24).

[LW93] E. P. Lafortune, Y. D. Willems. "Bi-Directional Path Tracing." In: *Proceedings of third international conference on computational graphics and visualization techniques*. 1993, pp. 145–153 (cit. on p. 41).

[LW96] E. P. Lafortune, Y. D. Willems. "Rendering Participating Media with Bidirectional Path Tracing." In: *Proceedings of the Eurographics Workshop on Rendering Techniques '96*. Porto, Portugal: Springer-Verlag, 1996, pp. 91–100. URL: http://dl.acm.org/citation.cfm?id=275458.275468 (cit. on p. 40).

[MN88] D. P. Mitchell, A. N. Netravali. "Reconstruction Filters in Computergraphics." In: *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '88. New York, NY, USA: ACM, 1988, pp. 221–228. URL: http://doi.acm.org/10.1145/54852.378514 (cit. on p. 33).

[MRR+53] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, E. Teller. "Equation of State Calculations by Fast Computing Machines." In: *The Journal of Chemical Physics* (1953). URL: http://dx.doi.org/10.1063/1.1699114 (cit. on p. 31).

[NLB+05] R. Ng, M. Levoy, M. Brf, G. Duval, M. Horowitz, P. Hanrahan. *Light Field Photography with a Hand-held Plenoptic Camera*. 2005 (cit. on pp. 22, 24).

[OA00] A. Owen, Y. Z. Associate. "Safe and Effective Importance Sampling." In: *Journal of the American Statistical Association* 95.449 (2000), pp. 135–143. eprint: http://amstat.tandfonline.com/doi/pdf/10.1080/01621459.2000.10473909. URL: http://amstat.tandfonline.com/doi/abs/10.1080/01621459.2000.10473909 (cit. on p. 32).

[PH10] M. Pharr, G. Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*. 2nd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010 (cit. on pp. 33, 34, 39).

[SCK08] H.-Y. Shum, S.-C. Chan, S. B. Kang. *Image-based rendering*. Springer Science & Business Media, 2008 (cit. on p. 24).

[Sie15] H. Siedelmann. "Recording, compression and representation of dense light fields." eng. MA thesis. Holzgartenstr. 16, 70174 Stuttgart: Universität Stuttgart, 2015. URL: http://elib.uni-stuttgart.de/opus/volltexte/2015/9926 (cit. on p. 25).

[Sze10] R. Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010 (cit. on p. 21).

[TAHL07] E.-V. Talvala, A. Adams, M. Horowitz, M. Levoy. "Veiling Glare in High Dynamic Range Imaging." In: *ACM SIGGRAPH 2007 Papers*. SIGGRAPH '07. San Diego, California: ACM, 2007. URL: http://doi.acm.org/10.1145/1275808.1276424 (cit. on p. 25).

[TARV10] Y. Taguchi, A. Agrawal, S. Ramalingam, A. Veeraraghavan. *Axial Light Field for Curved Mirrors: Reflect Your Perspective, Widen Your View*. CVPR 2010. 2010 (cit. on p. 23).

[UWH+03] J. Unger, A. Wenger, T. Hawkins, A. Gardner, P. Debevec. "Capturing and Rendering with Incident Light Fields." In: *Proceedings of the 14th Eurographics Workshop on Rendering*. EGRW '03. Leuven, Belgium: Eurographics Association, 2003, pp. 141–149. URL: http://dl.acm.org/citation.cfm?id=882404.882425 (cit. on p. 23).

[VG95] E. Veach, L. J. Guibas. "Optimally Combining Sampling Techniques for Monte Carlo Rendering." In: *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '95. New York, NY, USA: ACM, 1995, pp. 419–428. URL: http://doi.acm.org/10.1145/218380.218498 (cit. on p. 31).

[VG97] E. Veach, L. J. Guibas. "Metropolis Light Transport." In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 65–76. URL: http://dx.doi.org/10.1145/258734.258775 (cit. on p. 43).

[VRA+07] A. Veeraraghavan, R. Raskar, A. Agrawal, A. Mohan, J. Tumblin. "Dappled Photography: Mask Enhanced Cameras for Heterodyned Light Fields and Coded Aperture Refocusing." In: *ACM Trans. Graph.* 26.3 (July 2007). URL: http://doi.acm.org/10.1145/1276377.1276463 (cit. on pp. 24, 25).

[WIL+12] G. Wetzstein, I. Ihrke, D. Lanman, W. Heidrich, R. Raskar, K. Akeley. "Computational Plenoptic Imaging." In: *ACM SIGGRAPH 2012 Courses*. SIGGRAPH '12. Los Angeles, California: ACM, 2012, 11:1–11:265. URL: http://doi.acm.org/10.1145/2343483.2343494 (cit. on p. 20).

[WJV+05] B. Wilburn, N. Joshi, V. Vaish, E.-V. Talvala, E. Antunez, A. Barth, A. Adams, M. Horowitz, M. Levoy. "High Performance Imaging Using Large Camera Arrays." In: *ACM SIGGRAPH 2005 Papers*. SIGGRAPH '05. Los Angeles, California: ACM, 2005, pp. 765–776. URL: http://doi.acm.org/10.1145/1186822.1073259 (cit. on pp. 20, 24).

[WREE67] C. Wylie, G. Romney, D. Evans, A. Erdahl. "Half-tone Perspective Drawings by Computer." In: *Proceedings of the November 14-16, 1967, Fall Joint Computer Conference*. AFIPS '67 (Fall). Anaheim, California: ACM, 1967, pp. 49–58. URL: http://doi.acm.org/10.1145/1465611.1465619 (cit. on p. 28).

[YEBM02] J. C. Yang, M. Everett, C. Buehler, L. McMillan. "A Real-time Distributed Light Field Camera." In: *Proceedings of the 13th Eurographics Workshop on Rendering*. EGRW '02. Pisa, Italy: Eurographics Association, 2002, pp. 77–86. URL: http://dl.acm.org/citation.cfm?id=581896.581907 (cit. on pp. 20, 24).

All links were last followed on May 1, 2016.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature