FACULTY OF ENGINEERING
AND ARCHITECTURE

Towards Geometry-Consistent High-Dimensional Surface Light Field Modelling with Mixture Models

Stijn De Pauw
Student number: 01402287

Supervisors: Prof. dr. Peter Lambert, Prof. dr. ir. Glenn Van Wallendael
Counsellor: Ir. Martijn Courteaux

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Academic year 2019-2020

GHENT
UNIVERSITY

Preface

First and foremost I would like to express my gratitude towards my supervisor and future colleague Martijn Courteaux for his guidance and support through my thesis year. I also would like to thank Peter Lambert and Glenn Van Wallendael for offering this learning experience in the very interesting field of light fields.

I thank my loving parents who throughout the years gave up some of their freedom and dreams to support my education and hardware needs. For being there when I needed them the most, for putting up with my sh*t.

I thank my fiancée for catching me when I fall, and helping me back up again. For never stopping to believe in me, even when I doubted myself. For keeping strong together when life challenged our health.

I would like to thank my country for providing affordable education and healthcare, unlike other countries where you are on your own.

Finally, thanks to my family (Marijke, Michel, Ewout, Timo and Iebel), engineering friends (Matthias, Jonas, Jules, Marie, Niels, Victor-Louis, thesis weekend participants,...) and the university of Ghent for everything.

Abstract

Light-field technology proves itself quite useful in the world of interactive media. It allows for the much sought-after property of 6 Degrees of Freedom. There already exist various technologies for compressing and decompressing lightfields. An emerging technique is the Steered Mixture-of-Experts (SMoE) framework; a method which permits random access of light-field data. Currently, the SMoE framework models a light field on a planar structure. Although such structure provides a great sense of depth, there is no true imersiveness.

The goal of this dissertation is to improve the immersiveness of light fields in the context of SMoE. To this end, several options are research, with surface light fields being the predominant option in terms of immersiveness. To further assess the abilities of surface light fields, the necessary adaptations to model surface light fields with SMoE are researched, followed by the implementation of a novel method for preprocessing light-field data for SMoE. This dissertation contributes a novel light field renderer to primarily evaluate the implementation, but also to experiment with various (surface) light fields.

Researching other types of light fields such as surface light fields requires light-field data to be modelled. This dissertation also contributes a novel method for generating light-field data sets from more than a planar configuration.

We found that with the help of the developed tools, surface light fields certainly are a great candidate for solving the problem of immersiveness. The full implementation of surface light fields in the SMoE framework is currently unfinished; to continue future work, the implementation must be completed.

Towards Geometry-Consistent High-Dimensional Surface Light Field Modelling with Mixture Models

Stijn De Pauw, UGent
Ghent, Belgium
Smidpauw.DePauw@UGent.be

***Abstract*—Light-field technology proves itself quite useful in the world of interactive media. It allows for the much sought-after property of *6 Degrees of Freedom*. There already exist various technologies for compressing and decompressing light fields. An emerging technique is the Steered Mixture-of-Experts (SMoE) framework; a method which permits random access of light-field data. Currently, the SMoE framework models a light field on a planar structure. Although such structure provides a great sense of depth, there is no true imersiveness. Surface light fields can provide a solution. surface light fields are a light-field representation with spatial information, where the light is modelled on the surface where the light originates from. This article explains why surface light fields can be the solution to the problem of lacking immersiveness in the context of SMoE and presents the first steps towards integrating surface light fields in SMoE. Additionally, a novel method for creating light-field data sets is presented. In our surface-light-field visualisation it is already clear that immersive applications such as VR could benefit from surface light fields.**

***Index Terms*—surface light fields, Steered Mixture-of-Experts, SMoE, VR, light-field modelling, data-set creation**

I. Introduction

A light field represents the amount of light travelling in every direction through every point in space. An observer can both rotate and translate their viewpoint inside a light field. This is called Six Degrees of Freedom (6-DoF): 3 degrees of freedom in translational movements and 3 in rotational movements. The 6-DoF property makes light fields especially useful in Virtual Reality (VR). For multimedia purposes, a common mathematical representation is the following variation of the plenoptic function

$$\mathbb{R}^5 \to \mathbb{R}^3 : L(x, y, z, \theta, \phi) \mapsto \mathbf{c}.$$

x, y and z represent a point in 3-D space and the angles θ and ϕ indicate the direction of the incident light. The result is a three-component colour value $\mathbf{c}$, the colour of the light.

The SMoE framework models light fields by approximating this function with a *mixture model*, of which components are called *kernels* [1]. A kernel can be viewed as a multi-dimensional generalisation of the pixel. The kernels are simply multivariate normal distributions. Important to note is that the Gaussian kernels can only perform linear approximations; non-linear features cannot be modelled with a single kernel. The perspective projection causes non-linearity in the current representation of a light field in the SMoE framework.

A. SMoE

In the following a more practically-oriented insight is given in the workings of the current SMoE modelling process. The input of the SMoE modelling process requires a set of images representing a light field. Each pixel of each image represents a data sample. Each input sample therefore consists of two pixel coordinates and two camera coordinates, forming an evenly spaced 4-D grid The samples are used as input for the *Expectation-Maximisation* algorithm to correctly orient and shape the kernels. An important optimisation here is that the input data is split into equally-sized chunks. Instead of using all data at once, random samples of each chunks are combined to form minibatches, greatly improving performance [2]. The optimisation algorithm used is the *Expectation-Maximisation* (EM) algorithm.

The result of this approximation step is a kernel representation of the input data. The approximated plenoptic function can be queried to reconstruct viewpoints [3].

The steps from input data to view rendering in practice can be summarised as follows:

1) Structure data as 4-D grid of pixel and camera positions.
2) Split the data in manageable blocks.
3) Initialise mixture-model components.
4) Extract a minibatch from the data blocks.
5) Perform an EM step.
6) Repeat the previous two steps until the kernel-approximation converges.
7) The approximated function is sampled according to the desired view [3].

B. Perspective Projection

SMoE currently operates on planar light-field data. Planar light fields have the big advantage that they contain a lot of linearity. This is illustrated with the help of Figure 1. Data is captured with a camera moving along axis t. Stacking the continuous set of generated images results in the spatio-temporal volume in the top right. A slice of this volume parallel to the time axis shows how the pixels in a single row or column of the camera viewpoint change along axis t (bottom figure). Such a slice is called an epipolar plane image (EPI) and the sloped strips corresponding to an object are called EPI-strips. Those EPI-strips are the source of linearity in planar light fields.

Planar light fields are not always linear everywhere; features along the depth of the scene are more complex to model in case of a perspective projection (which entails all practical cameras). This is illustrated in Figure 2. This image shows an array of images, each capturing the same plane in 3-D space with a blue-black checkerboard texture from a slightly

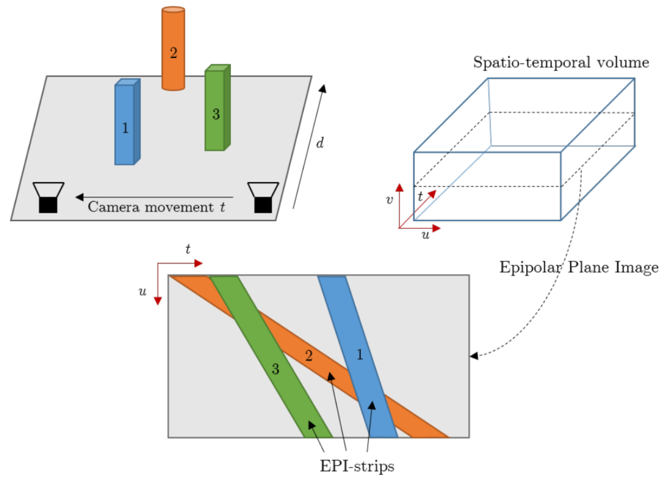


Fig. 1: Linearity in planar light fields comes from the EPI-strips in a plane of the spatio-temporal volume. Image from doctoral dissertation of Ruben Verhack [4].

different position. The frontmost image was captured looking almost along the plane. The non-linear effect of the perspective projection for depth-aligned features is clear on the top edge of the rectangle highlighted in red. This feature is not linear along consecutive images.

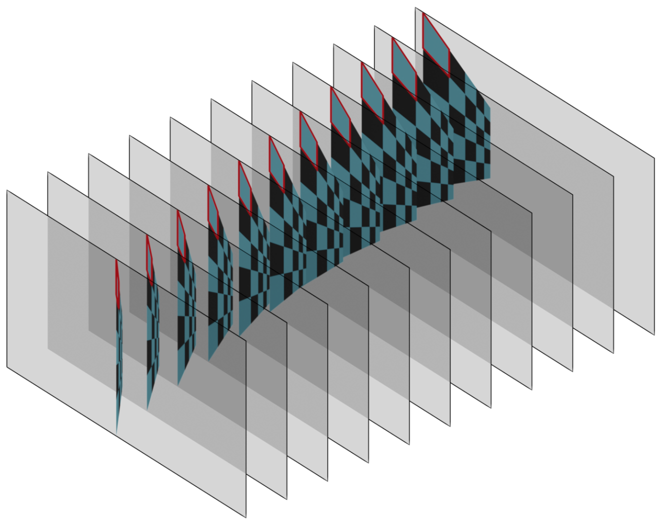
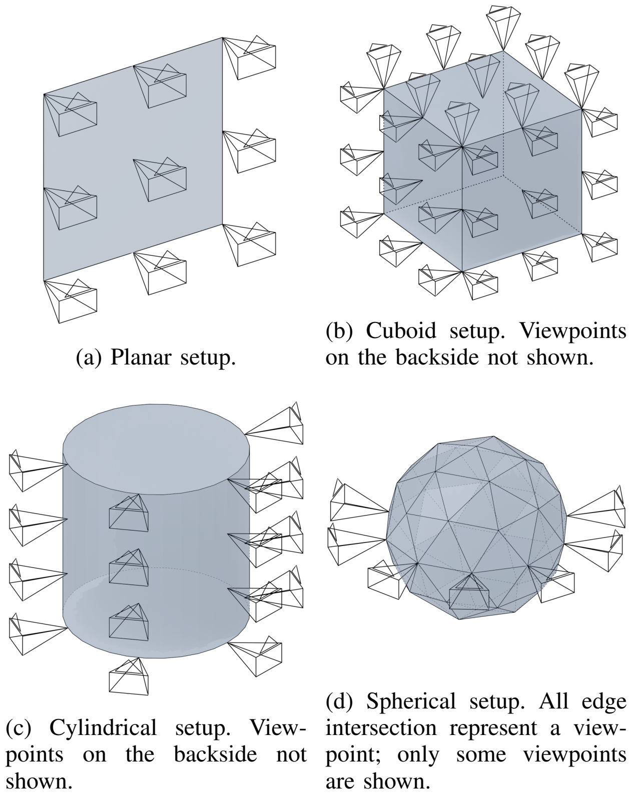


Fig. 2: A stacked set of images of a plane with a checker-board texture. The images were captured from evenly spaced viewpoints on a line perpendicular to the plane. The top and bottom edge of the highlighted red square cannot be modelled by a single kernel because it is not linear.

Non-linear features such as the above are more complex to model because they need multiple kernels for a good approximation. A single Gaussian kernel is unable to model a non-linear feature by itself.



(a) Planar setup.

(b) Cuboid setup. Viewpoints on the backside not shown.

(c) Cylindrical setup. Viewpoints on the backside not shown.

(d) Spherical setup. All edge intersection represent a viewpoint; only some viewpoints are shown.

Fig. 3: Different light-field capture configurations implemented in the Blender light-field add-on.

II. Improving Immersiveness

The basic idea to improve immersiveness is simple: capture light from all directions. In practice, this means adding cameras in multiple directions. Sections II-A to II-C will discuss several methods to effectuate this.

A. *Cubical Configurations*

By placing six planar light fields on the sides of a cuboid, light can be captured in all directions (see Figure 3b). The question is if the light field should be modelled as six separate planar light fields or as one cuboid light field. The advantage of using six separate planar light fields is that the algorithms already exist; the theory of planar light fields is now simply applied on six light fields instead of one. On the other hand, there is a possibility that some artefacts might become visible at the edges and corners of the cuboid when using separate planes because the data and the model of one light field stops and that of another one starts.

Cuboid configurations do not have a lot of space in which a user can look in all directions without missing data. For this reason, other types might be more applicable.

B. *Round Configurations*

In round configurations, the immersiveness is improved by placing cameras on a circle. Circles have an inside viewing area which is quite elegant compared to cuboid configurations. Figure 4 shows this area in green, where the cameras are placed on the outer circle. The ratio of inner area to area covered by the capturing circle (i.e. circle with cameras) is:

$$\frac{a_{\text{inner}}}{a_{\text{capture}}} = \sin^2(\frac{1}{2}\theta_{\text{FoV}}),$$

where a_{inner} is the area of the inner zone, a_{capture} is the area of the capturing circle, and θ_{FoV} is the field-of-view angle of the capturing cameras. Configurations that benefit from this improvement are spherical and cylindrical configurations, see Figures 3c and 3d.

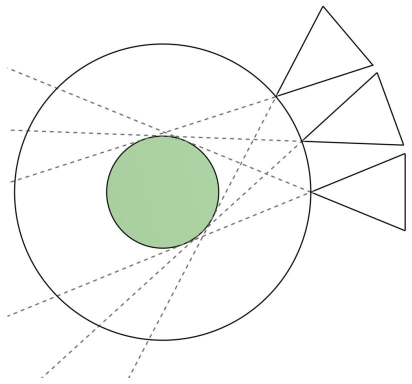


Fig. 4: Cameras on a circle all cover the green central area. Inside this area a viewer can look in all directions without missing data in their view.

Despite this nice property, round configurations are not easily modelled due to their curved surfaces. A curved surface is not ideal for modelling with linear kernels such as Gaussian kernels. Luckily, the features in round configurations are still approximately linear and might therefore yield reasonable results when modelled with the SMoE framework. It is up to future research to confirm this statement.

C. Surface Light Fields

Instead of constructing the light field on the plane it was captured from, the captured light could be projected back onto the geometry it originated from, i.e. the light ray's last bounce. The light field is then modelled on the geometry, assuming the geometry is approximately known. This type of light fields is called *surface light fields* [5]; the light field is modelled on a surface in 3-D space.

There are three main advantages of surface light fields. Firstly, because the perspective projection is undone by the reprojection in space, there are almost no non-linearities due the perspective projection (this was a problem for planar and round light fields, see Sections I-B and II-B). Almost, because if the geometry does not match perfectly, the error between the actual geometry and the approximate geometry will again introduce a perspective projection that will compensate for that error. However, as the geometry should be quite close to the real geometry, the features will most likely not be depth aligned. It can therefore be concluded that the problematic non-linearity of depth-aligned features is minimal with approximate geometry. The linearity of features now only depends on the properties of the captured surface because the concept of EPI-strips no longer exists.

Secondly, surface light fields expand the viewing area and therefore allow more immersiveness. SMoE currently models the light field on a plane. As the user walks into a planar light field, data might be missing due to depth-based occlusions, breaking the immersiveness. When the light field lies on geometry, the observer can walk around that geometry, possibly even interact with it. When light information is not captured in the direction the user is looking, light information could potentially be inferred by extrapolating neighbouring light.

Thirdly, light information close in space is even more correlated in surface light fields compared to planar light fields. In the latter case, the information alternates between highly correlated data with abrupt changes. Light in the same pixel-location of a planar light field can originate from a multitude of different objects. When modelled on the originating geometry, abrupt changes only appear when they are inherent to the surface.

This work continues with the idea of surface light fields, considering the prominent benefit of possibly unlimited immersiveness. The next section discusses the necessary changes to the SMoE framework to allow modelling of surface light fields.

III. Surface Light Fields and SMoE

The main principles of the process summarised earlier in Section I-A remain the same. However, each individual step has to change slightly. For the first two steps, a novel preprocessor has been developed.

In the first step, the data is no longer structured as a dense 4-D grid of pixel positions and camera positions. Instead the data is spread in 3-D space. For mathematical convenience a third dimension is added to the light direction of each sample, to prevent discontinuities; the light-field data is now sparse and six dimensional. To get the data in 3-D space, the depth-map output from the Blender add-on is used to place pixel samples at their corresponding depth.

In the second step, the data is divided into blocks. For surface light fields, a form of space partitioning is used to alter the splitting procedure from Verhack et al. [2]; splitting the sparse data in equal-sized blocks would result in very unbalanced minibatches. Two types are available in the novel preprocessor: octree space partitioning and median cut partitioning. The advantage of using octrees is the speed and ease of doing splits. Unfortunately octrees cause minibatches to be unbalanced in number of samples, but mild repetition of data in a minibatch to attain equally-sized blocks is allowed. The median cut partitioning does not have this unbalance and splits the data in nearly equal-sized blocks (nearly because the number of blocks does not necessarily divide the number of samples). The trade-off for the median cut algorithm is

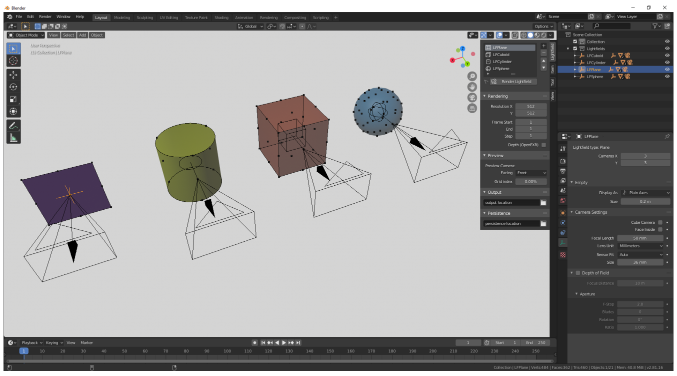


Fig. 5: Blender with the different configurations in the viewport. The toolbar shows options for rendering the light field, while the data panel on the right allows for configuring each individual setup.

performance; finding the median of millions of light samples is slow. Finally, the data of all blocks is written to disk.

In the fourth and fifth step the minibatches are extracted, followed by an EM step. In theory these steps do not change. The fourth step simply extracts its minibatches from the 6-D blocks on disk, while the principle of the EM step remains. In future research, different methods of approximating the data with the kernels should be looked.

To render a view, the approximation function is evaluated on the visible geometry of the surface light field. This sounds simple but it is not; naively implementing this results in incredibly long computation times. It remains to be seen how surface light-field rendering can be optimised.

IV. Data-Set Creation

To research and test different types of light fields, a method of obtaining the corresponding data is needed. Previous research already provided a tool for creating synthetic light fields [6]. Along with some performance issues, the tool lacks the ability to capture more immersive light fields than planar light fields. A novel tool is created to cope with those shortcomings, but also to allow capturing of different light-field configurations. It comes in the form of a Blender add-on, similar to the existing tool.

A. Light-Field types

The novel add-on provides new capture setups along with the existing planar type. Support is added for cuboid, cylindrical and spherical light-field capture setups. Capturing spherical light-field data has been tried before, but not in a synthetic manner [7, 8]. All three new types provide data from all around the scene, whereas the planar setup only captures in a single direction. The captured views are spaced over the respective surface, as illustrated in Figure 5, showing a screenshot of the configurations in the Blender add-on.

B. Inverting Cameras

The add-on allows cameras to face the opposite direction. This option is mostly useful for the cylindrical and spherical light-field types. Facing the cameras inwards as opposed to outwards causes neighbouring cameras to have more overlap in their viewing range. Objects are therefore more likely to be captured by more cameras than before. The scenario with the cameras facing outside is pictured in Figure 6a. The different viewpoints in this figure overlap only at a greater distance. In Figure 6b, a small area is covered by all three cameras, while two cameras have quite a large area in common. The benefit is that when more viewpoints capture an object, more specular information can be stored. On the other hand, the viewpoints are further away from the objects. Slightly less detail might be captured, especially for objects close to the edge of the circle.

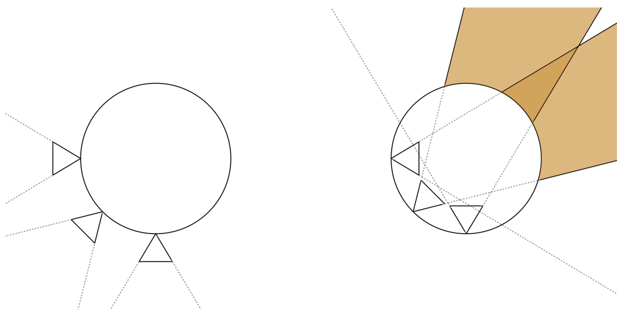

(a) Outwards. (b) Inwards.

Fig. 6: Different light-field capture configurations implemented in the Blender light-field add-on.

V. Results

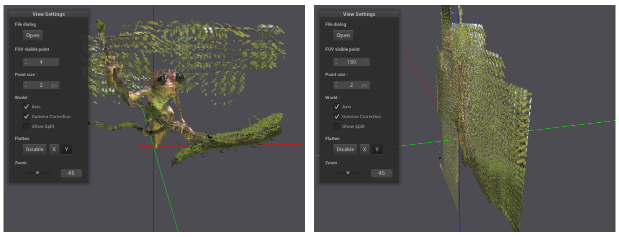
A. Light-Field Visualisation

For some early results and experimenting a novel light-field visualisation tool is developed. The tool allows for navigating through a light-field point cloud of 6-D samples: 3 location coordinates, 3 coordinates for a unit vector indicating the light direction. A sample is only rendered if the sample its light direction approximately hits the viewers viewpoint.

Along with some other options, the light field data can be mapped on a coordinate plane (see Figures 7a and 7b) This is done by regarding each sample as a light ray and moving the sample to the intersection of the ray and the plane used for projection. The result of mapping a surface light field on a plane is similar to a planar light field. This indicates that surface light fields are some sort of generalisation of planar light fields. A user can navigate through the geometry of the original scene with the help of the spatial information of surface light fields, contrarily to planar light fields. From the first visual evaluations it is evident that surface light fields are a great candidate to solve the problem of immersiveness.

B. Blender Add-On Performance

Compared to the an existing method of creating light-field data sets, the novel implementation is 30% faster in terms of generating images for each viewpoint, due to the simple difference between versions of Blender. Additionally, the novel



(a) Front view of the mapped light field. This is very similar to the unaltered surface light field from this direction.

(b) Samples are visible under a larger angle, clearly showing that the data is flattened.

Fig. 7: Mapping a surface light field to a coordinate plane.

method also has access to Blender's physically-based real-time rendering engine. That engine is useful when physical inaccuracies are allowed, speeding up the data-generation time drastically.

In terms of time it takes to create a light-field capture configuration, the novel method outperforms the existing one by a lot; the existing method its creation time is quadratic in function of number of viewpoints. Where the existing add-on takes 2 seconds for a configuration with 200 viewpoints, the novel add-on takes less than a second for a configuration with a million viewpoints.

VI. Conclusion

The goal of this work was to improve the immersiveness of light fields. To this end, several solutions where researched, with surface light fields coming out on top; surface light fields allow for true immersiveness due to the incorporation of the geometry of the scene. As a part of the goal, a novel tool to generate light-field data sets was developed from which future research will also benefit. It improves on an existing method in terms of performance and variety in capture configurations. To evaluate the work of integrating surface light fields in the SMoE framework, a 6-D light-field point-cloud renderer is developed to assist in a qualitative evaluation.

In the years to come a multitude of design options and optimisations need to be researched. The choice of using surface light fields in this work does not exclude cuboid, cylindrical and spherical configurations from future work. In terms of SMoE modelling, maybe gradient descend performs better than the EM algorithm, although EM already forms a good match with Gaussian mixture models [9]. Also, the Gaussian kernels could be replaced by alternatives that are better at modelling non-linearities such as the curved surface of the round configurations, while maintaining compression performance and simplicity.

But most importantly, the work towards integrating surface light fields in the SMoE framework needs to be completed. Only then can the properties of surface light fields in the SMoE framework be evaluated in terms of complexity, modelling times, compression rates, rendering times and geometry approximation.

References

[1] R. Verhack, T. Sikora, G. Van Wallendael, and P. Lambert, Steered mixture-of-experts for light field images and video: Representation and coding, *IEEE Transactions on Multimedia*, vol. 22, no. 3, pp. 579–593, 2020.

[2] R. Verhack, G. Van Wallendael, M. Courteaux, P. Lambert, and T. Sikora, Progressive modeling of steered mixture-of-experts for light field video approximation, eng, in *2018 PICTURE CODING SYMPOSIUM (PCS 2018)*, San Francisco, CA, USA: IEEE, 2018, pp. 268–272, ISBN: 9781538641606.

[3] I. Saenen, R. Verhack, V. Avramelos, G. Van Wallendael, and P. Lambert, Hard real-time, pixel-parallel rendering of light field videos using steered mixture-of-experts, eng, in *2018 PICTURE CODING SYMPOSIUM (PCS 2018)*, San Francisco, CA, USA: IEEE, 2018, pp. 337–341, ISBN: 9781538641606.

[4] R. Verhack, Steered mixture-of-experts for image and light field representation, processing, and coding: A universal approach for immersive experiences of camera-captured scenes, PhD thesis, Technical University of Berlin, 2020.

[5] X. Zhang, P. A. Chou, M. Sun, M. Tang, S. Wang, S. Ma, and W. Gao, A framework for surface light field compression, in *2018 25th IEEE Int. Conf. on Image Processing (ICIP)*, 2018, pp. 2595–2599.

[6] K. Honauer, O. Johannsen, D. Kondermann, and B. Goldluecke, A dataset and evaluation methodology for depth estimation on 4D light fields, in *Asian Conf. on Computer Vision*, Chem, Germany: Springer, 2016, pp. 19–34.

[7] B. Krolla, M. Diebold, B. Goldlücke, D. Stricker, and H. Collaboratory, Spherical light fields, in *BMVC*, 2014.

[8] A. Akin, Ö. Cogal, K. Seyid, H. Afshari, A. Schmid, and Y. Leblebici, Hemispherical multiple camera system for high resolution omni-directional light field imaging, *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, vol. 3, pp. 137–144, Jun. 2013.

[9] L. Xu and M. I. Jordan, On convergence properties of the em algorithm for gaussian mixtures, *Neural computation*, vol. 8, no. 1, pp. 129–151, 1996.

Contents

Preface iii

Abstract v

Extended Abstract vi

Contents xi

List of Figures xiii

Abbreviations xvi

Image Attribution xvii

1 Introduction 1
- 1.1 Context . . . 1
- 1.2 Problem Statement . . . 7
- 1.3 Goals and Objectives . . . 7

2 Background and State of the Art 9
- 2.1 Light Fields . . . 9
  - 2.1.1 Capturing Light Field Data . . . 10
  - 2.1.2 Rendering Light Fields . . . 13
  - 2.1.3 Compressing Light Fields . . . 15
  - 2.1.4 Conclusion . . . 16
- 2.2 Steered Mixture-of-Experts . . . 16
  - 2.2.1 Kernels . . . 17
  - 2.2.2 Modelling using SMoE . . . 17
  - 2.2.3 Rendering using SMoE . . . 18
- 2.3 Blender . . . 19
- 2.4 Previous Blender Light-Field Creation Add-On . . . 19
- 2.5 Conclusion . . . 19

3 Improving Immersiveness of Light Fields 21
- 3.1 Introduction . . . 21
- 3.2 Configurations . . . 22
  - 3.2.1 Cubical Configuration . . . 22

3.2.2 Round Configurations 23
3.2.3 Problems with Round Configurations 27
3.3 Surface Light Fields 31
3.4 Surface light fields for SMoE 33
3.5 Conclusion 34

4 Implementation 36
4.1 Introduction 36
4.2 Preprocessor 37
4.2.1 Data Preprocessing 37
4.2.2 Splitting 38
4.2.3 In-Memory Processing 40
4.3 Data Creation 41
4.3.1 Blender Add-On 41
4.3.2 Inverting Cameras 41
4.3.3 Performance 43
4.3.4 Blender Integration 43
4.4 Conclusion 43

5 Evaluation 45
5.1 Blender Add-On Performance Comparison 45
5.1.1 Creation Performance 45
5.1.2 Rendering Performance 47
5.2 Qualitative Evaluation of Preprocessed Light-Field Data 48
5.2.1 Verification 49
5.2.2 Flattening 49
5.2.3 Real-World Captured Surface Light Field 51
5.3 Conclusion 55

6 Conclusion and Future Work 57

References 59

List of Figures

1.1 Schematic of a planar light field. The most common light field capture configuration. The cameras are evenly spaced on a grid. 3

1.2 Illustration of SMoE in 2 dimensions on the Lena test image. Notice how each ellipsoid (a kernel) has only one colour, but still succeeds in conveying the image. The kernels are ellipsoid shaped because of their Gaussian nature. Although Gaussians have unlimited support (they never become zero), in this image they are truncated for illustration purposes. Image from [2] 4

1.3 Schematic overview of how what EPI-strips are. Capturing images during camera movement t results in the spatio-temporal volume in the top right. Taking the dotted slice from that volume makes the EPI-strips visible, which represent the movement of an object through a row or column of pixels. 5

1.4 A stacked set of images of a plane with a checkerboard texture. The images were captured from evenly spaced viewpoints on a line perpendicular to the plane. The top and bottom edge of the highlighted red square cannot be modelled by a single kernel because it is not linear. 6

2.1 Camera-based light-field capture setups. 11

2.2 The principle of a plenoptic camera. The micro-lenses are placed in a hexagonal grid between the image sensor and the main lens. Each micro-lens observes a slightly different section of the coloured objects through the main lens. The result of each individual micro-lens is projected on the image sensor. 12

2.3 Google spherical light-field setup. It consists of 16 GoPros arranged on a rotating mount. This setup can capture one light-field still in about a minute. 13

2.4 From simple image-based rendering to light-field rendering. The image is based on an image from [1]. 14

2.5 A schematic overview of light-field coding with video based techniques. Light-field views are treated as video-frames and are encoded based on other views. A B_4 frame requires decoding of a maximum of four other frames. Those frames must in their turn first be decoded based on even other frames. 16

2.6 High-level schematic of the Steered Mixture-of-Experts (SMoE) pipeline. 17

3.1 Planar light field with new viewpoints. While view **B** succeeds at rendering its view, viewpoint **C** is missing some light data, and viewpoint **A** has no data at all. 21
3.2 A cuboid light-field configuration which captures light in all directions. The cameras on the back side are not shown for illustration purposes. 23
3.3 Camera overlap in the 2-D version of a cuboid. Greyer areas indicate more overlap in viewing region. When the field of view is too small, there is no area where all cameras are visible. 24
3.4 When cameras are placed in a circle formation, there exists an inside viewing area, in which the user does not have limited vision in any point or orientation. 25
3.5 Ration between inner area and capture area in function of field of view. 26
3.6 The cylindrical light field configuration. The cameras are placed on equally spaced rings on the cylinder. Viewpoints on the back are not shown. 26
3.7 The spherical light field setup. Only viewpoints around the centre are shown. Each line intersection in the figure represents a camera position. 27
3.8 Schematic of the relation between an object and its position in an image. k is the distance from the centre of the image to the object in the image. 28
3.9 k in function of θ and o_x. Notice how the green area is almost linear. For large θ or objects close to the capture surface (small o_x), the relation is no longer approximately linear, but approaches infinity. . . 29
3.10 Comparison of k and its linear approximation for several values of o_x. Objects further away (larger o_x) result in better approximations. . . . 30
3.11 Figure 3.11b shows a top view of the setup. An array of cameras captures a plane aligned with the depth axis of all cameras. The result of each viewpoint is shown in Figure 3.11a. Both figures show the red rectangle highlighting one rectangle of the blue-gray grid. The volume encapsulated by the red highlights cannot be represented by a single kernel since it is not linear. 31

4.1 High-level schematic of the SMoE pipeline. The blue components require modifications to function with surface light fields. 36
4.2 The inverse process of performing a perspective projection with a camera. The image in screen coordinates (Figure 4.2a) is first transformed to clip space with coordinates between -1 and 1 (Figure 4.2b). The data is then projected back into space from the camera with the help of the provided geometry in the form of a depth map (Figure 4.2c). Finally the resulting samples are transformed to be in the correct place in the complete scene (Figure 4.2d). 39
4.3 Changing the direction of the camera's on cylindrical or spherical setups capture more specular information (more directions), but lose some detail. Its use in real world scenarios is fairly limited. 42

5.1 Configuration-creation time in function of number of cameras for the existing add-on. Creation times skyrocket with increasing numbers of cameras. 46
5.2 Configuration creation time in function of number of cameras for our add-on. The inaccuracy is due to the timing method not being perfectly accurate, even though each configuration was tested a 100 times. Also note that the time axis is in milliseconds, not in seconds. 47
5.3 The different scenes used for comparing render times. 48
5.4 Comparison between Blender 2.81 and 2.79b in terms of rendering speed of the ray-tracer. It is clear that Blender 2.81 is faster. 48
5.5 Two images of the preprocessed surface-light-field data of the tree creature. Notice the highlights changing depending on the view, which indicate the preprocessing succeeded. 50
5.6 The tree-creature data set with the median-cut partitioning visible. The white boxes indicate volumes containing samples, all of which have the same amount. 51
5.7 A flattened version of the tree-creature data set. 52
5.8 Cameras on a plane with data points flattened. The blue dots indicate the original position, while the orange dot indicates the projected position. 53
5.9 A scene in the 6-D point-cloud viewer captured with a physical camera. The black dots are used for tracking. 54
5.10 The geometry used for generating the accompanying depth information of the surface light field. Only the main objects are blocked out. 55
5.11 Four screenshots from the resulting 6-D point cloud. Notice how the reflection on the white tabletop changes depending in the angle. Other interesting areas are objects with no geometry (e.g. bag of candy in front of the bottle, or glass to the left of the bottle) and the transparency of the bottle. 56

Abbreviations

6-DoF Six Degrees of Freedom. 2, 7

API Application Programmable Interface. 19, 20, 43, 49

BLAS Basic Linear Algebra Subprograms. 38

EM Expection-Maximization. 18, 33

GMM Gaussian Mixture Model. 17

HEVC High Efficiency Video Coding. 15

JPEG Joint Photographic Experts Group. 15

MoE Mixture-of-Experts. 17

MPEG Moving Pictures Experts Group. 15

SMoE Steered Mixture-of-Experts. xiii, xiv, 6, 7, 8, 9, 10, 15, 16, 17, 18, 20, 22, 27, 30, 31, 32, 33, 34, 35, 36, 37, 43, 45, 48, 57, 58

VR Virtual Reality. 1, 2, 3, 7, 14, 16, 19, 21, 23

Image Attribution

- Figures 1.2 and 1.3 on pages 5 and 4 use images from the doctoral dissertation of Ruben Verhack.
- Figure 2.1a on page 11 uses a photo from `https://graphics.stanford.edu/projects/array/`.
- Figure 2.1b on page 11 uses a photo from `http://lightfield.stanford.edu/acq.html#gantry`.
- Figure 2.3 on page 13 uses a photo from `https://www.blog.google/products/google-ar-vr/experimenting-light-fields/`.
- Figure 2.4 uses self-made images inspired by images in the Light Fields and Computational Imaging paper [1].
- Figure 2.5 on page 16 is an image from the design of multimedia applications course for which explicit permission was received.
- Figures 5.3a to 5.3d on page 48 are rendered versions of the available Blender demo-files at `https://www.blender.org/download/demo-files/`.

All unlisted images are my own.

Chapter 1

Introduction

1.1 Context

Virtual Reality (VR) is a technology that allows a user to be presented with visual multimedia content in an immersive way. VR in the context of this dissertation forms a subcategory of the VR domain, where the user is visually immersed in another reality by replacing real-world visual stimulus by computer generated content. This is done by mounting a display in front of the user's eyes, which presents content according to the user's visual actions. Visual actions include looking around by moving or rotating the head, the eyes, or even changing the position of the whole body. The goal is to simulate the feeling of being in the presented reality as accurately as possible. This is what is meant by immersive: the more users feels like they belong in the presented contend, the more immersive the experience is.

The majority of VR content today is computer generated. To understand why, we have to look at the content creation possibilities today. When artists create computer-animated movies such as Disney's Frozen or games such as Valve's Portal 2, they design a 3-D scene first. Afterwards, a camera can be moved around freely and explore the scene from different standpoints, providing a VR user with views corresponding to their movements. In contrast, when shooting a film with cameras in the real world, artists do not have the luxury of adapting the camera position after the scene was shot earlier. Perspectives that were not captured during the original capture session cannot be veraciously shown. This makes it hard to present real-world content compared to computer-generated content.

To cope with this problem of real-world content (e.g. not being able to capture every angle), researchers found ways of synthesising multiple existing viewpoints into novel viewpoints. It is now possible to generate infinitely many viewpoints from surrounding captured data, at the expense of truthfulness of the created view. For example, when there are two cameras which are pointing in the same direction and spaced 30 cm apart, a viewer can move between those two cameras. With the help of view synthesis, the viewer can now have a continuous image when moving between the captured viewpoints, as if they were in the original capture moment and location.

What is still missing compared to a computer-generated VR environment is both perception of depth and more freedom. Creating depth could be resolved by synthesising a second viewpoint, in the location of the second eye of the user. Two separate viewpoints for both eyes, or stereopsis, is what gives humans 3-D vision. With stereopsis, an important depth cue is obtained, which is an important factor in immersiveness.

Granting the viewer more freedom is somewhat more complicated. In computer-generated content, the user can rotate and translate their viewpoint freely, not restricted by the camera positions during capturing; this is called Six Degrees of Freedom (6-DoF). The viewer can move (translate) in 3-D space, while rotating their head in the coronal, transverse or sagittal planes. For real immersiveness, six degrees of freedom is a must. Simple view-synthesis is not enough since it performs better at view interpolation compared to view extrapolation.

This is where light fields come in. A light field is a representation of the light in a section of space. It gives the amount of light travelling in every direction through every point of that space. Alternatively, it can be thought of as the collection of all light rays which intersect a given region. A picture is actually a small 2-D slice from the surrounding light field; all light rays passing through the lens of the camera which hit the image sensor are converted into pixels that represent those light rays.

For the purpose of VR, we want to capture, store and quickly query the complete light field of a given real-world scene. In theory it is simply a matter of querying the light field in the locations of the viewer's eyes and displaying the result to the viewer; the amount of eyes the viewer has is of no importance. It is clear that light fields can provide the six degrees of freedom required for immersiveness in VR because any viewpoint can be recreated. Chapter 2 will go more in depth on the theory of light fields.

In practice, capturing, storing and querying of light fields are difficult problems. Firstly, a complete light field cannot be captured; a captured light field has a finite resolution. This is similar to pictures: when light falls on an image sensor in a camera, the total light over a small area is recorded to a pixel, not individual light rays. It is impossible capture every ray in every point in every direction; not for photos nor for light fields. Therefore we have to compromise in which points and which directions light is captured. The simplest and most common capture setup is a configuration of evenly spaced cameras on a plane, all cameras pointing in the same direction, see Figure 1.1. This is called a planar light field. There of course exist other types of light fields, some of which are discussed in Section 2.1.1 and later. Standard cameras are used to capture light fields; they already capture light in a single point in space, so why not use them? There actually exists specific hardware to capture light fields, see Section 2.1.1. But in most cases that hardware can be abstracted to a setup of individual cameras, very similar to planar light fields.

Secondly, storage is a problem due to the massive size of light fields. For a planar light field of 10 by 10 cameras, 100 images are needed. Where a single uncompressed 512 by 512 colour image is 786 KiB, a light field image is 75 MiB. A two second light-field video at 30 frames per second results in 4.4 GiB. This obviously becomes

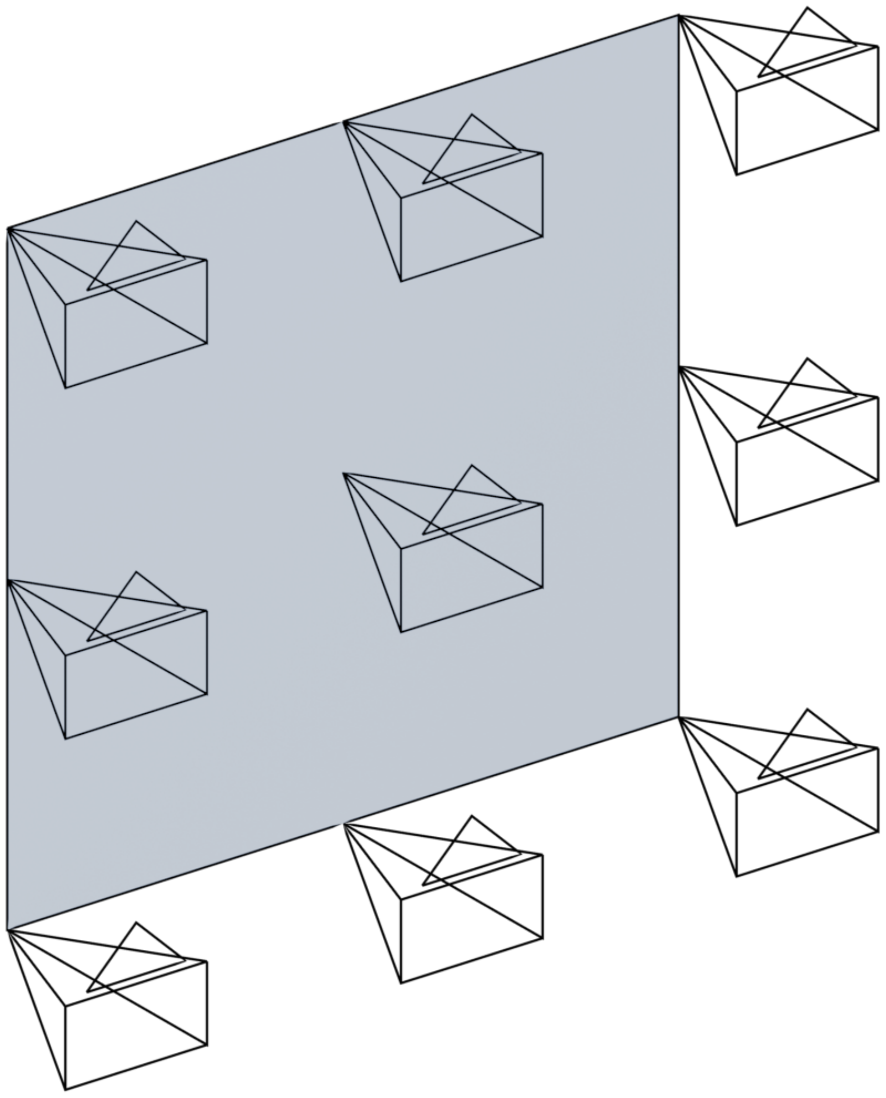


Figure 1.1 – *Schematic of a planar light field. The most common light field capture configuration. The cameras are evenly spaced on a grid.*

unmanageable without compression. Popular light-field compression techniques look over to video-compression standards for help, see Section 2.1.3. If a planar light field is treated similar as a video, good compression results can be achieved. Unfortunately, video coding standards are not designed for the random-access needs of VR. They often require decoding of neighbouring frames before a requested frame can be extracted. Also, video-coding based techniques struggle with the high dimensionality of the data.

This leads the third problem, which is fast access of light field views. A viewer can turn their head freely in any direction, resulting in a different sequence of frames each time. Accessing views from compressed data is slow when working with video compression. Another light-field compression possibility which does allow fast random access is the steered mixture-of-experts framework. This framework is an integral part of this dissertation. The SMoE framework abandons the concept of a pixel-grid altogether and transitions to a more free, content-dependent workflow for representing features (features are patches of similarly-coloured data). Instead of pixels, kernels are used. Kernels can be seen as the multi-dimensional generalisation of a pixel. The SMoE framework uses Gaussian kernels, or multivariate normal distributions. The idea is to model features or correlated areas in the input data with such kernels. Figure 1.2 illustrates what is meant by modelling features with kernels. This example shows how SMoE finds regions with coherent colours and groups them together with a single kernel.

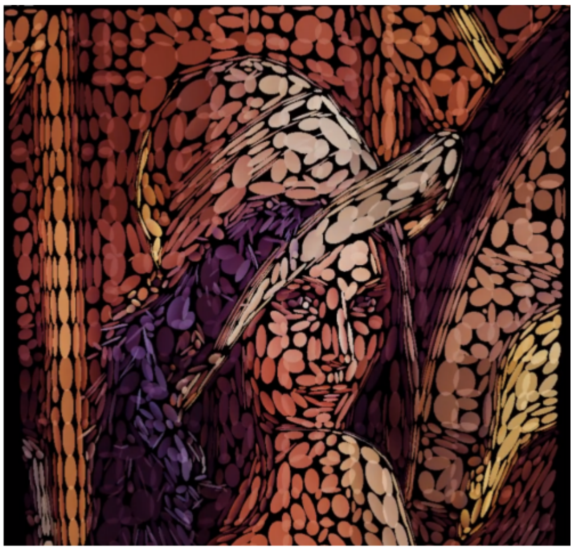


Figure 1.2 – *Illustration of SMoE in 2 dimensions on the Lena test image. Notice how each ellipsoid (a kernel) has only one colour, but still succeeds in conveying the image. The kernels are ellipsoid shaped because of their Gaussian nature. Although Gaussians have unlimited support (they never become zero), in this image they are truncated for illustration purposes. Image from [2]*

The data used in Figure 1.2 is a 2-D image, not a light field. This is not a problem since SMoE can be easily extended to more dimensions. This is the consequence of an interesting property of working with kernels, which is that they can model an arbitrary number of dimensions. For video, which has two spacial components and one temporal component, the kernels are 3-D. Planar light fields have two spatial camera components, and per camera two spatial pixel components, needing in 4-D kernels. All of these can be modelled by the same kernel type, independent of dimensionality. Light fields can have even more dimensions, as we will see in Section 3.2.

The Gaussian kernel type of the SMoE framework has the advantage of handling linearity very well. But this also works the other way around, Gaussian kernels cannot model non-linear elements. Luckily, planar light fields have mainly linear features in their data. To explain this, the situation is simplified to a single row of cameras instead of a grid. The idea is illustrated in Figure 1.3. The top left figure shows a scene with coloured objects, in which a camera moves from right to left, simulating a row of cameras. Along this path the camera captures a continuous set of images. The resulting data is 3-D, composed of an array of 2-D images at each point during the camera movement (1-D). This 3-D volume is shown in the top right image. v and u are the image coordinates, while the depth is the temporal component of the movement, t. Instead of looking at a single image like before in Figure 1.2, lets now look at the data in another plane, an epipolar plane. The resulting data is shown at the bottom. It originates from taking the slice indicated by the dotted line (e.i. epipolar plane) from the 3-D spatio-temporal volume; the result is a row of pixels from each camera image, stacked on top of each other. In this epipolar plane image (EPI), linear strips are visible, or EPI-strips. Those strips show how the objects in the scene move along a pixel row during the camera movement. The two objects in front (1 and 3) occlude the object in the back (2) at a certain point during the movement. Also, due to parallax, the front two objects which are close to the camera only appear for a shorter amount of frames in the EPI, while an object further away such as the orange object in the back is almost always visible along the camera movement. A side-benefit of SMoE here is that the slope of the EPI-strips can be interpreted as the inverse of the depth of the corresponding feature.

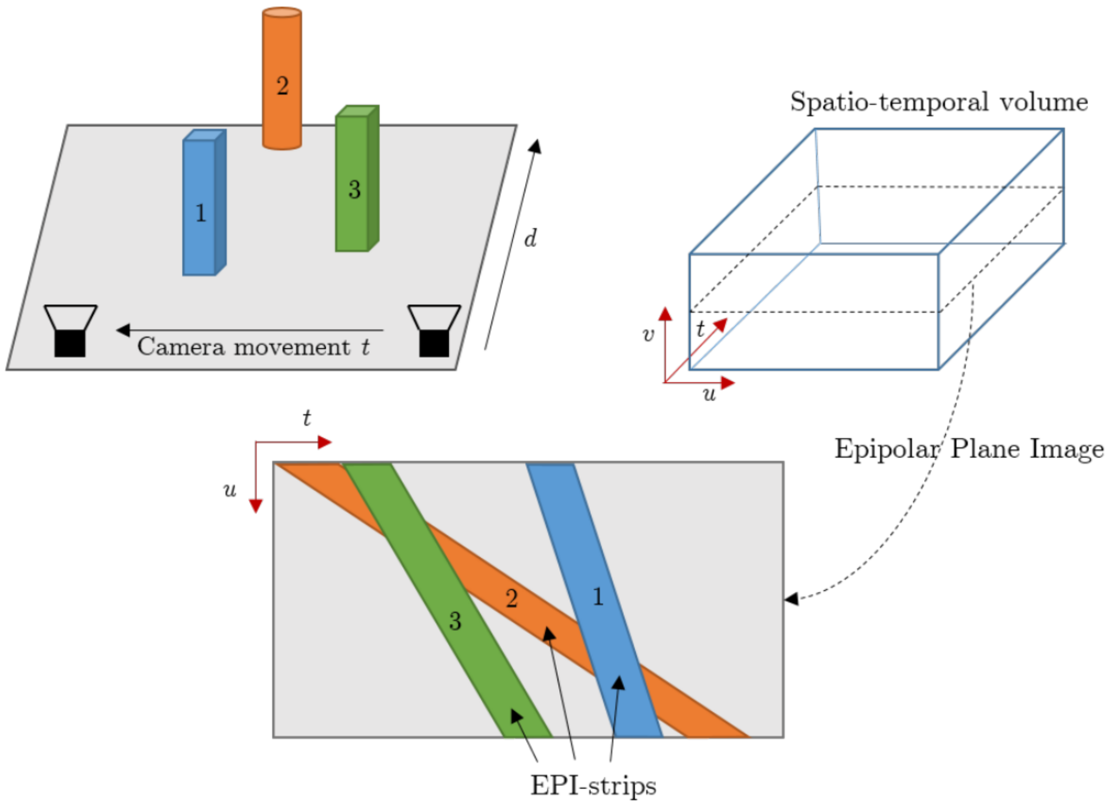


Figure 1.3 – *Schematic overview of how what EPI-strips are. Capturing images during camera movement t results in the spatio-temporal volume in the top right. Taking the dotted slice from that volume makes the EPI-strips visible, which represent the movement of an object through a row or column of pixels.*

Planar light fields are not always linear everywhere because features along the depth of the scene cannot be represented by a single linear kernel, in case of a perspective projection (which entails all practical cameras). This is illustrated in Figure 1.4. This image shows an array of images, each capturing the same plane in 3-D space with a blue-black checkerboard texture from a slightly different position. The front-most image was captured looking almost along the plane. The non-linear effect of the perspective projection for depth features is clear on the top edge of the rectangle highlighted in red. This feature is not linear along consecutive images, but more trapezoid shaped.

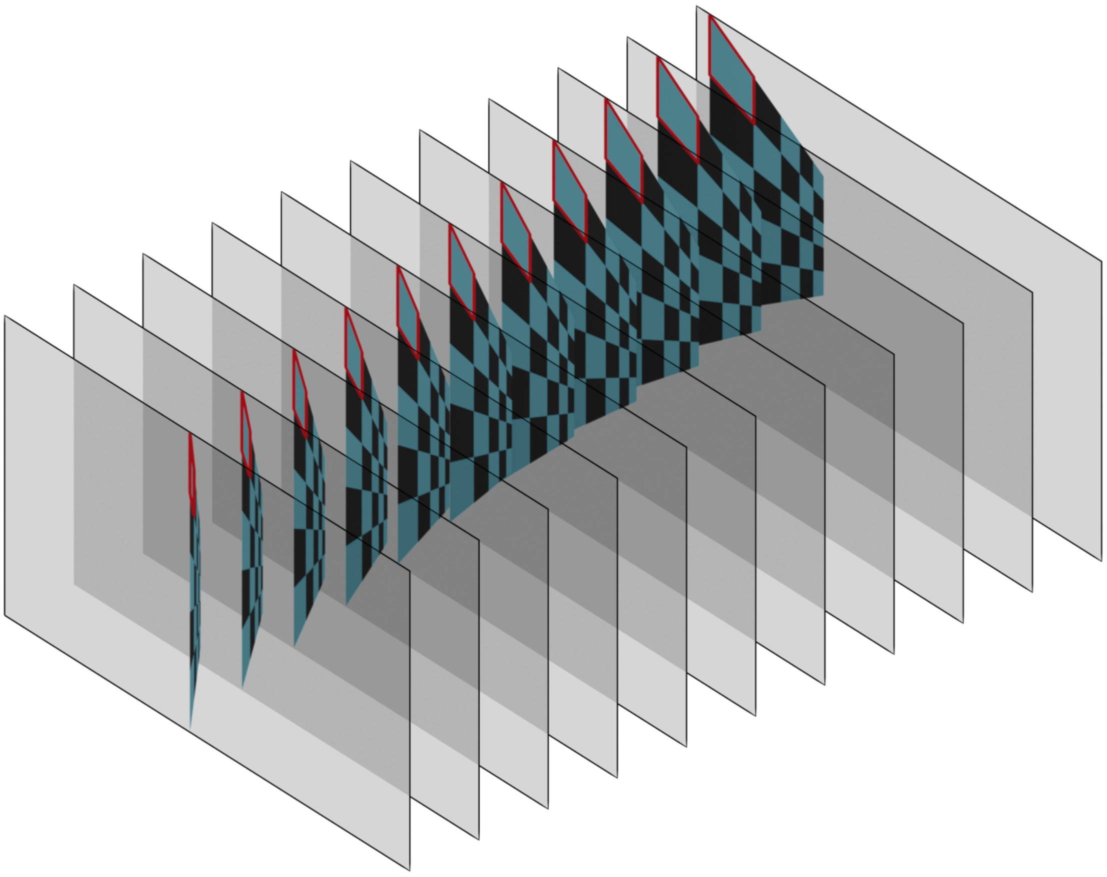


Figure 1.4 – *A stacked set of images of a plane with a checkerboard texture. The images were captured from evenly spaced viewpoints on a line perpendicular to the plane. The top and bottom edge of the highlighted red square cannot be modelled by a single kernel because it is not linear.*

In any case, it is clear that SMoE is a good fit for working with light fields. It achieves comparable compression rates to video-based codecs [3], while providing random access when querying views.

1.2 Problem Statement

The problem is the absence of complete immersiveness in current light-field technology, specifically in SMoE. This is caused by the fact that SMoE currently only works with planar light fields. Planar light fields suffice when wanting to take a 3-D picture of a scene with correct lighting, but they lacks immersiveness. A VR user might want to walk into the light field and look around. Planar light fields only capture light with cameras facing in a direction perpendicular to the camera plane; this only allows a user to 'look through' a window specified by the camera plane. The user cannot turn their head and see the other side of the original environment. Because light is only captured in one direction, a user cannot walk around an object and inspect it.

The question is, how can we make VR for real-world captured content be made more immersive than the current state of the art? We want to offer the complete 6-DoF experience to the user, not limited to a single plane. Consequently, we want to capture light in all directions. We want to use the SMoE framework due to its compression and random-access capabilities, but more importantly because video-based techniques will not be able to keep up with the rising dimensionality of the data, see Section 2.1.3. The SMoE framework will need to be adapted to operate on more complex data than that of planar light fields.

1.3 Goals and Objectives

The first goal of this dissertation is to explore the available options in terms of more immersive light fields. We want to research the advantages and disadvantages of each option, as to make an informed decision on how to continue this research. The implications of each option on the SMoE framework must be taken into consideration.

The second goal is to implement a solution in the SMoE framework. This dissertation does not intend to bring a finished product to the market, rather, it tries to show that the chosen solution might be the most profitable in terms of advantages. Because of this, this first work is intended to kick-start research in the chosen solution, which is surface light fields. To this end, we will evaluate surface light fields in their simplest form, as a point cloud.

To explore new options in practise, there is a need for light-field data sets tailored to those options. This is the third and final goal of this dissertation: we want to develop a method for generating light-field data sets according to the needs of the research. We will compare our method to others, as to show how we improved the state of the art.

Concretely, this dissertation presents the following novelties and contributions:

- We perform an in-depth comparison of the different options available for improving immersiveness of light fields.
- We show how the SMoE framework must be adapted to work with surface

light fields, and implement a novel data preprocessor which is responsible for preparing the data for SMoE.

- We contribute a novel tool that is able to synthetically generate light-field data sets of several types such as surface-light-field data sets. We evaluate and compare this tool to an existing method.

Chapter 2

Background and State of the Art

The current chapter presents a more in-depth look at light fields; how they are captured, rendered to a viewer and compressed. The state of the art is briefly reviewed for each of those. Some work on surface light fields is listed in the same context. Next, the SMoE modelling pipeline is scrutinised. Blender is introduced at the end of this chapter, alongside an existing light-field data-set creation solution.

2.1 Light Fields

As seen before, a light field gives the amount of incoming light in all directions in every point in space. A light field can be mathematically defined with the help of the plenoptic function. *The plenoptic function* is generally of the form:

$$\mathbb{R}^5 \to \mathbb{R}^3 : L(x, y, z, \theta, \phi) \mapsto \mathbf{c}.$$

The first three parameters represent the location, the last two represent the incident light direction. The result is a 3-dimensional colour value, the colour of the light. In the actual plenoptic function, the result would be the radiance along the light ray. Here the result is replaced since we mostly care about a 3-D colour value in computers. The plenoptic function is not very useful in practice, but it can help to understand the concepts.

From the plenoptic function it is evident that light fields have a 5-dimensional domain. It is not always necessary to have all 5 dimensions. In most cases it can be assumed that the light along a ray does not change, and that there are no occlusions. In those cases there are only 4 degrees of freedom, since all 5-D points along a single ray output the same colour value, and the colour is therefore independent of the position in the ray. As a result, the light field is technically 4-D. If there are occlusions, this is no longer the case, since all three spatial coordinates are needed to find the output colour; a line does no longer uniquely define the colour of any point. This is for example the case when there is smoke, or mist, or a simple solid occlusion. A point in front of the occlusion or behind it might not have the same colour. Also keep in mind that in practise, light suffers from attenuation, which means that the assumption of colour output being equal in each point of a ray is physically incorrect.

The following three subsections illustrate how the abstract theory above is used in light field industry and research. Section 2.1.1 presents the different ways in which light fields are captured. Section 2.1.2 explains what light field rendering is, and its state-of-the-art. Finally, Section 2.1.3 discusses the various ways of compressing light fields, and where SMoE fits in.

2.1.1 Capturing Light Field Data

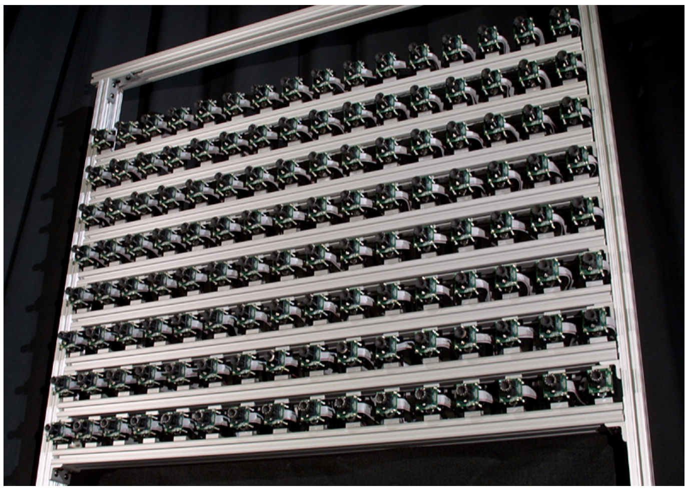
An image can be seen as a 2-D slice of a light field. A light field can be recreated by recombining enough of those 2-D slices. Capturing the complete light field would require all possible 2-D slices, which is of course not feasible. But with a limited set of images, it is still possible to approach the actual light field. Using images is currently the most common way to reconstruct light fields.

The most basic setup is a grid of cameras with the cameras evenly spaced in width and height. A good example of this is the Stanford Multi-Camera Array [4], shown in Figure 2.1a. It consists of 128 video cameras, and the spacing between them can be changed. Similarly, the Stanford Light Field Gantry can record the same light field, but uses only one camera [5]. The camera moves on rails in the horizontal and vertical direction (see Figure 2.1b).

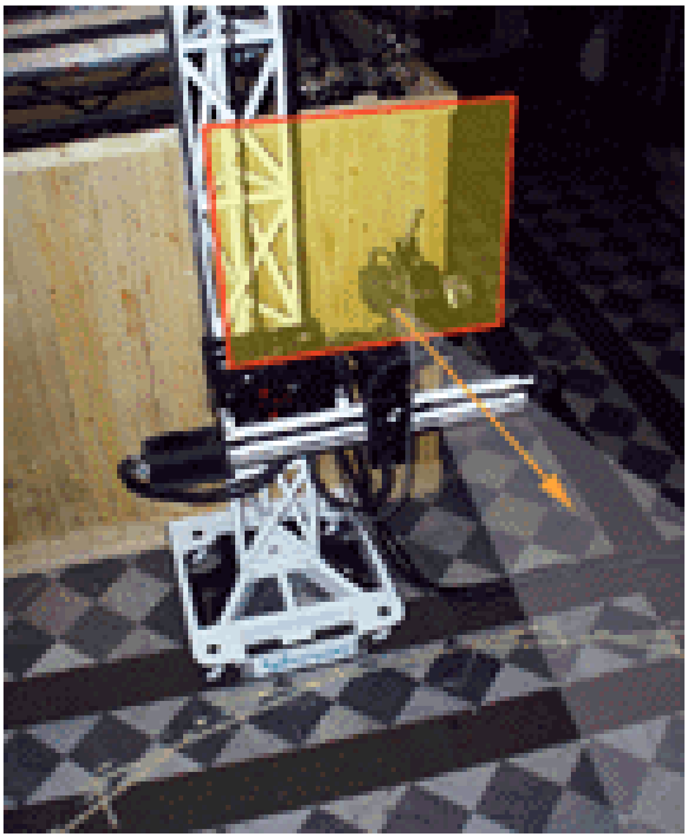
The previous setups capture 4-D light fields, but require a lot of cameras or rails. A more compact alternative is the plenoptic camera. The plenoptic camera is quite similar to a typical camera; it captures light on a traditional image sensor. The difference lies in the micro-lens array in front of the sensor. Examples of such plenoptic cameras are the Lytro [6] or the cameras designed by Raytrix [7] (photo, video and microscopy). The micro-lenses are placed in a hexagonal grid between the main lens and the image sensor. Each of the micro-lenses projects a shifted view on a small part of the sensor. It essentially creates a small image of what it sees. Figure 2.2 shows a schematic overview of this principle.

Planar light fields are not the only light fields that can be captured by hardware. There exists hardware for capturing spherical light fields. Google developed a system for working with panoramic light fields, which are sections of full spherical light fields [8]. To capture such light fields, Google modified a GoPro Odyssey Jump camera, which is an array of 16 cameras placed in a circle. The array is bent in such a way that it can rotate around the vertical axis and capture the light field. The setup Google uses is depicted in Figure 2.3. Other methods of working with spherical light fields have been proposed. Most of them build on the same principles; either a setup with multiple cameras placed on a sphere [9], or one camera which can move [10, 11].

Above solutions are all hardware technologies. Another way of capturing light fields is via software. The field of computer graphics allows us to take pictures in virtual scenes, which can then be used to construct a light field. Capturing light fields with software allows full control over the environment in which the light field is



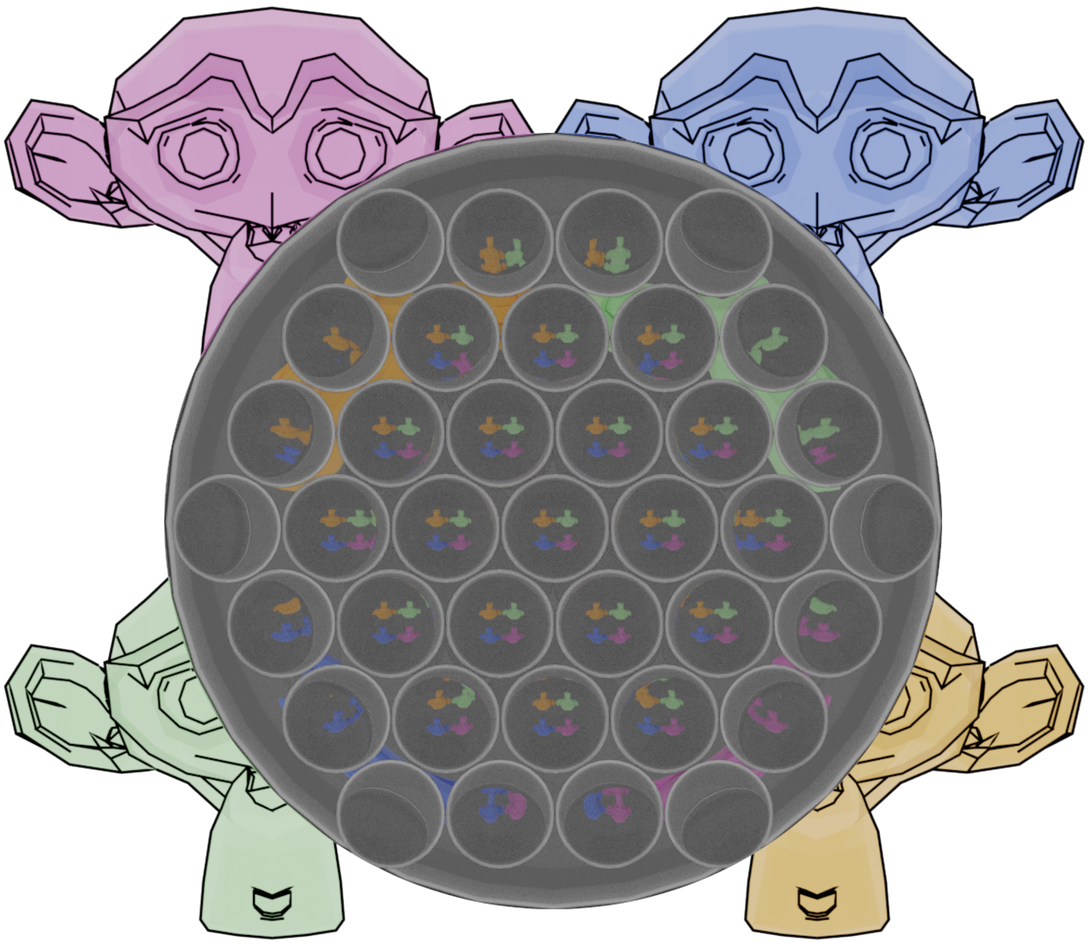
(a) *The Stanford Camera Array. The baseline between the cameras can be changed.*



(b) *The Stanford Gantry. The camera can move inside the indicated plane with sub-millimetre accuracy.*

Figure 2.1 – *Camera-based light-field capture setups.*

captured. More importantly, capturing light fields can happen in any configuration while recording exact parameters such as camera position and orientation, without the need of any kind of specialised capturing hardware. This drastically speeds up the process of research, not limited by real-world constraints. Also, the properties of the captured data do not depend on the method of capturing; once the data is



(a) *Orthographic view from the perspective of the image sensor. The hexagonal grid of lenses is very similar to the eye of a fly.*

(b) *Top view of the setup. From left to right: subject, main lens, micro-lenses and image sensor.*

Figure 2.2 – *The principle of a plenoptic camera. The micro-lenses are placed in a hexagonal grid between the image sensor and the main lens. Each micro-lens observes a slightly different section of the coloured objects through the main lens. The result of each individual micro-lens is projected on the image sensor.*

captured by software, it can be used as if captured by hardware.

In this dissertation we focus on light field capture by software. Capturing with

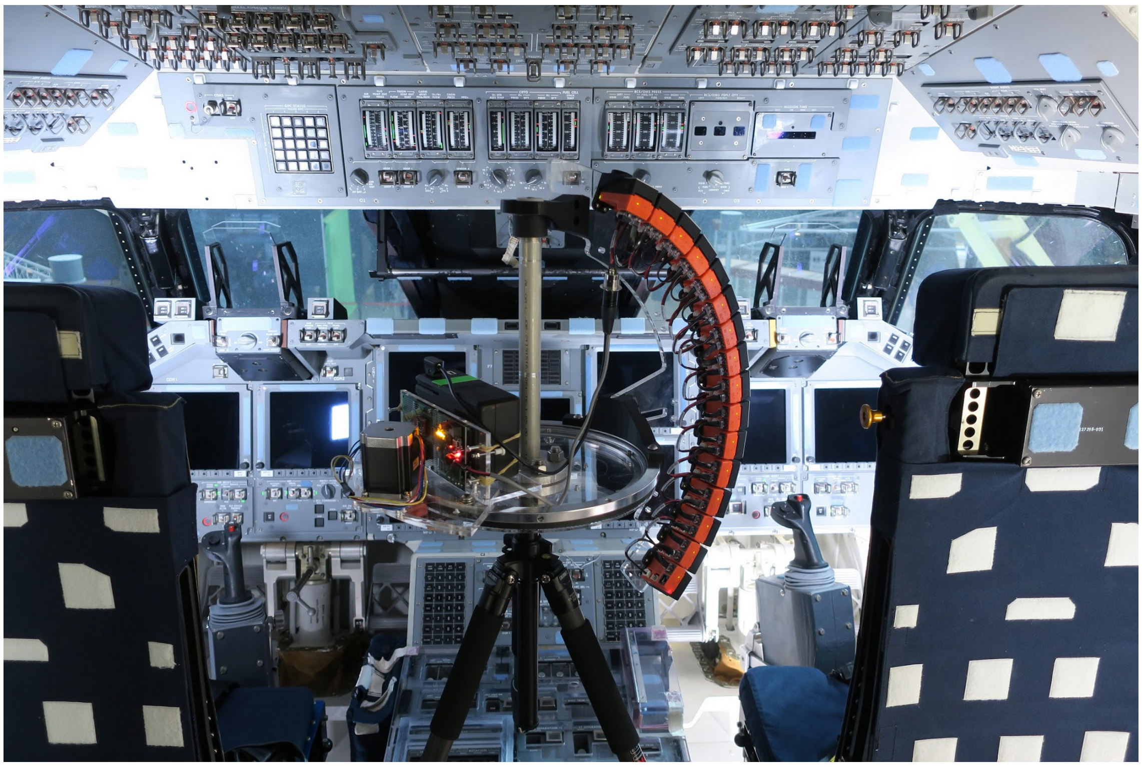


Figure 2.3 – *Google spherical light-field setup. It consists of 16 GoPros arranged on a rotating mount. This setup can capture one light-field still in about a minute.*

hardware would immensely slow down experimenting with different capture configurations, which in turn would steer away from the goals of this dissertation.

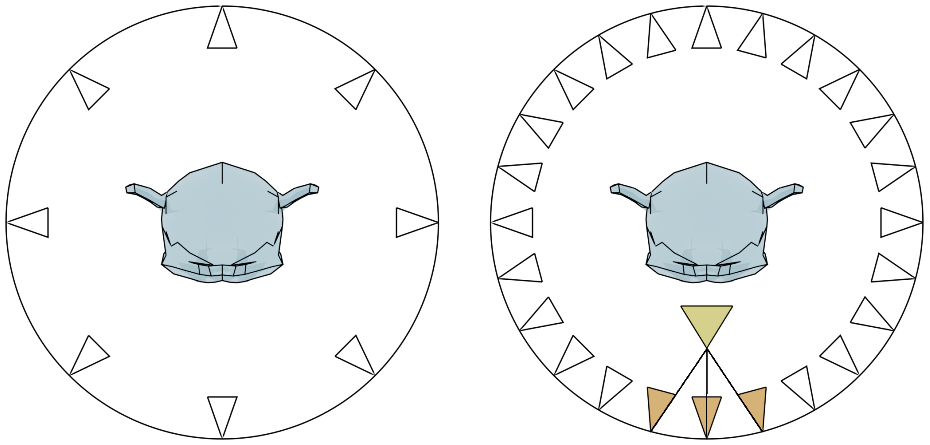
As for existing software solutions, there is Holografika, which created a script for Autodesk 3ds max that can capture planar light fields. Holografika is a Hungarian venture active in the field of photonic technologies. Katrin Honauer and Ole Johannsen made a similar script for Blender [12]. Section 2.4 will discuss this software solution more in depth, because we will use this software as comparison for our work.

2.1.2 Rendering Light Fields

Image-based rendering is a set of techniques that allow us to convey a 3-D scene to a user, solely from a set of images [1]. Those image-based rendering techniques are quite old, stemming from the 1990's. Consider a light field representation consisting of a limited set of images, as in Figure 2.4a. The different viewpoints are scarcely spaced on a circle. When the images are displayed in sequence in a rapid fashion, an observer gets the impression of moving around the object in the middle (a monkey head). This can be extended to allow a viewer to move around the object, albeit only on the predefined circle. It is simply a matter of showing the correct images to the viewer.

In Figure 2.4b, the captured viewpoints are more numerous and placed closer together; the *sampling* of the light field is different. Displaying such light field is a form of image-based rendering because the light-field data originates from images.

The novelty lies in the ability to create new views. By combining light information from the surrounding viewpoints, the yellow viewpoint can be deducted. The new viewpoint does not need to be located on the circle; it can be closer or further away. Every point and direction in space which has enough light data from nearby viewpoints can be used as a new viewpoint. Rendering a new view from any type of light field data to a user is called *light-field rendering.*



(a) *Image-based rendering. A viewer can use the different viewpoints to circle around the object, a monkey head.*

(b) *Light-field rendering. The viewpoints on the circle can now be combined to generate new viewpoints such as the yellow one.*

Figure 2.4 – *From simple image-based rendering to light-field rendering. The image is based on an image from [1].*

Light-field rendering is a difficult task for three reasons. We will explain those reasons in the context of VR because VR allows getting the most out of a light field. Firstly, light-field rendering in VR requires fast random access in the voluminous light-field data set, this for stereo views. Secondly, that large data set also needs to be downloaded and stored somewhere. Thirdly, the light field needs to be of high enough quality to come across as believable.

Very few studies have succeeded at overcoming the difficulties of speed, size and quality. One of the successful ones is the research from Google that resulted in *Welcome to Light Fields VR* [8], a VR application allowing a user to experience being in a light field. This is the same research as earlier, the spherical GoPro camera setup mentioned in Section 2.1.1. Google's light-field rendering approach can be explained as follows. A novel view is created by combining a set of disks, projected away from the observer on a captured view from the light field. Those disks were originally located on the surface of the sphere of cameras. To create denser light field data sets, Google enhances each captured view with extra geometry information, derived from a multi-view stereo depth map. The primary benefit of Google's disk based approach is the ability to independently create the disks and merge them afterwards on GPU.

Another successful method is described in Real-time Rendering with Compressed

Animated Light Fields, a work of Disney [13]. In this work, a significant reduction in size is achieved by optimising capture positions. That way, fewer camera positions are needed and hence less data. Note that a depth map is used to enhance the light field data. A novel view is rendered by ray casting and averaging samples from all views. The correct sample for each view is calculated based on the ray-depth-map intersection.

A key element that returns in the above to works is the use of geometric data or depth maps to augment the light field representation. The advantage of this has been demonstrated a long time ago [14, 15]. A subcategory of geometry-enhanced light fields are *surface light fields*, a type of light fields first described in the Lumigraph paper [16], although the paper does not mention the actual term 'surface light field'. A surface light field is rendered not by sampling directly from images, but from image data projected onto geometry; this is a key component in this dissertation.

2.1.3 Compressing Light Fields

Between capturing and rendering, light fields need an intermediate representation for storage and sampling. If it were not possible to shrink the size of an image-based light field, it would be very impractical to store light field stills and impossible to store light field movies. To store light fields, compression is necessary.

Unlike SMoE, most light field compression techniques recycle compression schemes from the 2-D image and video domain. For those schemes to work, the data is interpreted as a matrix of images, resembling a 4-D planar light field (2-D images, 2-D camera array). Joint Photographic Experts Group (JPEG) is in the progress of standardising light field coding. This standardisation attempt is part of the larger JPEG Pleno project [17], standardising not only light fields but also point clouds and holographic image data. As part of their call for proposals on light field coding, a discrete-cosine-transform (DCT) based approach was mentioned which efficiently encodes 4-D lenslet based light fields [18]. Other coding schemes involve structuring the light field data as a type of artificial video stream. The artificial video is then compressed using High Efficiency Video Coding (HEVC) [19].

The problem with video-based techniques is that they currently only work for planar light fields, not for other types. Figure 2.5 illustrates how video-based techniques encode frames based on other frames, building a hierarchy. Light fields with higher dimensions would require an even deeper hierarchy as to compress in all dimensions. This leads to even slower decoding. In Chapter 3 we will see other types of light fields which require other methods for compression.

Other techniques use geometry as part of their compression model. Moving Pictures Experts Group (MPEG) investigates a depth-map-based rendering approach, as part of the MPEG-I project (I for 'Immersive') [20]. Their technique is based on view synthesis utilising a minimal set of depth-enhanced views from the scene.

Although SMoE is amongst other things a coding framework, in which compression is (unsurprisingly) quite important, we do not elaborate more on the subject, since the focus of this dissertation is not specifically compression.

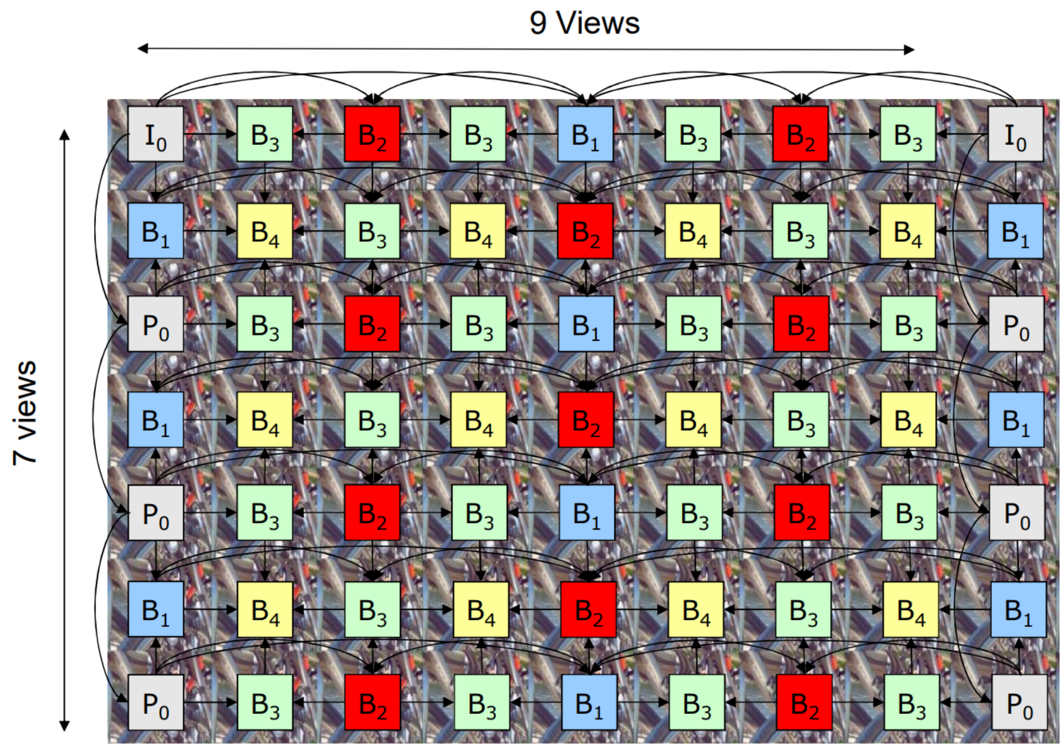

9 Views
7 views
I0 B3 B2 B3 B1 B3 B2 B3 I0
B1 B4 B3 B4 B2 B4 B3 B4 B1
P0 B3 B2 B3 B1 B3 B2 B3 P0
B1 B4 B3 B4 B2 B4 B3 B4 B1
P0 B3 B2 B3 B1 B3 B2 B3 P0
B1 B4 B3 B4 B2 B4 B3 B4 B1
P0 B3 B2 B3 B1 B3 B2 B3 P0

Figure 2.5 – *A schematic overview of light-field coding with video based techniques. Light-field views are treated as video-frames and are encoded based on other views. A B_4 frame requires decoding of a maximum of four other frames. Those frames must in their turn first be decoded based on even other frames.*

2.1.4 Conclusion

Light fields can provide an amazing solution for experiencing visually appealing VR experiences. There has been a massive effort in the domain of light field technology to push capturing, compression and rendering in the right direction. But this domain is still in its infancy at the time of writing. We believe that by combining the concept of surface light fields with SMoE, it is possible to take a leap forwards.

2.2 Steered Mixture-of-Experts

SMoE provides the full pipeline from captured data to rendering. This section discusses the current process of going from image data to rendering a 4-D light field with SMoE [3]. The modelled light fields are planar and only 4-D: the capturing viewpoints are placed on a regularly spaced rectangular grid. The objective is to familiarise the reader with the different components and steps involved, rather than diving into complicated mathematics. The complete process is depicted in Figure 2.6.

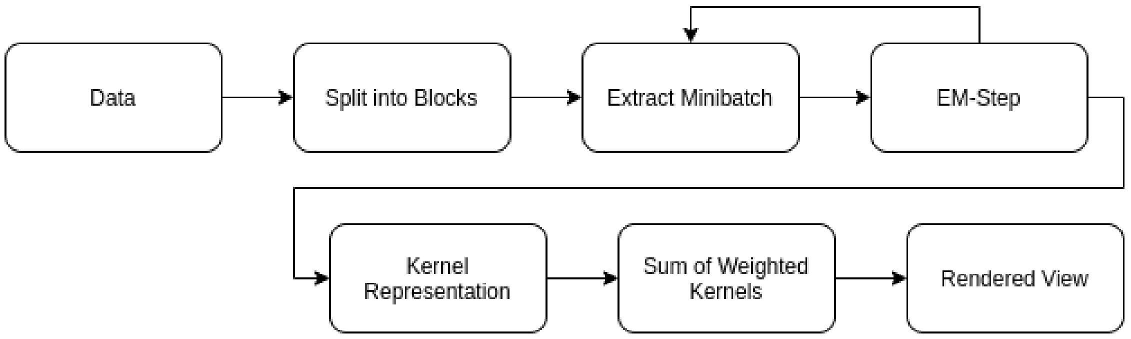


Figure 2.6 – *High-level schematic of the SMoE pipeline.*

2.2.1 Kernels

To understand the process, we first introduce the concept of kernels and why they are helpful. Natural images are composed of regions with high correlation separated by edges. Instead of representing those coherent patches using many similar pixels, each patch can be represented with the help of a single entity. Such entities are called *kernels*. Kernels conveniently can be deployed for any dimensionality; perfect for the high dimensionality of light field modalities. This is the main idea behind SMoE: represent data consisting of correlated data by a sparse set of kernels. Different types of kernels exist. SMoE uses Gaussian kernels, which are simply multivariate normal distributions.

2.2.2 Modelling using SMoE

SMoE takes as input a set of images which represent a 4-D light field. The input data is structured by camera position on the camera plane (2-D) and pixel coordinates (2-D), 4-D in total. Each camera-pixel pair has a colour value corresponding to the pixel-value of the selected camera. The colour value consists of three colour components (3-D). SMoE needs to divide the image into coherent patches in 4-D. For each of those patches it needs to find a local regressor that approximates the values of that patch. A *regressor* finds the most optimal prediction Y for a given X in the input space. The process of finding a good regressor for a given patch is known as *Mixture-of-Experts (MoE)* [21].

The segmentation of the data into patches is another part of the modelling process. Adding segmentation to MoE creates a so-called mixture model. SMoE ‘steers’ a mixture of regressors (experts) to collect as much correlation as possible in order to predict output values given any input. As mentioned earlier, SMoE uses Gaussian kernels; the mixture model is a *Gaussian Mixture Model (GMM)*. For the current representation in SMoE the GMM models 7-D data; 4-D for input data, 3-D for colours. This foreshadows how the dimensionality can spiral out of control.

GMMs display some useful properties. Firstly, they are quite easily parametrised; a GMM regressor uses only a mean and a co-variance as input. Secondly, GMMs divide the input space in smooth segments, meaning that there are no abrupt transitions. Furthermore, GMMs cause a smoothed piece-wise approximations of the input data.

This is quite useful under the assumption that natural images are actually piecewise smooth. Unfortunately that is not always the case, causing this method to have trouble with high spatial frequencies in textures for example. Lastly, the Gaussian kernels have global support in the input domain. This means that the model can be queried in every point; a kernel representing this point can always be found.

SMoE uses the Expection-Maximization (EM) algorithm to determine the most optimal kernel parameters. The EM algorithm performs small updates to the parameters in each iteration. The direction of the small updates is determined by maximising a loss using the current kernel parameters.

To increase performance, not all data is used at once [22]. Instead, the input space is divided in 4-D blocks. From each of those blocks a random sample of n values is taken, called a *microbatch*. All microbatches together form a *minibatch*. The minibatches are used for the EM algorithm. The samples are random to make sure that each minibatch forms a representation of all the data. The blocks are chosen to all be equally sized; this limits the number of free parameters when optimising the algorithm.

After performing EM, SMoE obtains a set of parametrised kernels representing the data. To summarise the modelling process of SMoE:

1. Divide input data in blocks.
2. Take a random sample of n values (a microbatch) out of each block.
3. Combine microbatches into a minibatch and use this for one EM step.
4. Repeat sampling and EM iterations until (local) optimum is found.

2.2.3 Rendering using SMoE

Each kernels also has a *gating function* or weight during modelling. To render the light field, each regressor is multiplied with its corresponding weight. The resulting values are summed and normalised, and the model is evaluated in all required input locations. The underlying mathematics is much more complex compared to how simple the process is described here, but it is left out for simplicity. We refer to [3] for more information. Rendering in the described way requires a calculation for every pixel of that view. Performing all the required computations without optimisation can take a really long time; for every pixel, all components in the model need to be evaluated. Preliminary (unpublished) work shows that optimisations are in fact possible.

Note how we do not perform any kind of view synthesis for novel view construction. New views are extracted from the modelled light field, with no direct connection to the originally captured views. SMoE provides a continuous representation, without a fixed grid of pixels. This is one of the most important properties of the SMoE framework.

2.3 Blender

Blender is a free 3D creation suite. It aims at providing a full 3D creation pipeline, which makes it also a very helpful visualisation tool. Specifically for this dissertation, it allows for creating light field data-sets without the need for special hardware.

Blender offers a very extensive *Application Programmable Interface (API)* via *Python* for creating add-ons. This will prove itself to be a useful tool in this dissertation for achieving a simplified way of light-field data-set creation.

Blender received a complete makeover mid 2019, resulting in Blender version 2.8. One of the main 2.8 features is the new EEVEE render engine. It is capable of real-time rendering while maintaining high realism. Unfortunately, add-ons created in earlier versions are not forward-compatible.

2.4 Previous Blender Light-Field Creation Add-On

This section discusses some earlier work on synthetically capturing light fields with software. The software in question is the work of Katrin Honauer and Ole Johannsen [12]. They developed a Blender add-on which is capable of adding a light field camera setup and capturing images with that setup. The add-on was created before the Blender 2.8 update; the new Blender version cannot be used.

The add-on works by adding regularly-spaced cameras into the scene. It is possible to change the *baseline* or spacing between the cameras. The setup displays a frustum, indicating the near plane, far plane and side-clipping planes. That frustum is used to visualise what the cameras will capture.

The add-on provides a single button to render the light field. When rendering, all cameras capture the scene sequentially, storing their result in the indicated directory. In 3-D artist lingo, rendering means creating an image of the scene with a camera; the term rendering is quite overused in this sense. The user has the option to also include depth or disparity maps, which store geometric information about the captured scene.

The software is quite slow, and is limited to capturing planar light fields. Because of this, and the fact that it is outdated, we will not directly use the add-on. It will serve as a basis for a new and improved method for creating light-field data, as we will see in Section 4.3.

2.5 Conclusion

In this chapter, we took a closer look at light fields, how to capture and store them, and how they can be displayed. A lot of technology already exists, but none have the required properties for real-time immersive light field experiences (through VR). As part of the next chapter, we will focus on how more imersiveness can be achieved. We briefly introduced SMoE and its process of transforming data for light field rendering. The random-access property of SMoE is needed for the real-time requirement. Of course it will be necessary to adapt the framework to the changes in the proposed methods of next chapter. Lastly, we introduced Blender and its Python API as a way of creating custom functionality, alongside an existing light-field add-on. Blender and the existing add-on provide a good starting point for creating more immersive scenes.

Chapter 3

Improving Immersiveness of Light Fields

3.1 Introduction

As discussed in Chapter 1, one of the goals of this dissertation is to improve the immersiveness of light-field applications, such as VR. Planar light fields already give a great sense of depth, but do not allow an observer to walk into and be immersed in the scene.

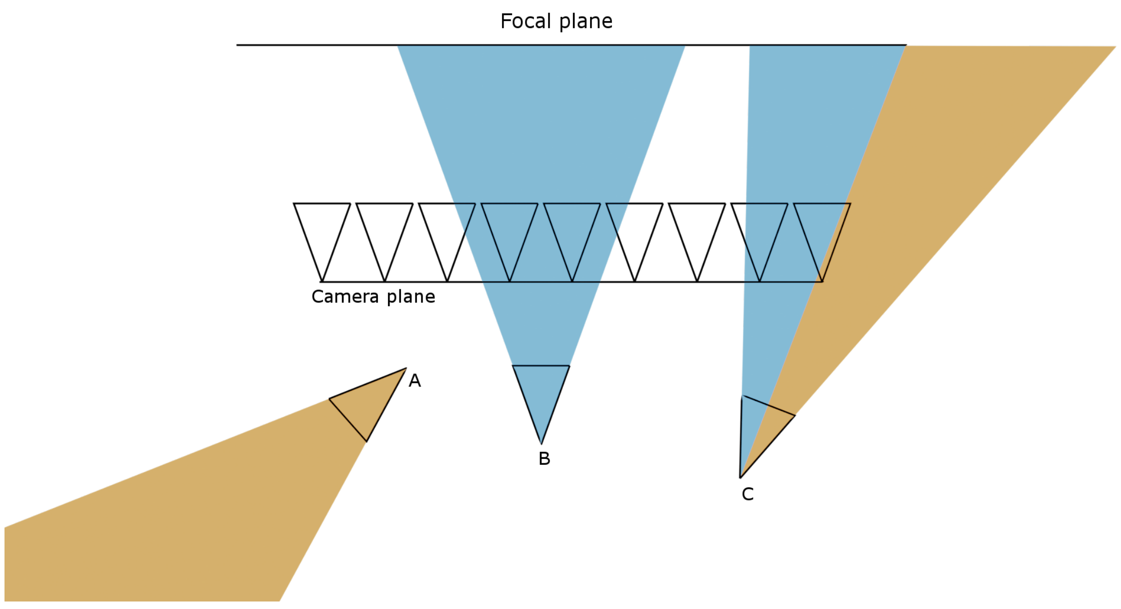


Figure 3.1 – *Planar light field with new viewpoints. While view* **B** *succeeds at rendering its view, viewpoint* **C** *is missing some light data, and viewpoint* **A** *has no data at all.*

The reason planar light fields lack this immersiveness is that the data resides on a plane and only only captures a scene in a single general direction. Figure 3.1 shows how the queried viewpoints cannot always create a full reconstruction of the original space. For illustration purposes we assume that the data is located on the focal plane of the planar light field. A new viewpoint can be created from sampling the data, see for example viewpoint **B**. Viewpoint **C** can only be partially reconstructed because the right side of its view has no corresponding data on the focal plane. When a viewer turns around such as in viewpoint **A**, there is no data at all.

The basic idea to improve immersiveness is simple: capture light from all directions. In Figure 3.1 this would allow viewpoints **A** and **C** to be completely rendered. In this chapter we explore multiple options for capturing light in all directions.

The most straight-forward solution for this problem is to add cameras in multiple directions. This is also the route we take in this dissertation.

3.2 Configurations

The question remains what the optimal configuration of cameras is to capture as much useful light as possible. An additional requirement is that the resulting data allows for efficient compression and believable reconstruction.

3.2.1 Cubical Configuration

A first solution is to create an assembly of planar light fields. Any number of planes representing light-field data could be placed in 3-D space to collectively show the light data. The more the light is capture, the more complete the light-field data will be. The problem is that a lot of the captured light becomes redundant if we simply keep adding planar light fields anywhere in arbitrary locations. A more structured setup results in more structured data which appears easier to compress. This assumption is based on the fact that compressing general light fields is more complex than planar light fields, which are highly structured [23].

By placing six planar light fields on the sides of a cuboid, light is successfully captured in all directions, see Figure 3.2. A cuboid light field encloses a space in which a viewer could move and look around. The question is if the light field should be modelled as six separate planar light fields or as one cuboid light field. The advantage of using six separate planar light fields is that the algorithms are already available; the theory of planar light fields is now simply applied on six light fields instead of one. On the other hand, there is a possibility that some artefacts might become visible at the edges and corners of the cuboid when using separate planes because the data of one light field stops and the data of another one starts. Think of the kernels in SMoE: the kernels could abruptly stop at the edges of neighbouring planar light fields. This might be very interrupting when an object spans over the edge. Modelling the light field as one cuboid light field (e.g. no separate planes) would prevent those discontinuities. This is because data on the edges and corners is modelled by a single entity, allowing more smooth transitions. The downside is that we move away from planar light fields and increase the dimensionality of the data. Also, discontinuities might still be present at the face transitions because there is

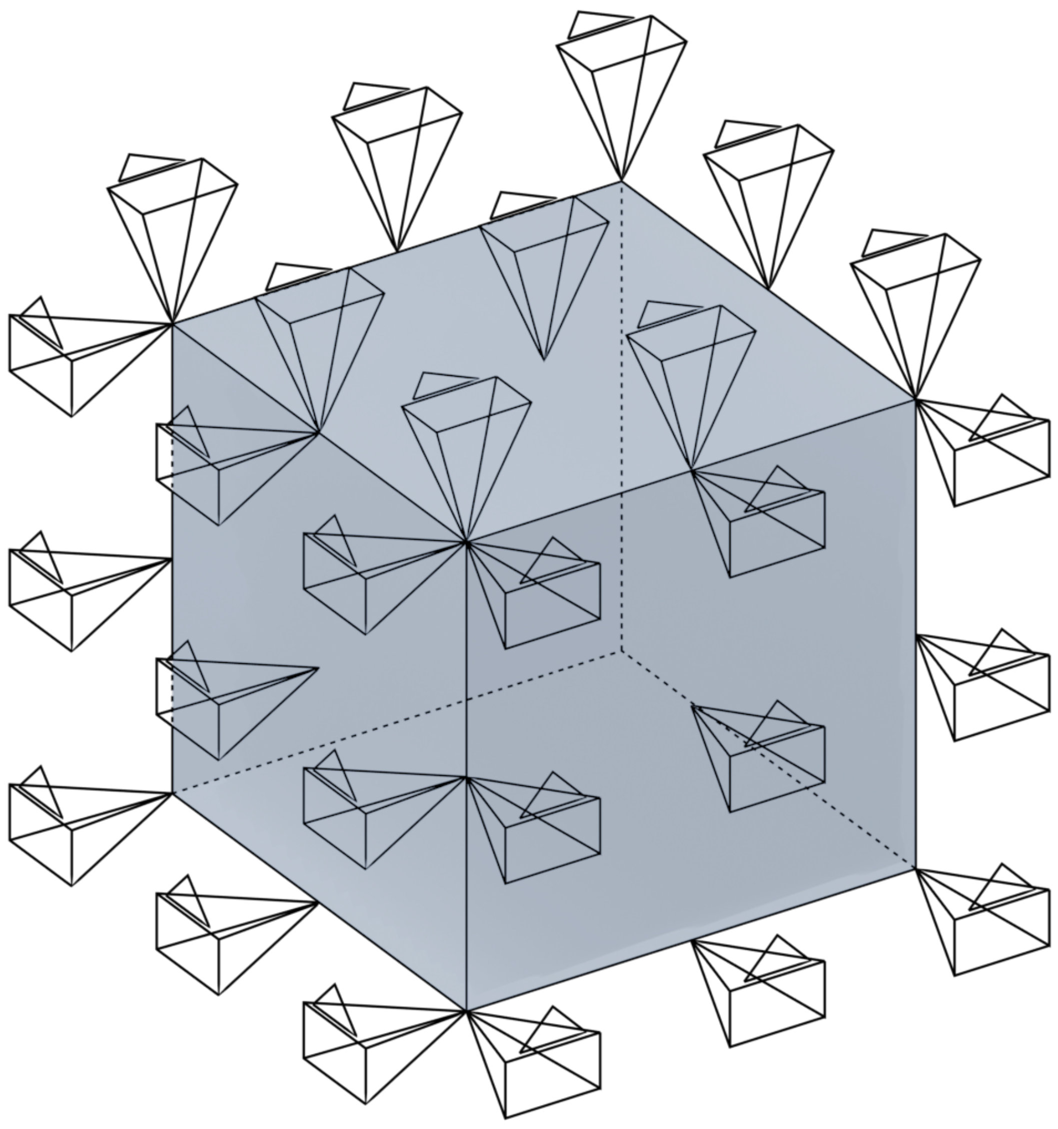


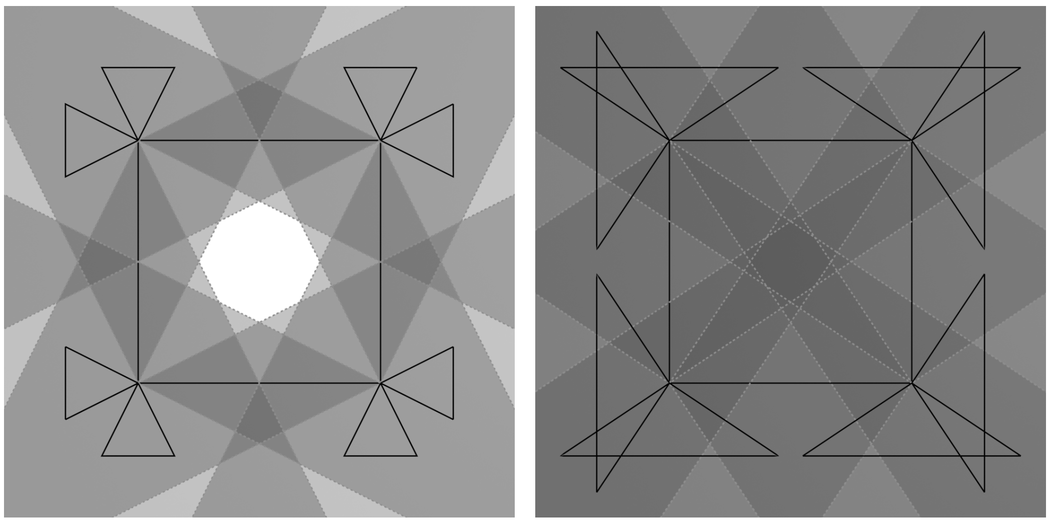
Figure 3.2 – *A cuboid light-field configuration which captures light in all directions. The cameras on the back side are not shown for illustration purposes.*

an abrupt change in camera direction.

Both options, one light field and six separate planar light fields, do not always have an area inside the cuboid in which a viewer can look in all directions without missing data. This is shown in Figure 3.3. The area is largely determined by the cameras on the edges and their field of view.

3.2.2 Round Configurations

To capture light in all directions without the problem of discontinuities, circles can be incorporated into the light-field configurations. Circles do not exhibit discontinuities. Additionally, the inner viewing area in which a VR user can look in all directions without missing data is quite elegant.



(a) *Camera overlap for small field of views (around 53 °).* **(b)** *Camera overlap for larger field of views (around 113 °).*

Figure 3.3 – *Camera overlap in the 2-D version of a cuboid. Greyer areas indicate more overlap in viewing region. When the field of view is too small, there is no area where all cameras are visible.*

The viewing area for circles is illustrated in Figure 3.4, showing a circle with cameras facing outwards. The idea is that in the green area, a viewer can look in any direction, and there will always be one or more cameras supporting their viewpoint. Outside of this viewing area, this is not always true. There are points in Figure 3.4 in which a user can only see data from one camera in the figure; a trivial example is a point right behind a camera.

The radius of the area of the green surface in Figure 3.4 depends on the field of view of the cameras, expressed as follows:

$$r_{\text{inner}} = r_{\text{capture}} \cdot \sin\left(\tfrac{1}{2}\theta_{\text{FoV}}\right).$$

where r_{inner} is the radius of the green area, r_{capture} is the radius of the circle with cameras, and θ_{FoV} is the angle representing the field of view of the cameras. The ratio of areas is then:

$$\frac{a_{\text{inner}}}{a_{\text{capture}}} = \sin^2(\tfrac{1}{2}\theta_{\text{FoV}})$$

This property does not hold if not all cameras have the same field of view. In Figure 3.5 the relation between field of view and the area ratio is plotted. This shows that small field of views do not provide a large inner area. Field of views larger than around 150 degrees don't provide significant gain in area, but might be useful when the available space for the capture setup is limited.

Two possible configurations that contain circles are spheres and cylinders. A cylindrical light field configuration is shown in Figure 3.6. Using a cylindrical light field avoids discontinuities in horizontal planes, but the configuration still has discontinuities at the top and bottom edges. This might not be a problem for certain types of content; for example, when the main content is not located above or below the

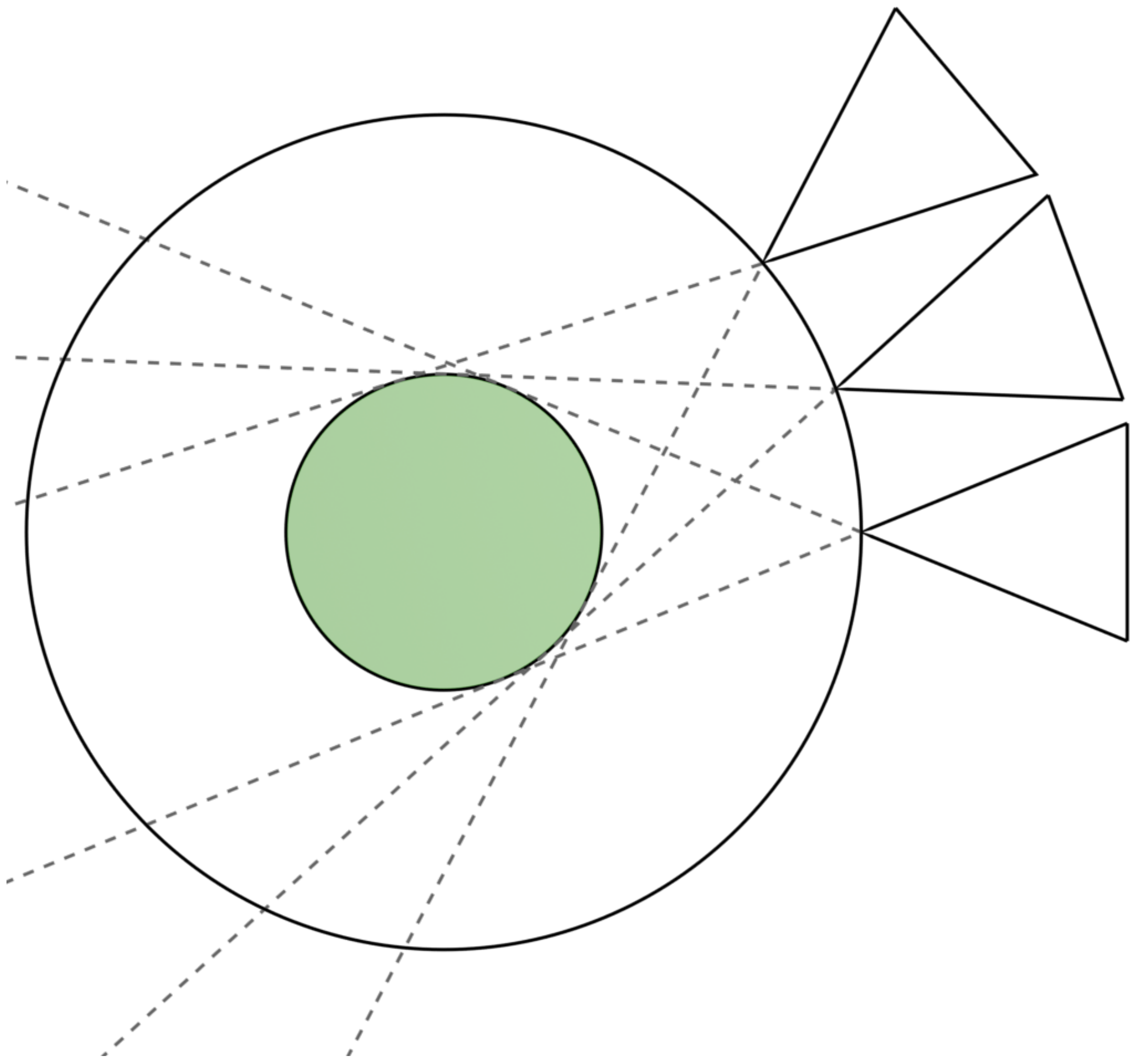


Figure 3.4 – *When cameras are placed in a circle formation, there exists an inside viewing area, in which the user does not have limited vision in any point or orientation.*

user. Spherical light fields do not have discontinuities at all (no corners or edges) and might therefore result in even more immersiveness.

The spherical light field configuration is shown in Figure 3.7, where we see that the cameras are placed on almost evenly distributed points. There are many ways of distributing point on a sphere, but in this case the cameras are placed on the vertices of a geodesic polyhedron. Geodesic polyhedrons are used because they are very convenient to work with in Blender.

The inner volume in which a user can move around in a cylindrical configuration is a combination of the circle inner area and rectangle inner area from Figure 3.3. It is the intersection of the cylinder formed by extending the inner area of a circle to a cylinder, and the volume created by rotating the inner area of a rectangle around

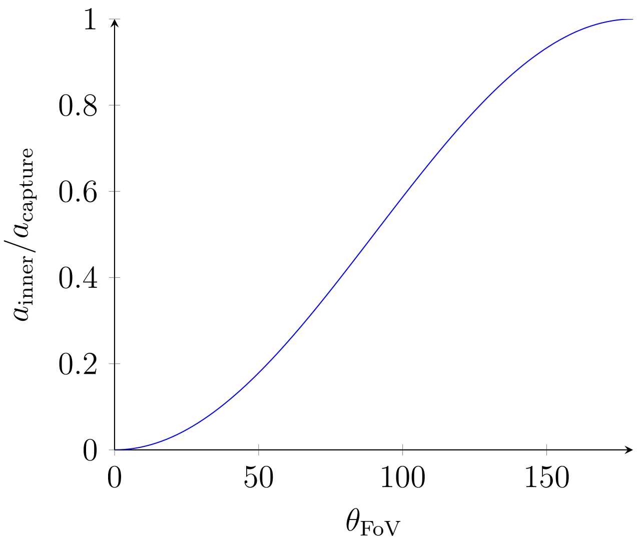


Figure 3.5 – *Ration between inner area and capture area in function of field of view.*

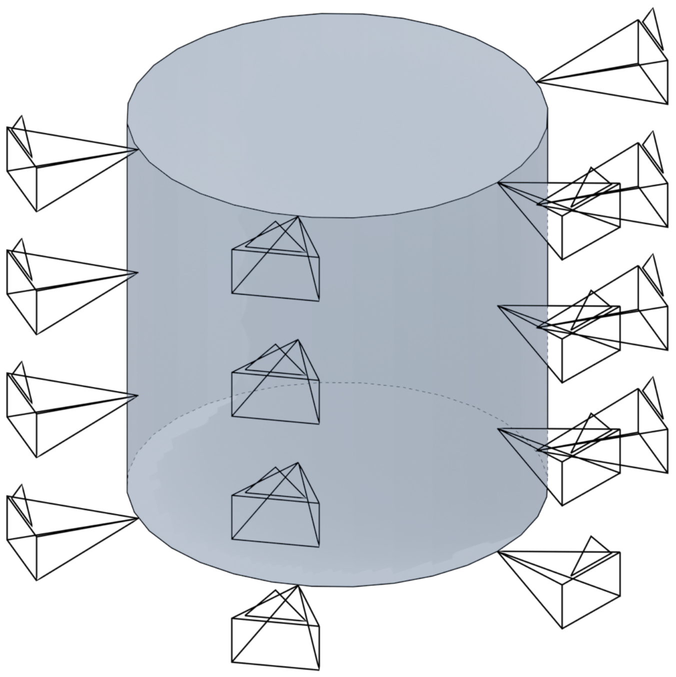


Figure 3.6 – *The cylindrical light field configuration. The cameras are placed on equally spaced rings on the cylinder. Viewpoints on the back are not shown.*

the height axis. There is simply no vision towards top and bottom if cameras are not placed on top and bottom sides. The inner volume of a sphere is also a sphere, and all frustum planes of all cameras are tangent planes to that inner sphere. This is very analogous to circles. When the aspect ratio of the viewpoints are different from one, the inner volumes of the configurations become more complex and non-trivial.

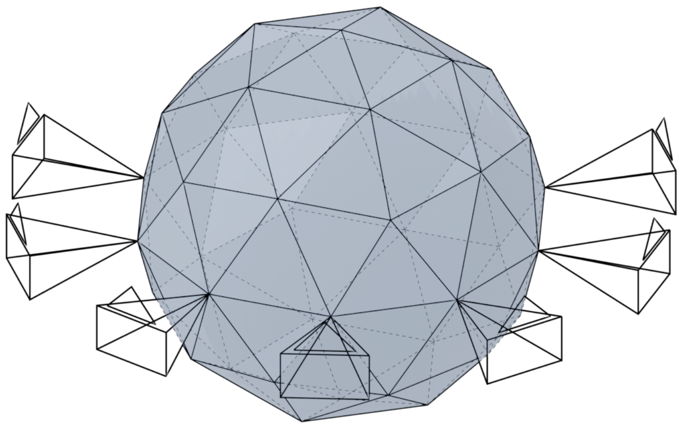


Figure 3.7 – *The spherical light field setup. Only viewpoints around the centre are shown. Each line intersection in the figure represents a camera position.*

3.2.3 Problems with Round Configurations

Although cylinders and spheres currently look very attractive, they also have limitations. Compared to planar light fields, the data no longer resides on a plane and is instead curved. This breaks existing techniques. Video-based codecs could possibly handle this by simply changing the way in which viewpoints are calculated from each other (see Section 2.1.3). But this is not as straightforward for SMoE. Remember from Section 1.1 that data in EPI-strips for planar light fields is mostly linear. The Gaussian kernels currently used in SMoE are linear, which works well with the linearity of that data. If the data is not linear, multiple kernels must be used for features that would be linear for planar light fields.

In the following we will show that for viewpoints on a circle, the EPI-strips are not linear. To do this, we want to find the relation between the position of a camera on a circle and the position of an object on the camera sensor (or the position in the output image). In Figure 3.8 this setup is shown, with a circle in the origin and an object on the x-axis. The camera lies on the circle, pointing outwards from the centre. This means that the projection plane is perpendicular to that direction, as shown in the image. It is important to see that choosing the radius of the circle as 1 and choosing the object on the x-axis can be extended to the general case by simple transformations. To find the position of the object on the sensor, the length k must be found, or distance from the centre of the image. As we are only looking for a relation, and not a function, it can be assumed that k is the tangent of φ; this indicates that the projection plane is at distance 1 from the camera. Variable φ is the angle between the camera direction and the direction from the camera to the

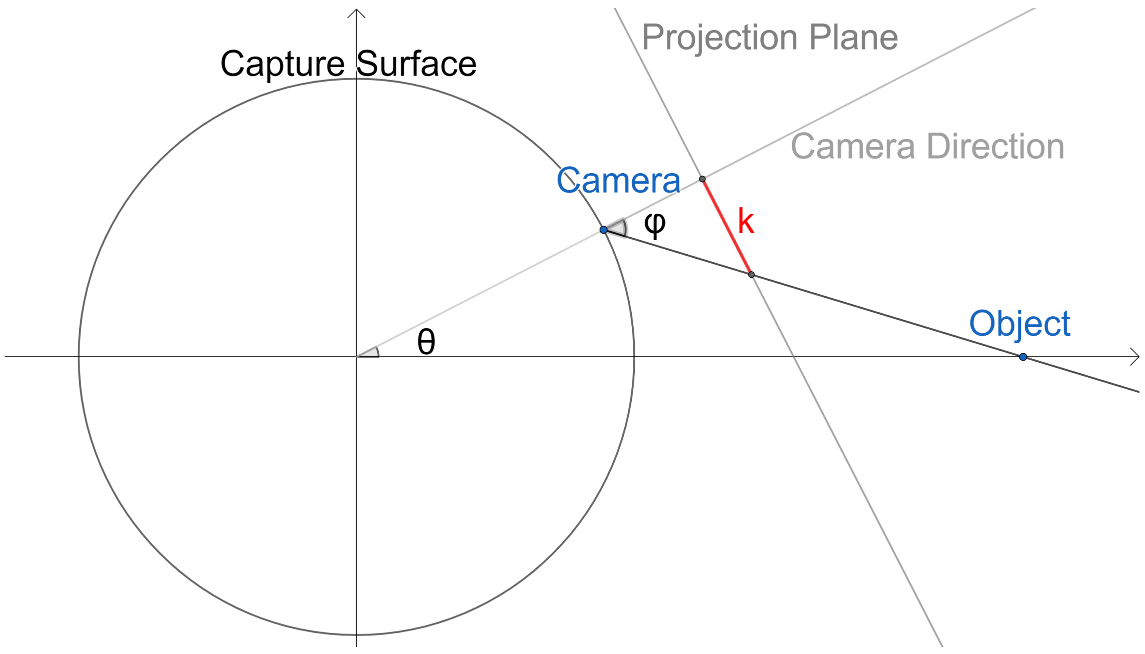


Figure 3.8 – *Schematic of the relation between an object and its position in an image. k is the distance from the centre of the image to the object in the image.*

object. We have:

$$k \propto \tan \varphi$$

Also, with $\vec{c}$ the vector from the origin to the camera, and $\vec{o}$ the vector from the camera to the object:

$$\varphi = \arccos \frac{\vec{c} \cdot \vec{o}}{|\vec{c}| \cdot |\vec{o}|}$$

The object cannot be projected onto the projection plane for θ larger than a certain angle because the object is behind the camera. The maximum value is given in function of o_x as follows:

$$\theta_{\max} = \arccos \frac{1}{o_x}.$$

Together with the following relationship,

$$\tan(\arccos x) = \frac{\sqrt{1 - x^2}}{x},$$

and $o_y = 0$, $\tan(\varphi)$ can be simplified. We get the following relation:

$$k(\theta, o_x) = \frac{o_x |\sin \theta|}{o_x \cos \theta - 1} \text{ for } \theta < \theta_{\max}.$$

Solving for a specific o_x results in the EPI-strip for an object at that distance.

Although the relation is obviously not linear, it is approximately linear for small θ and larger o_x. This can be seen in Figure 3.9, where k is plotted in function of o_x and θ. The angle θ is only plotted for values where the object can be projected onto

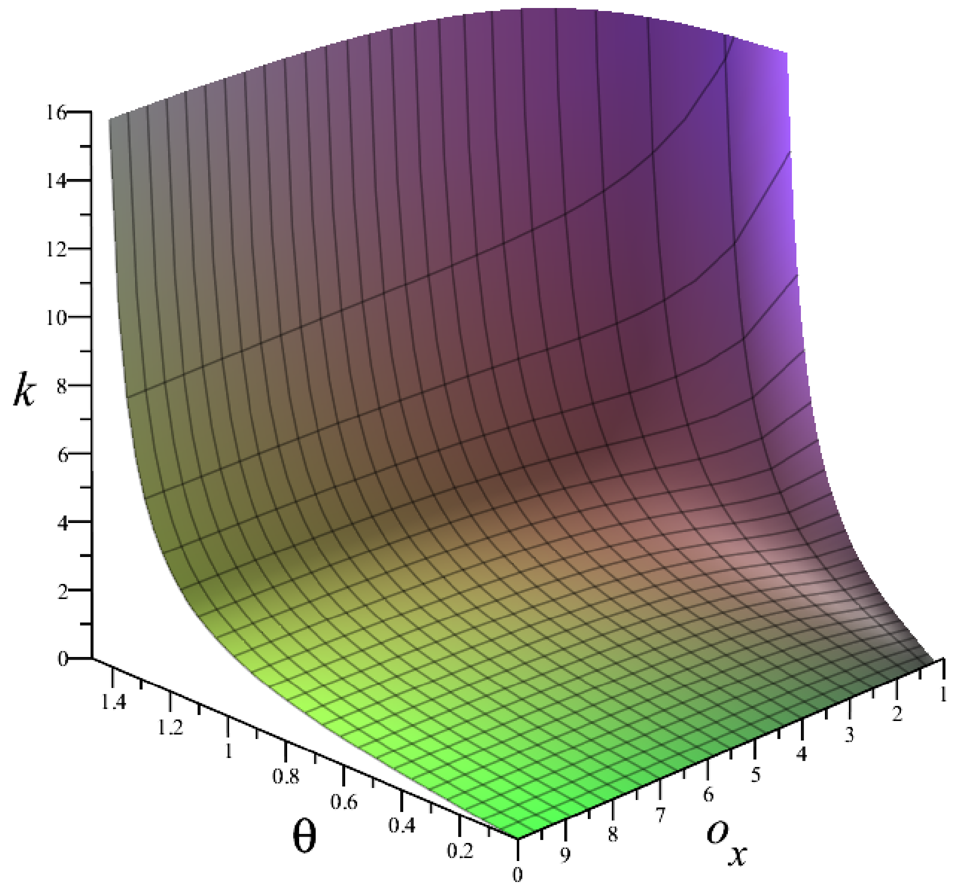


Figure 3.9 – *k in function of θ and o_x. Notice how the green area is almost linear. For large θ or objects close to the capture surface (small o_x), the relation is no longer approximately linear, but approaches infinity.*

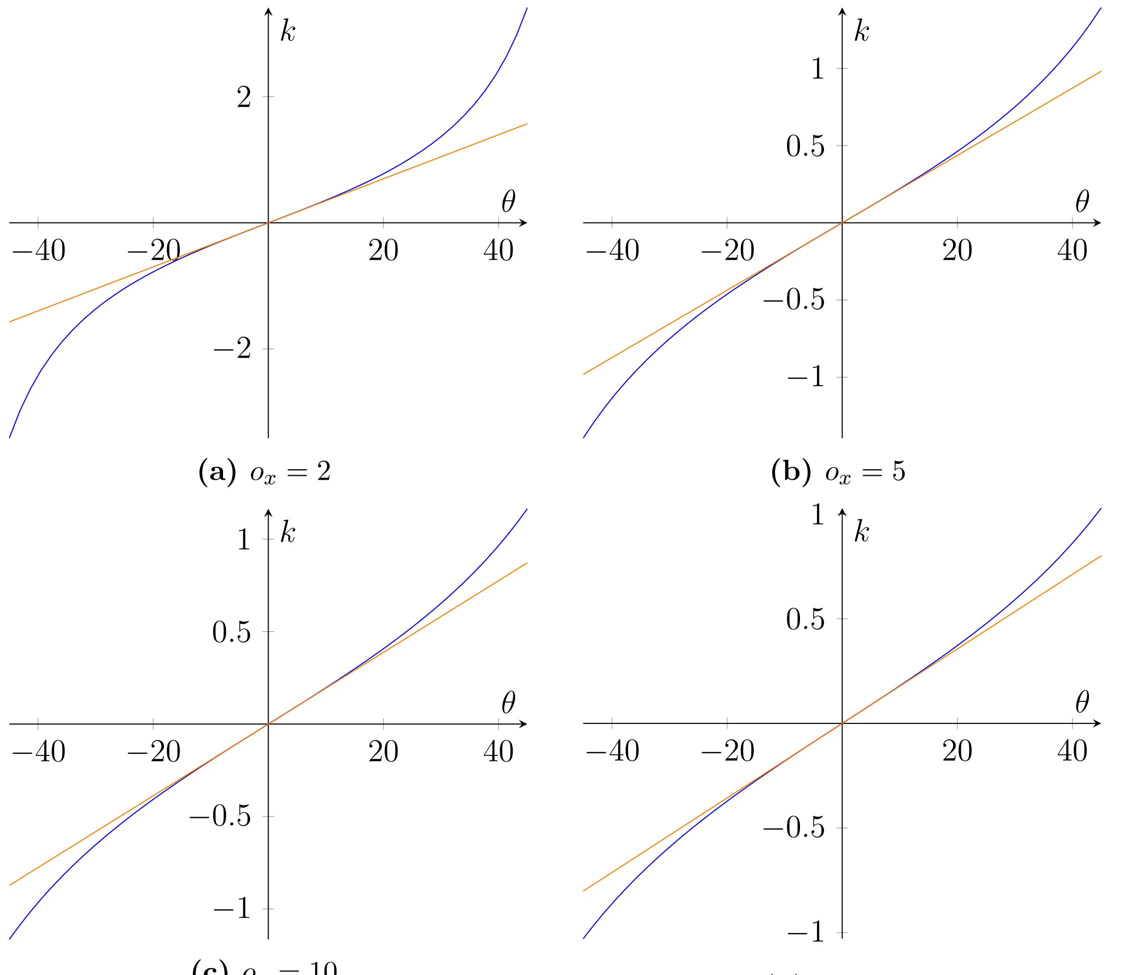
the projection plane, $\theta < \theta_{\max}$. Angles greater than the maximum result in negative k ($o_x \cos\theta - 1 < 0$ in the denominator). When θ approaches its limit, k approaches infinity. That is why the graph is truncated at $k = 16$, corresponding to a field of view of 173 °, which is already quite large.

Also notice that the values for θ start at zero. The situation with negative angles is symmetrical, and the plot is extended by mirroring the graph around the o_x-axis. This results in negative values for k, representing the image on the other side of the viewing direction.

The green area approximates a plane for relatively small angles θ. One might suspect this is because of the tangent in the calculation, with $\tan(\varphi) \approx \varphi$. The partial derivative of k with respect to θ in $\theta = 0$ determines the slope of the linear function approximating k for each distance o_x. The derivative evaluates to

$$\frac{\partial k}{\partial \theta}(0, o_x) = \frac{o_x}{o_x - 1}$$

in $theta = 0$. Figure 3.10 compares k and its approximation in several values of o_x.



(a) $o_x = 2$ **(b)** $o_x = 5$

(c) $o_x = 10$ **(d)** $o_x = 50$

Figure 3.10 – *Comparison of k and its linear approximation for several values of o_x. Objects further away (larger o_x) result in better approximations.*

This means that, even though not linear, cylinders and spheres could be approximated as linear for smaller theta, as long as the object is not too close, relative to the radius of the sphere. If k is limited to values smaller than one, the result might actually be usable. The restriction $k < 1$ results in field of views smaller than 90 °. We conclude that, although the data is not linear, round configurations might yet be used for those field of views. Consequently, cylindrical and spherical configurations can be transformed to planar light fields and SMoE can model their data. The mantle of the cylindrical light-field configuration could be unwrapped and modelled as if it was a plane; with possibly some artefacts at the line where it wraps around. Spheres might be harder to unwrap, because of the way the cameras are spread and the decision of the type of projection. The idea that this linear approximation can be used for SMoE is to be confirmed in future work.

3.3 Surface Light Fields

Although previous proposals for improving the immersiveness of light-field experiences are somewhat promising, they do not allow for complete immersion; the user is still restricted in location or rotation, they cannot move outside the camera configuration without losing (partial) sight. For example, in the previously proposed configurations it is not possible to capture light from the backside of objects. Along with this problem of depth-based occlusions, there is also the problem of the perspective projection, specifically for SMoE. Remember from Chapter 1 that the perspective projection introduces non-linearity in features lying across the depth axis. Those features are not simple to model with the SMoE framework.

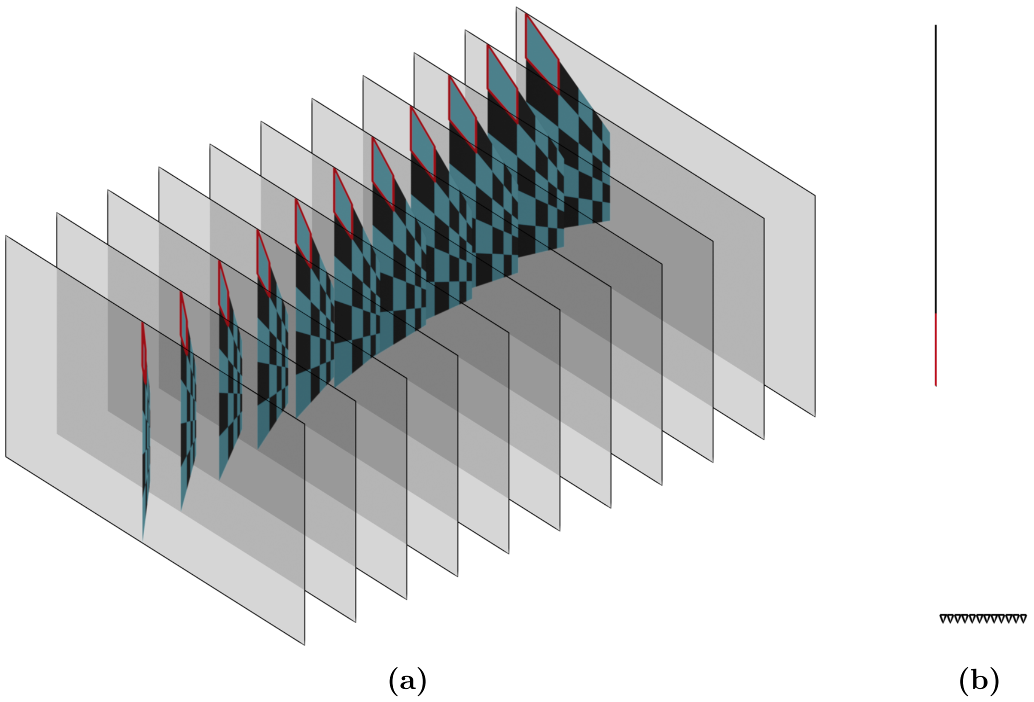


Figure 3.11 – *Figure 3.11b shows a top view of the setup. An array of cameras captures a plane aligned with the depth axis of all cameras. The result of each viewpoint is shown in Figure 3.11a. Both figures show the red rectangle highlighting one rectangle of the blue-gray grid. The volume encapsulated by the red highlights cannot be represented by a single kernel since it is not linear.*

This is illustrated in Figure 3.11 with only a single row of cameras. The scene contains a plane with a blue-black grid texture. The plane is conveniently aligned along the depth axis. A single rectangle is highlighted to emphasise the effect. Figure 3.11b shows a top view of the setup, while Figure 3.11a shows the resulting images stacked in order; the foremost image originates from the leftmost camera.

The red outlines change shape in each consecutive view in Figure 3.11a. The left and right edge of the red rectangle (or any other rectangle) does not change in size or slope, only in position. This change is linear through the stack, which is no problem. The top and bottom edges tell a different story. Similar to the left and right edges, their position changes linearly. Contrarily, the size and slope of the top and bottom edges do change. The change of slope and size is due to the features being oriented along non-constant depth in the scene (i.e. the grid is not parallel to the camera plane). Therefore it is impossible to accurately model such non-constant depth features with a single Gaussian kernel.

To solve these problems, the light field could be modelled on the surfaces the light rays originates from. Such a light field is called a *surface light field* [23]. Surface light fields are possible under the assumption that the perceived light along any point on a ray is the same in the direction of that ray. As discussed before in Section 2.1, this assumption is not physically correct but is in most cases (approximately) valid (e.g. fog).

Surface light fields incorporate the perspective projection of the capturing cameras into the model; the perspective projection is reverted by shifting the captured pixels back to the originating objects. This shift happens along the line defined by the camera position and original light ray direction. Be aware that, when using the data, one needs to account for the fact that the modelled data is is not projected data (which is the case for planar light fields). The projection for display must be performed during rendering.

The second problem that surface light fields solve is the problem of depth-based occlusion. Light can now be captured from any surface in any direction. A viewer is no longer restricted in movement when wandering through such light field, which was the case for planar light fields. They navigate through the geometry, not around a plane.

In addition to solving the aforementioned problems, surface light fields provide additional benefits. Firstly, the global support of the Gaussian kernels in SMoE can be limited to the underlying geometry. Having global support means that the gating function (see Section 2.2.3) never evaluates to zero. This is actually a good thing, because every point will have a value when rendering, no matter how far away from any kernel. The problem is that a pixel is thus also influenced by the most unrelated kernel to that pixel. Surface light fields can partially mitigate this problem. The support of a kernel can be limited to the surface of its parent object. It is preferred to keep material properties contained to an object instead of leaking to other objects; mixing within an object might come across as less uncanny than mixing two separate objects (e.g. a steel rod and a teddy bear). The downside of this property is that it requires kernels to be assigned to distinct pieces of geometry; that does not happen automatically. Consequently, this is only possible if the assigning algorithm has access to the individual pieces of geometry.

Secondly, with the added information of geometry, material properties can be extrapolated. Imagine a red sphere with a complete matte surface, or a diffusely reflecting surface. Such a surface exhibits *Lambertian reflectance*. As a consequence, when the point of observation changes, the colour and lighting of any point on the surface does not change. The lighting is independent of the viewing direction, because there is no specular highlight which depends on the viewing angle. Of course such surfaces are almost impossible to produce in reality [24], but we can learn from their properties. If during capturing of a surface light field some areas where not covered enough to reconstruct a view, treating the under-represented surface as having Lambertian reflectance might cover up its lack of data. In worst case scenario when no light was captured at all, material properties can be extrapolated from other parts of the object. Although the lighting in those areas will not be the same as when captured, it can be thought of as a form of graceful degradation. Extrapolation of views is already possible in the original SMoE implementation [3], but it is quite limited (only in the corners of light-field images) and cannot reconstruct the backside of an object (there are no objects).

Thirdly, it is even easier to find correlation now that the data is distributed in 3-D space; proximate points are even more probable to belong together and points far away from each other probably do not belong together.

Until now it was silently assumed the geometry of a scene was available. That is of course not true when taking real world pictures. Surface light fields do need a geometry to place the data on. There are several options for extracting that geometry. A solution could be to use stereo views for depth extraction [25], or use specific hardware such as a LiDAR [26]. Those techniques are often error prone due to noise in physical data. The reason this is not such a big problem is that the geometry only needs to be approximate. When a point is visible, it is only visible in the direction the user is looking at. Therefore, the depth of the data point when it is visible must not be perfect; it will be perceived as if it was on the geometry. When surface light fields are evaluated in Chapter 5, this will be qualitatively confirmed.

The following section discusses the implications of surface light fields for SMoE; how it is different from planar light fields and the challenges.

3.4 Surface light fields for SMoE

Surface light fields have specific properties when it comes to using SMoE. Firstly, the current implementation of the SMoE framework is designed specifically for planar light fields and therefore 4-D data. The data of surface light fields is five-dimensional, because a data point has three location coordinates and two angles for the direction. Changing the dimensionality should be quite straightforward. As mentioned in Section 2.2, the SMoE kernels can be of any dimension while the EM algorithm still works. The reason is that we want to avoid discontinuities in angles. Using vectors instead of angles to represent the light direction in a data point changes the 2-D directional component to 3-D coordinates on the unit sphere, while removing the problem of discontinuities. The dimensionality remains at five in theory, since the length of the direction vectors is always one (one less degree of freedom), but for the EM algorithm this will be six, independent from theory.

A second difference is that the data is now not nicely placed on a plane, or in linear strokes in four dimensions. Consequently, there is no longer such a thing as EPI-strips or linearity due to the configuration of the cameras. All the data is located on the surface of the geometry of the captured light field. This means that any occurrence of linearity must be part of the surface lighting. A sphere in planar light fields could be modelled with a single kernel, while the same sphere has a curved surface in a surface light field, and needs multiple kernels to represent the same object. This lack of artificial linearity might cause issues when compressing surface light fields with SMoE. Chances are that the compression rates are not nearly enough to be viable. Features that would be efficiently compressible in planar light fields might need a considerable amount of kernels to achieve similar rates. This is to be determined by further research, but falls outside the scope of this dissertation.

Thirdly, surface-light-field data is quite sparse compared to planar light fields. Planar light fields have no gaps in the data at all; there is a value for every camera position and every pixel position. Surface light fields have a relatively small amount of 3-D locations that are filled. For each of those locations, only a part of light directions are covered. This poses a problem for SMoE when it creates minibatches. Dividing the 6-D space into equal sized blocks results in poorly distributed batches. Since division into minibatches and randomising of samples is quite important (see Section 2.2.2), we need a way of performing a similar operation.

Finally, another difference between planar light fields and surface light field is their rendering. The naive approach is to use a ray-casting algorithm and sample the model at the intersections of the rays with the surface. The parent geometry also needs to be determined when the kernels are limited to that geometry. Although this is the mathematically accurate approach, surface light fields require more calculation and possibly complex look-ups. Finding the object intersections and correctly sampling the kernels could prove to be quite complex. Therefore, the question remains how the process of rendering surface light fields can be optimised. Surface-light-field-rendering falls out of scope for this dissertation.

3.5 Conclusion

The goal in this chapter was to explore the available options for increasing immersiveness of light fields, discussed in Section 3.1. Firstly, cuboid light fields are an intuitive continuation of planar light fields which allow the ability to capture light in all directions. This type of light field looks like it might create artefacts at the edges, both when modelling as one light field, or 6 planar light fields as the side of a cuboid.

To counter this, we reviewed the option of cylindrical and spherical light fields in Section 3.2. Both present interesting properties due to the discussed properties of circles. Although they might allow for quite intuitive compression with more immersion, the goal is to achieve full immersion.

The option that grants the ability to capture a full light field with full immersiveness is surface light fields, discussed in Section 3.3. The section explained why surface light fields are preferred and which problems they solve. Surface light fields are continued in Section 3.4, discussing what implications they have on the SMoE framework. This is important, because the remainder of this dissertation focusses on surface light fields, and how SMoE will operate on them. An overview of the different options and their properties is shown in Table 3.1.

	Cuboid	Round	Surface
Dimensionality	6x 4-D / 1x 6-D	6-D / 4-D as plane	6-D
Linearity	Yes, except along depth	Approximately, for objects further away and small FoV	Only when geometry allows it
Sparse data	Only when modelled in 6-D	Only when modelled in 6-D	Yes
Edge artefacts	At edges	At edges and wrapping line of cylinder when modelled in 4-D	Unlikely
Immersiveness	Yes but limited in space	Yes but limited in space	Yes
Inside viewing area	Complex, small	Elegant, larger	Everywhere
Complexity	Low / already solved for 6 separate planes	Higher, but can possibly be reduced to planar light field by unwrapping and modelling in 4-D	High

Table 3.1 – *An overview of the different approaches for improving immersiveness, each with their advantages and disadvantages.*

Chapter 4

Implementation

4.1 Introduction

In the previous chapter we decided on using surface light fields as a solution to our problem of immersiveness in light field technology. The main reason for using surface light fields was that they allow a viewer to walk around in the light field and observe it in all directions. Unfortunately, SMoE does not operate on surface light fields yet. This chapter explains what needs to change in practise, and discusses the implementation of the first components of the modelling process.

There are few light-field data sets which can serve as data for a surface light field. It is possible to convert planar light fields to a simple surface light field by changing the depth of the data points, but this depth is not always available. For that reason (lack of data), this dissertation contributes a novel method for creating light-field data sets. The provided method can also be used in future research for other types of light fields.

Figure 4.1 shows the same pipeline as in Section 2.2. The blue components indicate in which steps of the SMoE pipeline modifications are required. This dissertation contributes to the first two, data and preprocessing. The third part, or the rendering stage, is not implemented in this dissertation.

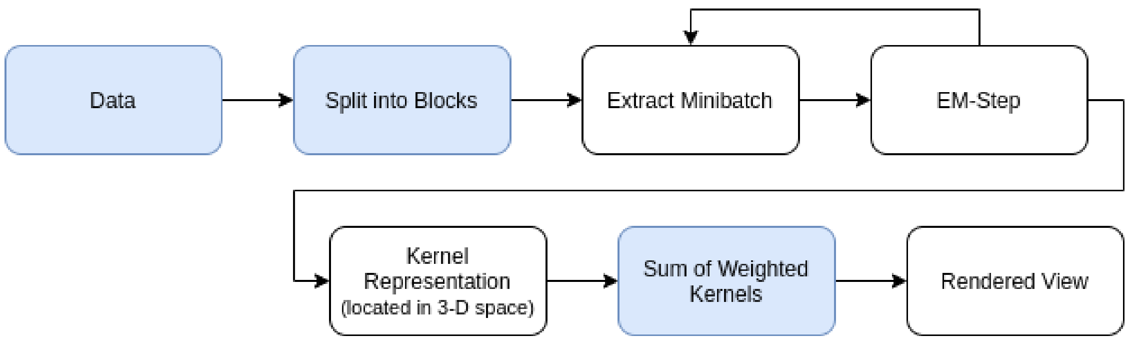


Figure 4.1 – *High-level schematic of the SMoE pipeline. The blue components require modifications to function with surface light fields.*

4.2 Preprocessor

We start at the beginning of the pipeline: the data representation. The light information of surface light fields does not reside on a plane but in 3-D space. Each data sample has three coordinates, x, y and z, together with a light direction. The light direction can be represented by only two coordinates, a horizontal and a vertical angle. In our representation we choose to use three components and make it a unit vector. The reason for this is to prevent discontinuities in angles and avoid gimbal lock, as discussed in Section 3.4. As a result, the data is six-dimensional. Another strategy could be to keep the data at the location of the cameras, and append the depth to the data point instead of displacing the point. This essentially represents the same data, but now stored in another location in space. The advantage is that the preprocessor would save in computations because it does not have to evaluate the actual location of the point. A problem is that points which appear close to each other in coordinates can appear far away when looking at the coordinate values and vice-versa. To use that data with SMoE, the actual location would need to be evaluated or a special kind of distance metric must be used, because SMoE exploits correlation of local data. For this reason, data on the surface (i.e. the depth is incorporated) is preferred over depth-augmented data (i.e. the depth serves as an extra coordinate).

The generation of data for surface light fields will be discussed later in Section 4.3. The generated data consists of images (light data) with corresponding depth maps. The remainder of this section shows how this data is processed in preparation for modelling with SMoE.

4.2.1 Data Preprocessing

The goal is to transform image data to spatial light data. To do this, the images are projected back in space. Remember that an image is a perspective projection of what a camera sees. To go back to world space, the perspective projection is undone. This process is shown in Figure 4.2. The starting image is shown in Figure 4.2a. Each pixel has a coordinate in *screen space*, depending on the image resolution. The screen space is normalised in both axis to a $[-1, 1]$ range; the image resolution does not matter anymore. This normalised space is called *clipping space*, see Figure 4.2b.

The next step is placing everything back to how the camera observed it in 3-D space. This is called *view space* and is visualised in Figure 4.2c. Typically, the camera is facing along the negative z-axis. Extra geometry information must be available because all original depth information was lost in the perspective projection. Depth maps are used as geometry information. For each pixel a depth-lookup is performed in that depth map. The pixel is then moved away from the camera with a distance corresponding to that depth. The direction in which the pixel moves depends on the *projection matrix*; the projection matrix was originally used to transform from view space to clip space. Each pixel is augmented with a direction vector pointing to the camera. That direction vector constitutes the light direction of that pixel.

Caution is needed with the type of depth that is dealt with. Cycles, the ray tracing engine of Blender, outputs depth maps as euclidean distance. On the other hand, Blender's EEVEE outputs depth as the z-coordinate in view space. Linear depth is the distance from the camera to a point, parallel with the depth axis. The depth type can be specified as command line argument.

Finally, all data points are transformed to *world space*, see Figure 4.2d. This operation requires the camera position and orientation.

The above operations are part of the preprocessor script in a Python. Numpy is used to execute all transformations as efficient matrix multiplications, with the help of Basic Linear Algebra Subprograms (BLAS). Efficiency is essential due to the size of light fields.

The end result of this step is a long array of 9-D elements:

- (3-D) Three location dimensions, indicating the position of the data point in space.
- (3-D) Three direction components, forming a normalised direction vector.
- (3-D) Three colour dimensions, red, green and blue.

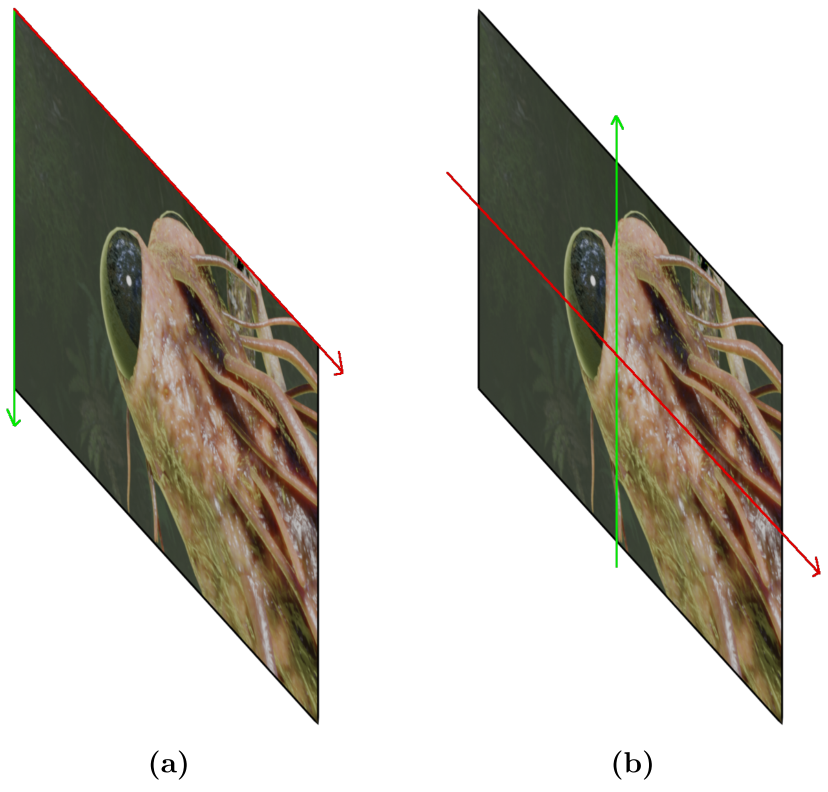
The direction vector has three dimensions and not two. Two angles is actually enough, but as mentioned before we can run into discontinuities or the gimbal-lock problem. Working with an extra dimension solves this problem. The direction vector also indicates the outgoing light instead of the incident light; it would not make sense to look at incoming light on a surface.

4.2.2 Splitting

Remember from Section 2.2.2 that the performance of the EM-algorithm can be drastically increased by not using all data at once but by splitting the data into minibatches. This is important for surface light field, because the dimensionality and therefore amount of data has increased; each sample has two more values going from 4-D to 6-D.

The previous splitting technique, which entailed splitting the data in equally-sized blocks won't work: some blocks might contain very few samples, while others might contain a lot. Ideally we want to prepare data chunks with an equal amount of samples. This constraint should not be taken too seriously; data samples within a block can be repeated with moderation. Repeating is used to attain an equal number of samples in each block if that is not already the case. This indicates that what is written to disk and what is contained in a spatial block is not necessarily the same.

To efficiently split up the data in similarly-sized blocks, space partitioning concepts from the computer graphics domain are used. The implementation of the preprocessor provides two of them, the *octree* and the *median-cut tree*. Both techniques are implemented recursively. Space partitioning algorithms start by determining the boundaries of the data, followed by performing a split depending on the technique. For each split the partition technique evaluates if there are enough samples



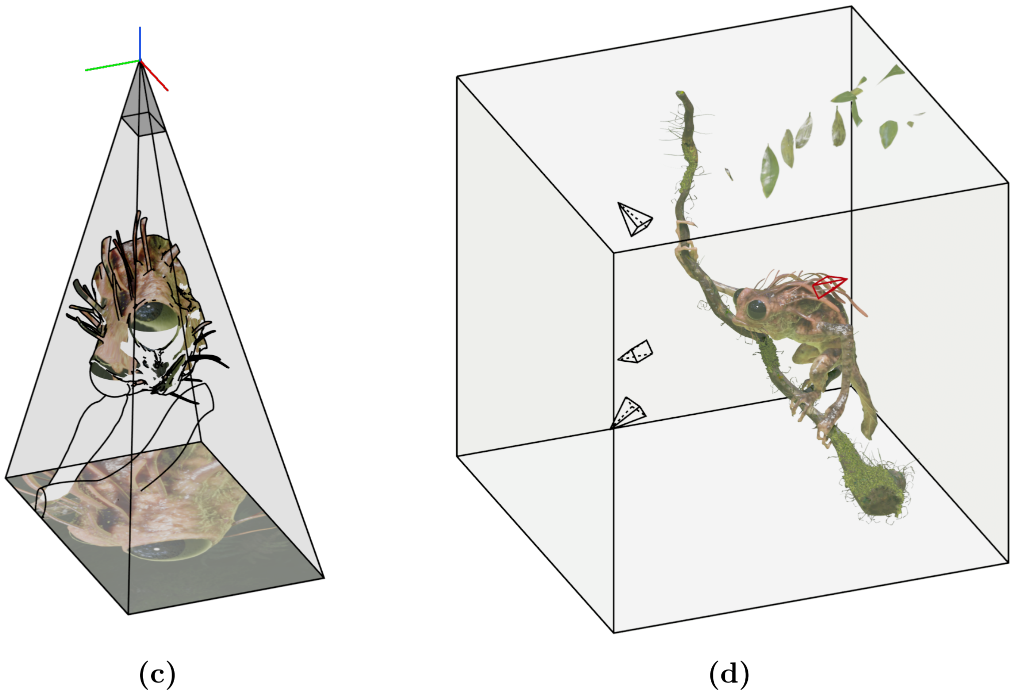


Figure 4.2 – *The inverse process of performing a perspective projection with a camera. The image in screen coordinates (Figure 4.2a) is first transformed to clip space with coordinates between -1 and 1 (Figure 4.2b). The data is then projected back into space from the camera with the help of the provided geometry in the form of a depth map (Figure 4.2c). Finally the resulting samples are transformed to be in the correct place in the complete scene (Figure 4.2d).*

to perform another split. If not, the node is a leaf node. Otherwise, split the data again.

An octree recursively splits the 3-D space in 8 equal octants. The advantage of an octree is its speed, while keeping nodes relatively balanced. It is fast because it only has to calculate the midpoint of a minimum and a maximum in each dimension. In contrast, the implementation of the median-cut tree splits each dimension by an equal number of data points on each side. For each split the median needs to be determined in all directions, which is quite costly. Compared to an octree, the samples are almost perfectly balanced between nodes.

The type of partitioning is specified in the command-line arguments. Both octree and median-cut are implemented. After the process of splitting is finished and all nodes are leaf nodes, the content of each block is completely shuffled. This is because the data is still in a certain order caused by their originating image but each microbatch should represent the full block. The blocks themselves are still in some kind of order because the data is only shuffled inside the blocks. Finally, the preprocessor writes the content of the nodes to separate files on disk.

The preprocessor can switch between splitting in the complete 6-D space, or only in the 3-D space of sample locations. Samples close to each other in 3-D space often have similar light directions. Splitting based on light direction could create unnecessary splits between co-located samples.

4.2.3 In-Memory Processing

A large flaw of the current preprocessor is that it operates on in-memory data. It uses uncompressed arrays of pixel data augmented with location data and direction data. Data sets where the views have larger resolutions or data sets which contain a lot of views simply cannot be processed due to this limitation. The preprocessor fails when sorting the array to be cut in half by the space partitioning algorithm.

A data set with 169 512×512 images is doable. The maximum amount of data in memory at once for such a data set is equal to:

$$512 \cdot 512 \cdot 169 \cdot (12 + 12 + 3) = 1387266048 = 1323\text{MB}.$$

In this equation, $12 + 12 + 3$ stands for 3 times a float of 4 bytes for the location component, 3 times a float of 4 bytes for the direction component and 3 bytes for the colour component.

A possible solution could be a form of external sorting. Firstly, all raw data points are written to disk in several files and sort each of the manageable-sized files individually. Then, the first part of each file is loaded into memory and sort all those parts together; this sorted data is then written to the final file with sorted data. Next, the following chunks of each file are loaded, sorted, written to the final file, and so on.

4.3 Data Creation

4.3.1 Blender Add-On

The desired data should have the following properties:

1. It must contain light information.
2. The data must include location information
3. The data is captured from one of the possible configurations discussed in Section 3.2.

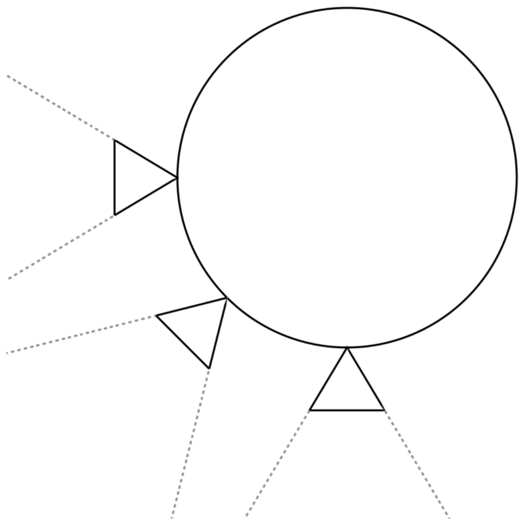
There already exist tools for generating light-field data, such as Honauer's add-on for Blender [12]. Unfortunately, this add-on allows for capturing planar light fields and depth, but does not allow to generate data from other configurations. Additionally, its performance decreases for larger setups and does not work with Blender 2.8. The problem with incompatibility with Blender 2.8 is that it does not have access to the physically based real-time renderer EEVEE, nor does it benefit from great performance improvements of the ray-tracing renderer.

To be able to create more immersive light fields during the research in this master dissertation, a novel Blender add-on was created from scratch which fulfils the above requirements. The add-on is capable of generating images from a planar setup and the configurations discussed in Section 3.2 (cuboid, cylinder and sphere). This way, future researchers can generate data sets for each of those configurations, allowing to experiment with them. The add-on outputs depth maps for each viewpoint, which serve as the location information for the data points. Finally, the add-on also outputs a configuration file, indicating the positions and orientations of each viewpoint; this serves as input for the preprocessor discussed earlier. Section 5.1 will compare our method with the existing add-on in terms of rendering and creation time of configurations.

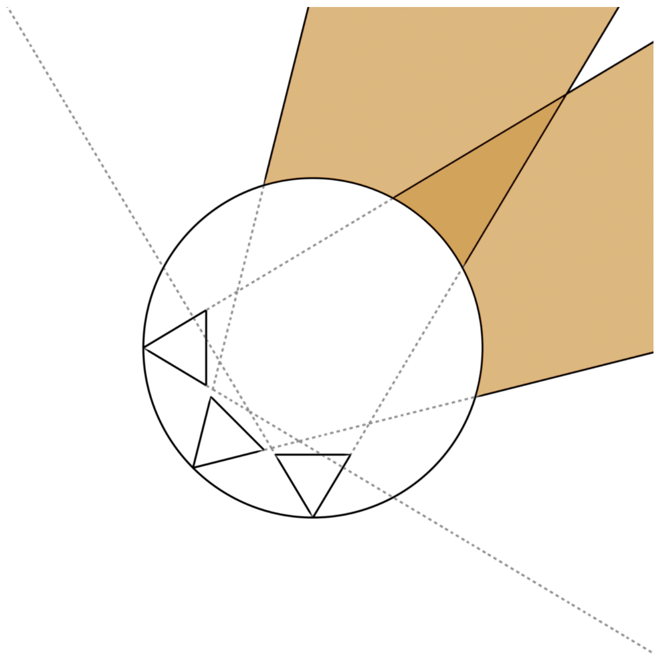
4.3.2 Inverting Cameras

The add-on allows cameras to face the opposite direction in the configurations. This option is mostly useful for the cylindrical and spherical light field types. Facing the cameras inwards as opposed to outwards causes neighbouring cameras to have more overlap in their viewing range. Objects are therefore more likely to be captured by more cameras than before. The scenario with the cameras facing outside is pictured in Figure 4.3a. The different viewpoints will only slightly overlap at a greater distance and with at most two cameras. In Figure 4.3b, a small area is covered by all three cameras, while two cameras have quite a large area in common.

The benefit is that when more viewpoints capture an object, more specular information can be stored. On the other hand, the viewpoints are further away from the objects. Slightly less detail might be captured, especially for objects close to the edge of the circle.



(a) *When the cameras face outwards, their viewing regions do not overlap very much.*



(b) *When facing the cameras inwards, surfaces are covered by more camera viewing regions.*

Figure 4.3 – *Changing the direction of the camera's on cylindrical or spherical setups capture more specular information (more directions), but lose some detail. Its use in real world scenarios is fairly limited.*

The biggest problem with this configuration is its physical value. In a software scenario there is no problem; cameras are invisible and do not occupy space. In reality cameras on one side of the circle block the view of cameras on the other side.

4.3.3 Performance

Honauer's add-on has some performance problems when working with larger amounts of cameras. Viewport navigation becomes quite slow. Also, each time the number of cameras is updated, new camera data is added while old data is not removed. The same applies for the material-data of the setup, which is purely visual. When working for an extended period, data can pile up. Instead of cleaning unused data, the add-on only uses a single camera. The camera is then translated and rotated according to the next viewpoint between each render. In addition, the novel add-on works for Blender 2.8, gaining a significant overall performance boost compared to Honauer's add-on. Section 5.1 will go more in depth on evaluating the performance of the add-on.

4.3.4 Blender Integration

The add-on is designed to completely adhere to Blender add-on standards. The correct menus are used for hosting the different settings. This makes maintenance of the code manageable and future-proof.

The light-field types are placed in their own data structure. This permits a user to create multiple light fields of different types with distinct settings in the same Blender file. Comparing types or light-field configurations can conveniently occur in the same file.

The add-ons light-field setups can be treated as a generic object in Blender. It supports the most common operations such as translating, rotating, uniform scaling, deleting, renaming, etc. A small flaw is the lack of non-uniform scaling for spherical and cylindrical types. Doing so will also scale the cameras, and when the scaling axis is not axis-aligned, a shearing effect is applied on the resulting image. Blender documentation states that cameras can only scale uniformly, but a bug allows this when working with the Python API.

4.4 Conclusion

In this chapter the implementation of the SMoE preprocessor is discussed, along with a method for creating data from several configurations. The preprocessor serves as a basis for future continuation of research in surface light fields. Unfortunately, the preprocessor currently holds the complete light-field data set in memory, which makes it unusable for larger light-field data sets. As part of future work, the preprocessor must be adapted to allow preprocessing of data sets of any size. In the following chapter the perprocessor will be qualitatively evaluated with the help of a light-field visualisation tool.

The Blender add-on for generating light-field data sets from planar, cuboid, spherical and cylindrical configurations also provides a helpful tool in light field research. A specific method for generating surface-light-field data sets is not implemented. This is due to the complexity of generating random viewpoints that capture the necessary light data of an arbitrary scene. A possible manual solution is to use a camera which is animated and moves on a path specified by the user. The animation can then be rendered in individual frames of which the metadata is stored each frame. This method is implemented in a simple script which is discussed more in Section 5.2.3. Other methods are also part of future work. The novel Blender add-on will also be compared to the previous add-on, in terms of performance.

Chapter 5

Evaluation

This chapter evaluates the novel methods for creating light-field data and preprocessing it for modelling with the SMoE framework. The Blender add-on is compared to the previous add-on in terms of performance. Even though it has multiple novel features, those features can only be qualitatively evaluated. Those qualitative tests are considered out of scope for this dissertation. To evaluate the preprocessor, a novel 6-D light-field point-cloud renderer is developed. The renderer allows for qualitatively evaluating point-cloud light fields such as the surface light fields in this dissertation.

5.1 Blender Add-On Performance Comparison

5.1.1 Creation Performance

The existing Blender add-on contains several flaws which impact performance. When a setup is created, several objects are instantiated to represent the configuration and capture light fields. For each viewpoint, a new camera is created. Blender makes a separation between the visual camera object and the data it contains. The camera data holds all camera information (e.g. resolution, focal length, F-stops,...). The problem is that a lot of such objects in a single scene slow down the performance of the viewport, which becomes less responsive or even impossible to work with. There are other objects which are instantiated, but those do not scale with the configuration, and are therefore not too relevant to the problem.

Creating camera data in itself does not take very long; adding a single camera to the scene happens visually instantly. The problem is that creating numerous cameras is slow, and planar light-field configurations often have a lot of viewpoints. To show the impact on performance, the time it takes to create a light field configuration was measured for different amounts of viewpoints. A function was implemented in the existing add-on which executes the light-field-creation routine 10 times and averages the timing results. The result can be seen in Figure 5.1; the tests are performed on a laptop with an Intel i7-7500U CPU.

Weirdly enough, the creation time is not linear but quadratic in function of the

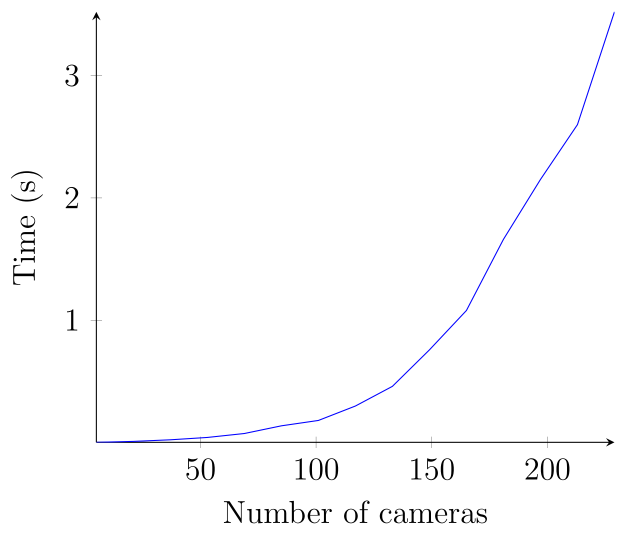


Figure 5.1 – *Configuration-creation time in function of number of cameras for the existing add-on. Creation times skyrocket with increasing numbers of cameras.*

number of cameras, contrary to what is expected. When we run the test in the same file for an equal number of cameras each time, the computation time also grows, which could explain the non-linear growth. The fact that the creation time rises for an equal number of cameras might have something to do with yet another problem: only the camera visuals disappear when cameras are destroyed, not the corresponding camera data.

One might think this is not a big problem because a user can simply work with a light-field configuration once it is created. That is unfortunately not true. Changing camera parameters or the baseline between cameras also uses the creation routine. This means that each time a user wants to change a single parameter of a configuration with 200 cameras, they have to wait more than 2 seconds. That time only increases for larger configurations, and the longer the user is working with the file.

The novel Blender add-on solves this problem entirely. Only a single camera is used instead of one for each viewpoint in the configuration. When rendering the light field, each viewpoint is rendered by moving the camera to the corresponding viewpoint. This solves the problem of the increasing amount of camera data. Other occurrences of orphan data in the light-field setup are also removed, solving that problem completely.

Figure 5.2 shows a similar plot as before, for the novel Blender add-on. Note that the time axis is in milliseconds, not seconds like before. A configuration with 16000 viewpoints can be created in 12 milliseconds, which is an improvement compared to the previous add-on. The plot only shows a fraction of the number of cameras that are possible now. A configuration with a million cameras only takes 0.789 seconds; an amount which will probably never be rendered in one light-field configuration. In conclusion, our method outperforms the existing add-on in terms of creation speed by a lot.

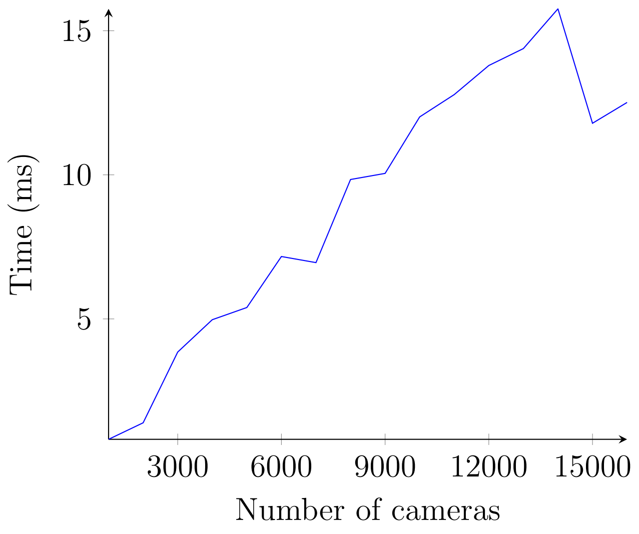


Figure 5.2 – *Configuration creation time in function of number of cameras for our add-on. The inaccuracy is due to the timing method not being perfectly accurate, even though each configuration was tested a 100 times. Also note that the time axis is in milliseconds, not in seconds.*

5.1.2 Rendering Performance

The novel add-on is compatible with Blender versions 2.8 and above. This means it has access to the EEVEE rendering engine and also benefits from increased performance of the ray-tracing engine. To compare performance of the ray-tracing engine, several scenes are rendered in version 2.81 and in version 2.79b which is the latest version in which the previous Blender add-on is compatible. The test scenes are Blender demo files, available on the official Blender website; the corresponding render results of each scene are shown in Figure 5.3.

Each scene is rendered four times in both versions on an Intel i7-7500U CPU and the rendering times are averaged. The results can be seen in Figure 5.4. It appears that version 2.81 is on average 30% faster compared to what the previous blender add-on could do in version 2.79b. This allows for faster creation of light-field data sets with the ray-tracing engine.

The EEVEE render engine is a physically-based real-time rendering engine. This engine can prove to be very useful in creating light-field data sets really fast. Unfortunately, EEVEE is not perfect when it comes to rendering certain types of shading effects, such as global illumination, ambient occlusion, reflection, transparency and more. The EEVEE render is a good choice for light-field data-set creation if the limitations of the engine are not of issue.

(a) *Classroom (1920×1080)*

(b) *Barbershop (2048×856)*

(c) *BMW (1920×1080)*

(d) *Pavillion (1920×1080)*

Figure 5.3 – *The different scenes used for comparing render times.*

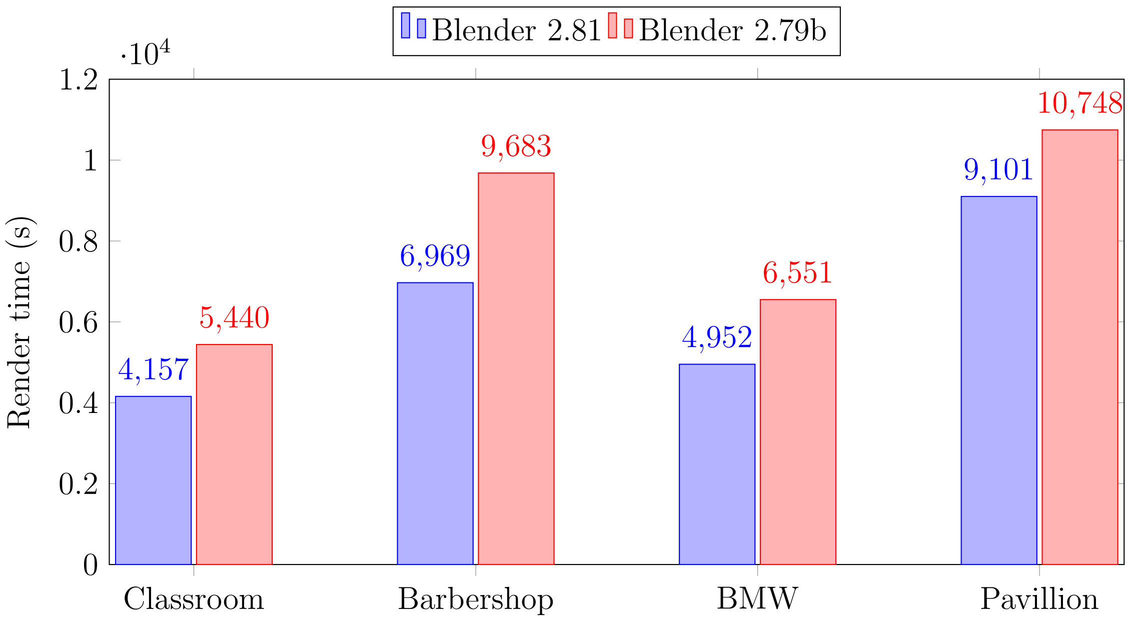


Figure 5.4 – *Comparison between Blender 2.81 and 2.79b in terms of rendering speed of the ray-tracer. It is clear that Blender 2.81 is faster.*

5.2 Qualitative Evaluation of Preprocessed Light-Field Data

It is not yet possible to perform any quantitative quality evaluation of non-planar light fields compressed with SMoE, simply because those types of light fields have not been created yet with SMoE. Instead, the preprocessor will be evaluated qualitatively with the help of the novel 6-D light-field point-cloud renderer. This visualisation tool allows for inspecting the preprocessed data. The tool uses the OpenGL API for rendering graphics to the screen. It draws all data points of which the direction approximately points to the virtual camera. The angle under which a point is visible can be changed in the view settings.

5.2.1 Verification

To show that the preprocessor actually works, a light field of a tree creature is captured with EEVEE from a sphere with 169 viewpoints. The sphere encapsulates the tree creature, with the cameras facing inwards. Figure 5.5 shows two viewpoints from the visualisation tool. Figure 5.5a shows strong specular highlights on the tree creature's nose while in Figure 5.5b the nose-highlights are in a different place. The highlight changes because the users viewpoint changes; different light data is shown for different viewpoints.

The tool also allows us to visualise the spatial partitioning result of the data. With the spatial partitioning visible, it becomes apparent that the octree structure has more empty spaces, but also groups closer data. The median cut partitioning tends to create stretched blocks which sometimes contain uncorrelated data points. Figure 5.6 shows an example of a median cut partitioning of one of the Blender demo-files. Blocks which do not contain any data are not shown to increase visibility.

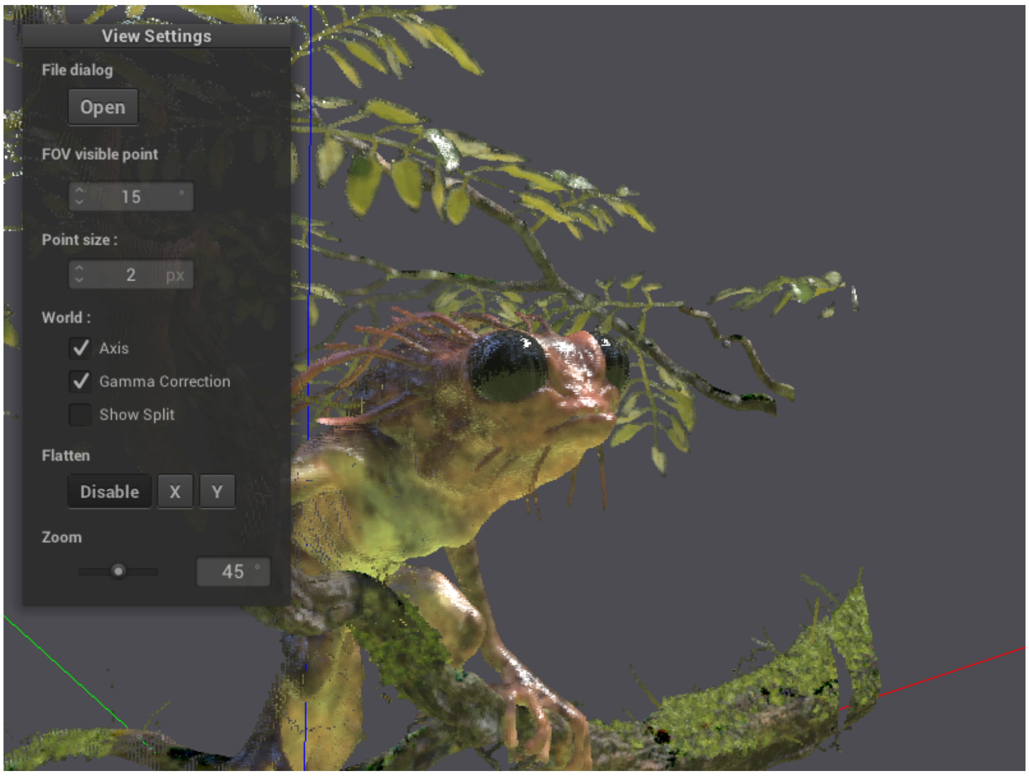
One problem with the visualisation is that it can become quite slow for larger light fields. The data sets quickly contain millions of 9-D elements representing the light. Rendering those all at the same time is computationally intensive. A possible optimisation could be to use the spatial partitioning of the data to prevent data points outside of the viewing frustum from rendering.

5.2.2 Flattening

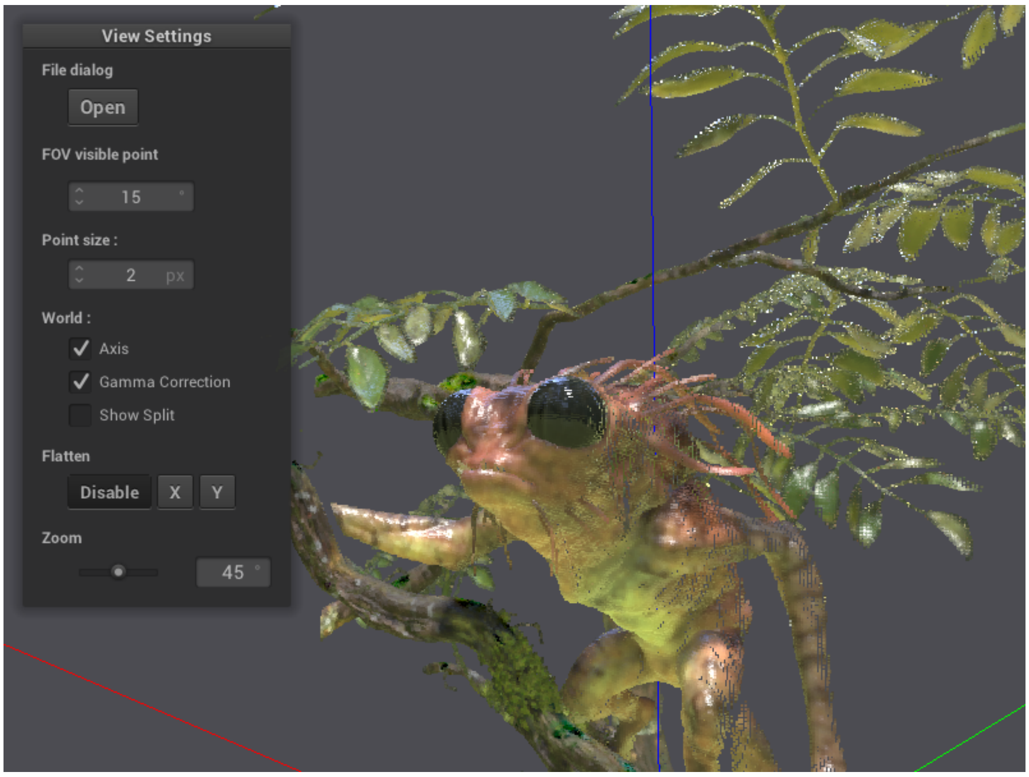
The visualisation tool allows for relocating all data points to the xz-plane or yz-plane. A data point is moved along its light direction until it is on the selected plane. The result of this can be seen in Figure 5.7. The surface light field in this image is captured with a planar grid of 10 by 10 cameras. The result is visualised and flattened on the y-axis. Figure 5.7a shows the result with a small light opening angle of 4 degrees. Figure 5.7b shows a side view with a large opening angle of 180 degrees to show that the data indeed lies on a plane.

Observe how the result is not a perfect view of the light field anymore. For example, take a look at the leaves behind the tree creature in Figure 5.7a. The foliage appears multiple times. What is actually going on is the effect of adding a focus plane, much like in 2-D photographs; The leaves are located behind the focus plane, in this case the plane formed by the x and z-axis. As Figure 5.8 illustrates, surface points in front and behind the focus plane are out of focus.

Remember that the angle for which light is shown is about 4 degrees in the example. For very small opening angles for the light, the out-of-focus effect does not appear;



(a)



(b)

Figure 5.5 – *Two images of the preprocessed surface-light-field data of the tree creature. Notice the highlights changing depending on the view, which indicate the preprocessing succeeded.*

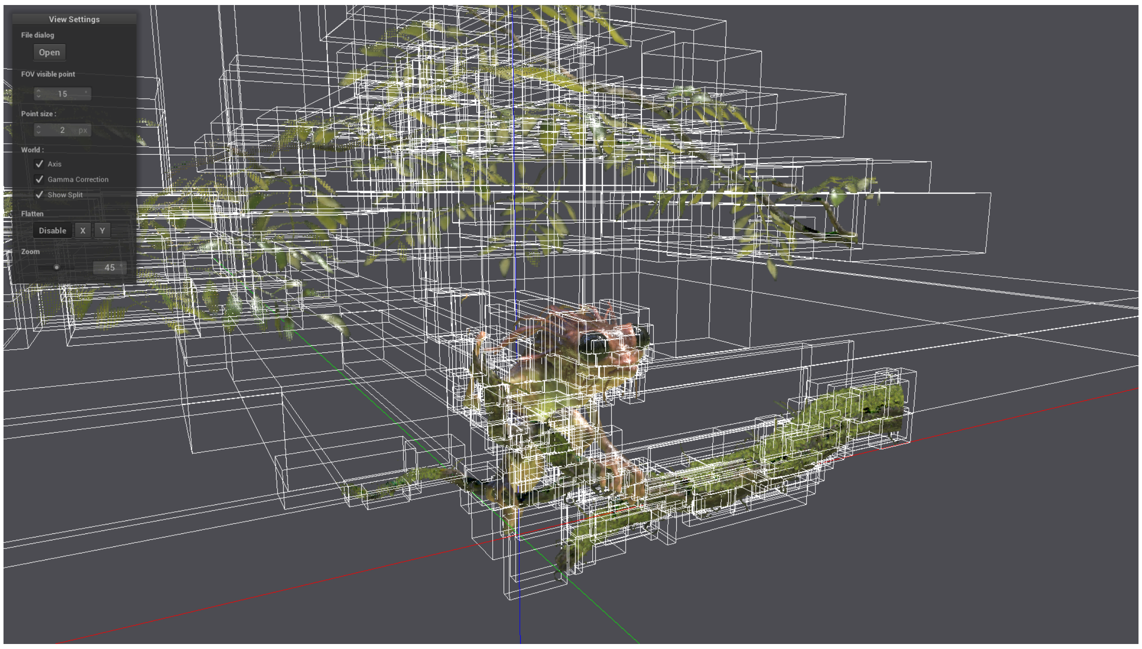

View Settings
File dialog
Open
FOV visible point
15
Point size :
2 px
World :
✓ Axis
✓ Gamma Correction
✓ Show Split
Flatten
Disable X Y
Zoom
45

Figure 5.6 – *The tree-creature data set with the median-cut partitioning visible. The white boxes indicate volumes containing samples, all of which have the same amount.*

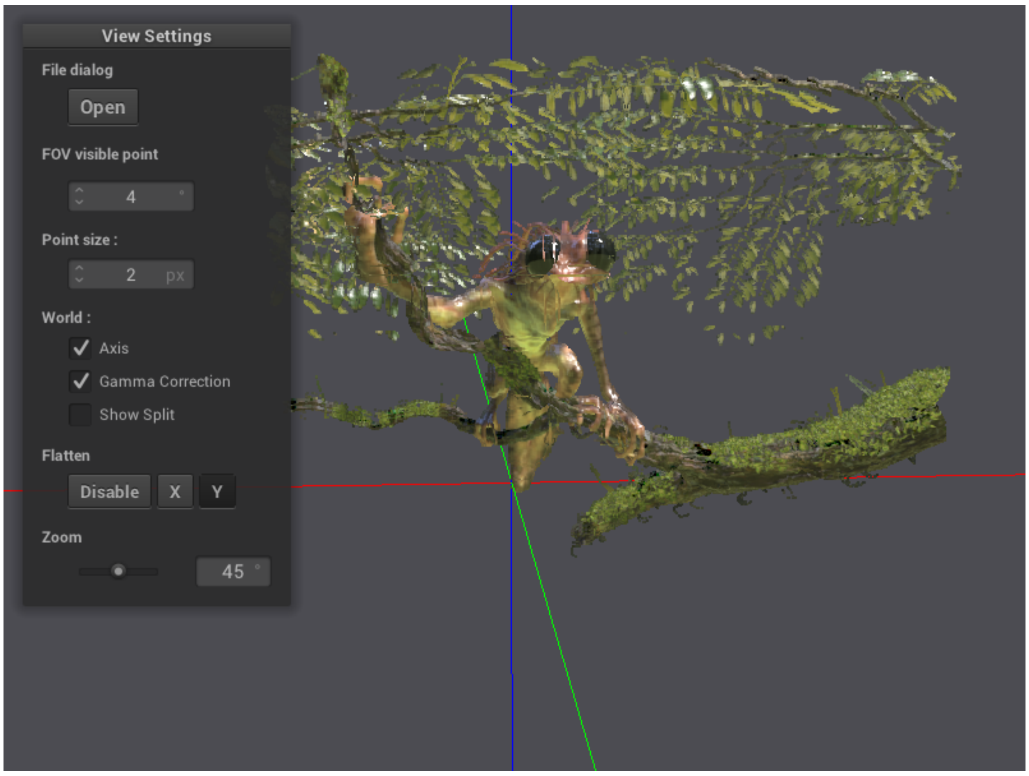
the light must be directed perfectly at the viewport camera.

5.2.3 Real-World Captured Surface Light Field

As an experiment, the preprocessor is also tested with real-world light data. The data comes from filming a scene we want to convert into a 6-D light-field point cloud. The biggest problem is getting the positions of the camera in space, relative to the captured objects. This was easy when capturing in software with the help of the novel add-on; we do not have this luxury when capturing with hardware. To solve this, tracking software was used to approximate that information. The tracking software in question is a part of Blender. Depth information can be created afterwards by blocking out the scene with simple geometry, which is also done in Blender.

First, markers are placed over the scene in preparation of tracking; the software needs anchor points in the scene to solve the camera motion. It is possible to do this without markers, but the process is drastically sped up when they are available. One of the viewpoints of the scene is shown in Figure 5.9. Notice the black dots all over the setup.

Next, the footage is captured and loaded into Blender. It is important to capture light from multiple directions because this improves the visibility of the light field. In Blender, virtual trackers are placed on key locations in the footage such as the black dots, but other features work as well (e.g. red cap of the bottle, button on one of the controllers). Blender allows for automatic detection of good features. In combination with some manual tweaking, the camera motion can be resolved. This



FOV visible point 4 °

Point size : 2 px

World :

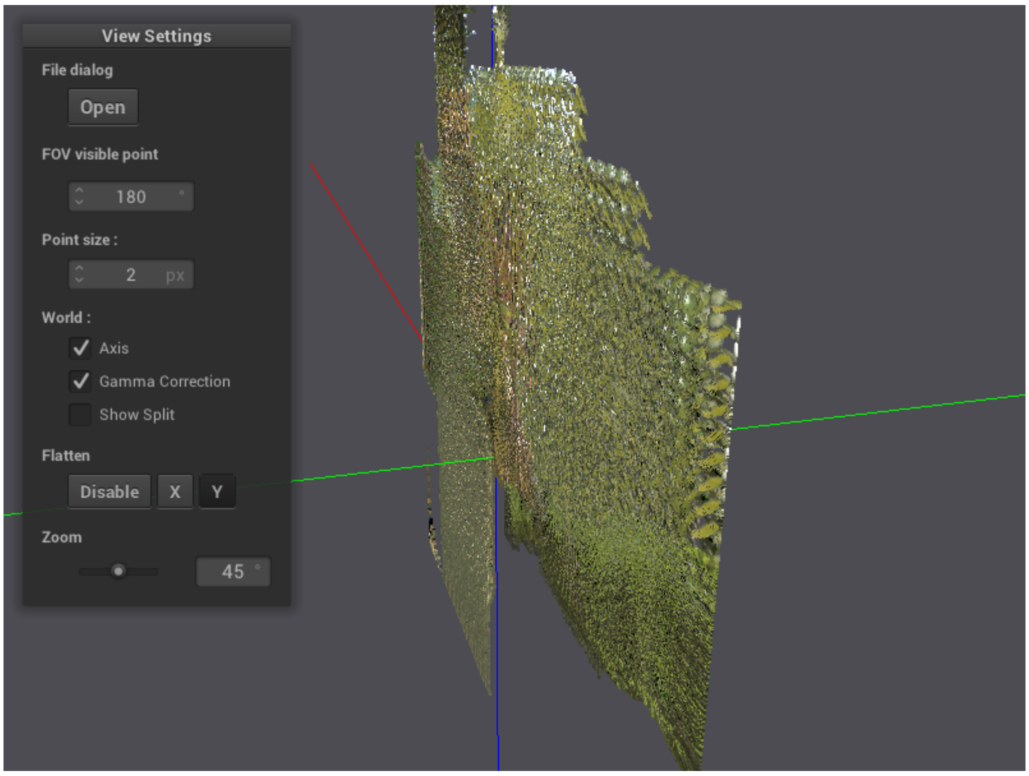
✓ Axis

✓ Gamma Correction

Show Split

Zoom 45 °

(a) *The data of the tree-creature light field flattened on the xz-plane. Notice the multiple appearances of leaves in the background.*



FOV visible point 180 °

Point size : 2 px

World :

✓ Axis

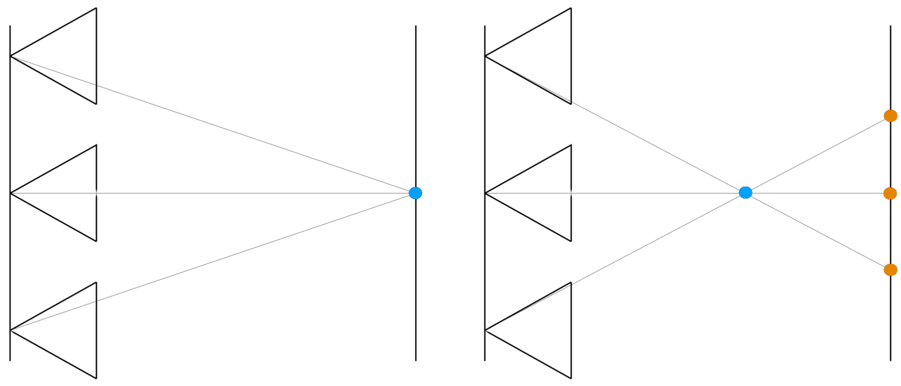
✓ Gamma Correction

Show Split

Zoom 45 °

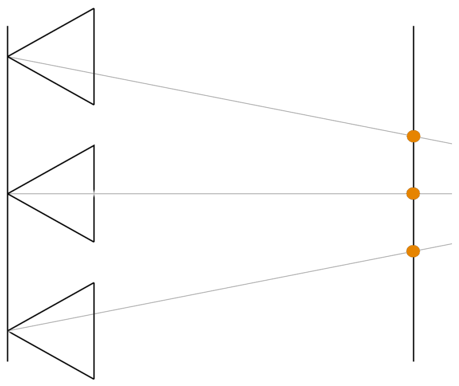
(b) *A side view of the flattened data with all light angles at 180 degrees. This way all samples are visible for the current view.*

Figure 5.7 – *A flattened version of the tree-creature data set.*



(a) *In case a point on the surface coincides with the focus plane, there is no discrepancy between original location and flattened.*

(b) *When a point on the surface lies in front of the focus plane, it is projected in three different places on the focus plane.*



(c) *When a point lies behind the focus plane, the samples from different cameras are spread away from each other.*

Figure 5.8 – *Cameras on a plane with data points flattened. The blue dots indicate the original position, while the orange dot indicates the projected position.*

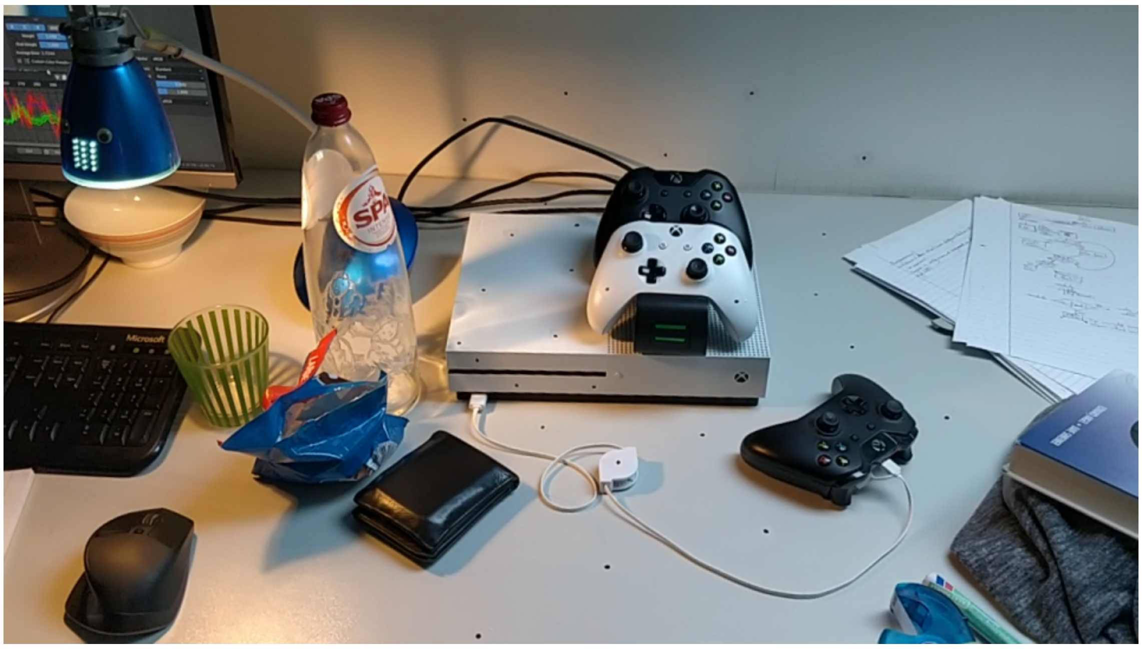


Figure 5.9 – *A scene in the 6-D point-cloud viewer captured with a physical camera. The black dots are used for tracking.*

means that the software places virtual trackers in 3-D space in the locations they appear in the filmed data and animates the camera relative to those trackers.

Finally, every tenth frame is extracted together with a depth map and the position and orientation of the camera. This depth map is rendered from a blocked-out approximation of the scene (see Figure 5.10). The idea is that the geometry does not need to be perfect.

The preprocessor is then used to transform the data into a 6-D point cloud, located on the geometry. The resulting point cloud is illustrated in Figure 5.11 with the help of some screenshots. We see that objects which lie on their corresponding geometry are very clear, even though the geometry was very rough. For example, the wallet is more complex than a simple cuboid, but no artefacts are visible. Some objects that don't have geometry are also visually accurate, such as the paper or the cable in the middle of the scene. Those objects are very close to another surface, which confirms that approximate geometry is already very helpful in showing light data. On the other hand, some objects such as the glass left of the bottle are hazy; this is because their light data is spread all over another surface.

We conclude that approximating the geometry of a scene and using it for surface light fields is a good method for improving the immersiveness of light field technology. Although some manual labour is required to create a surface light field of a real-world scene, nice results are achieved and it is definitely a step in the right direction. Imagine an on-stage performance filmed with smartphones by thousands of spectators, where the footage is streamed to a server that processes that data; a distant participant can follow along virtually, from anywhere in the crowd!

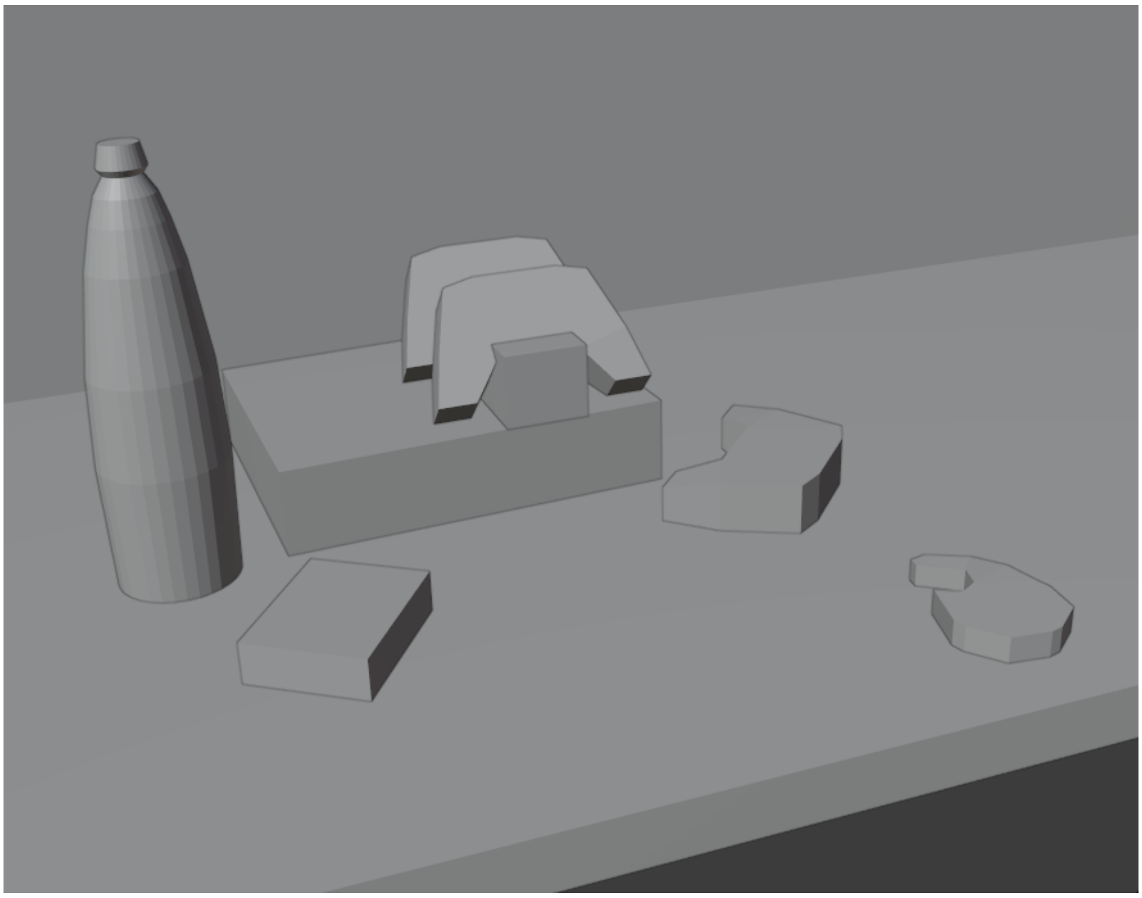


Figure 5.10 – *The geometry used for generating the accompanying depth information of the surface light field. Only the main objects are blocked out.*

5.3 Conclusion

The novel Blender add-on is certainly an improvement compared to older synthetic methods for creating light fields. The performance and user-friendliness make it a good tool for light-field data-set creation and light-field research. A great addition to the add-on could be to implement a method for capturing a light field from randomly spread cameras in the scene, with the purpose of capturing more generic surface light fields.

The preprocessor works correctly and has already been useful in light-field research. We succeeded at processing a real-world captured scene with the preprocessor, which indicates it is not bound to light-field data created with software. The resulting surface light field is structurally correct, indicating that surface light fields are certainly a valid option for improving immersiveness of light fields. A necessary change to the preprocessor is to load and process manageable chunks of data instead of the complete light field at once, as discussed in Section 4.2.3.

A possible improvement for the visualisation tool is to use hidden point removal while rendering. The data already disposes of a space partition which can be used to exclude complete blocks from rendering. This would speed up the viewport, which

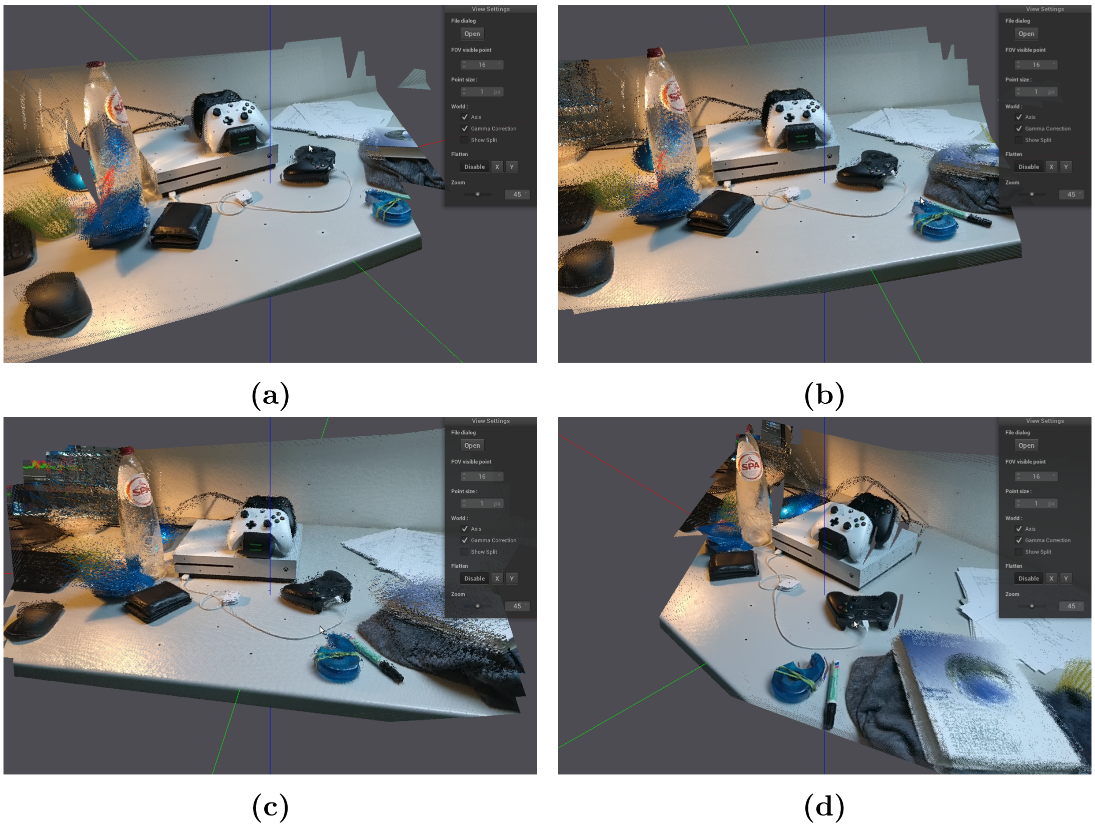


Figure 5.11 – *Four screenshots from the resulting 6-D point cloud. Notice how the reflection on the white tabletop changes depending in the angle. Other interesting areas are objects with no geometry (e.g. bag of candy in front of the bottle, or glass to the left of the bottle) and the transparency of the bottle.*

will certainly be necessary for larger light fields.

Chapter 6

Conclusion and Future Work

The first goal of this dissertation was to explore the available options on improving immersiveness for light fields. Surface light fields presented themselves as the most prominent solution, as they allow for true immersiveness. By no means were exclude cuboid, cylindrical or spherical light fields excluded as a solution to the problem; it is up to future research to work out the details of those methods. A start to that research could be the linear approximations for cylindrical and spherical light fields. With linear approximations, the light fields could be treated as planar light fields.

We believe that the SMoE framework is a good solution for modelling light fields. Consequently, we investigated how the current SMoE framework must be adapted in order to model surface light fields. The core algorithms do not change because they can work with any dimensionality; it is the structure of the data and the algorithm for rendering that must be adapted. This dissertation contributed to the first, the structure of the data; a novel data-preprocessor allows for modifying a light-field data set consisting of images and depth images into a 6-D light-field point-cloud data set. That point cloud is intended to be modelled by SMoE.

Evaluation of this preprocessor requires data to operate on and a method of displaying the processed data. This dissertation contributes a novel tool for generating the corresponding data. The tool is able to generate planar, cuboid, cylindrical and spherical light-field data sets. It is an improvement compared to existing methods; this was quantitatively evaluated. Another contribution is a 6-D light-field point-cloud visualisation tool, capable of reliably rendering smaller light field generated by the data-generation tool. With the help of this tool it became apparent that a planar light field and the flattened version of a surface light fields captured from the same planar configuration essentially contain the same information.

There is still a lot of work left on the road of modelling surface light fields with the SMoE framework. A non-exhaustive list of possible improvements and tasks related to future work could be as follows:

- SMoE has not yet been used to model a surface light field. Doing so would help understand the properties of SMoE for surface light fields and enable quantification of compression ratios.

- To render a surface light field modelled with the SMoE framework, improvements need to be developed instead of using the naive approach of ray-casting from a viewpoint and sampling at the intersection with the surface. Maybe some optimisations of the current SMoE implementation can be used for that purpose.
- Other types of light-field configurations could also provide more immersiveness. Future research could explore how to use the linear approximation for compression.
- Gaussian kernels are not the only possibility for kernels. Other types might work better for specific types of light fields, such as the curved surface of a sphere or cylinder, or for surface light fields.
- To make the current preprocessor usable for larger light-field data sets, the problem of processing everything in memory should be resolved.
- Because the goal is to model real-world content eventually, methods of capturing such data with the necessary information is a part of future work. This master dissertation already experimented with a simple example in Section 5.2.3, but there is a lot more work to do yet before real-world surface light fields becomes tractable.
- The add-on could be improved by implementing a method for capturing light fields from viewpoints in arbitrary locations. Currently, the viewpoints are located on a fixed structure such as a plane, a cuboid, a cylinder or a sphere.
- Finally, future research should assess whether modelling and rendering of surface light-field video is possible. An extra dimension makes everything more complex but would be a great addition to the world of immersive content.

References

[1] M. Levoy, “Light fields and computational imaging”, *Computer*, vol. 39, no. 8, pp. 46–55, Aug. 2006.

[2] R. Verhack, “Steered mixture-of-experts for image and light field representation, processing, and coding: A universal approach for immersive experiences of camera-captured scenes”, PhD thesis, Technical University of Berlin, 2020.

[3] R. Verhack, T. Sikora, G. Van Wallendael, and P. Lambert, “Steered mixture-of-experts for light field images and video: Representation and coding”, *IEEE Transactions on Multimedia*, vol. 22, no. 3, pp. 579–593, Mar. 2020.

[4] B. S. Wilburn, M. Smulski, H.-H. K. Lee, and M. A. Horowitz, “Light field video camera”, in *Media Processors 2002*, vol. 4674, San Jose, California, pp. 29–36.

[5] D. Koller, M. Turitzin, M. Levoy, M. Tarini, G. Croccia, P. Cignoni, and R. Scopigno, “Protected interactive 3D graphics via remote rendering”, *ACM Transactions on Graphics (TOG)*, vol. 23, no. 3, pp. 695–703, Aug. 2004.

[6] T. Georgiev, Z. Yu, A. Lumsdaine, and S. Goma, “Lytro camera technology: Theory, algorithms, performance analysis”, *Multimedia Content and Mobile Devices*, vol. 8667, 2013. DOI: `10.1117/12.2013581`.

[7] Raytrix. (). Raytrix, [Online]. Available: `https://raytrix.de/` (visited on 07/23/2020).

[8] R. S. Overbeck, D. Erickson, D. Evangelakos, M. Pharr, and P. Debevec, “A system for acquiring, processing, and rendering panoramic light field stills for virtual reality”, *ACM Transactions on Graphics (TOG)*, vol. 37, no. 6, pp. 1–15, Jan. 2019.

[9] A. Akin, Ö. Cogal, K. Seyid, H. Afshari, A. Schmid, and Y. Leblebici, “Hemispherical multiple camera system for high resolution omni-directional light field imaging”, *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 3, pp. 137–144, Jun. 2013.

[10] B. Krolla, M. Diebold, B. Goldlücke, D. Stricker, and H. Collaboratory, “Spherical light fields”, in *British Machine Vision Association (BMVC)*, Nottingham, England, 2014.

[11] M. Levoy and P. Hanrahan, “Light field rendering”, in *Proc. of the 23rd Annu. Conf. on Computer graphics and interactive techniques*, New York, 1996, pp. 31–42.

[12] K. Honauer, O. Johannsen, D. Kondermann, and B. Goldluecke, “A dataset and evaluation methodology for depth estimation on 4D light fields”, in *Asian Conf. on Computer Vision*, Chem, Germany, 2016, pp. 19–34. DOI: `10.1007/978-3-319-54187-7_2`.

[13] B. Koniaris, M. Kosek, D. Sinclair, and K. Mitchell, “Real-time rendering with compressed animated light fields”, in *Proc. of the 43rd Graphics Interface Conf.*, Edmonton, Alberta, Canada, 2017, pp. 33–40.

[14] J.-X. Chai, X. Tong, S.-C. Chan, and H.-Y. Shum, “Plenoptic sampling”, in *Proc. of the 27th Annu. Conf. on Computer graphics and interactive techniques*, New York, 2000, pp. 307–318.

[15] Z. Lin and H.-Y. Shum, “A geometric analysis of light field rendering”, *Int. Journal of Computer Vision*, vol. 58, no. 2, pp. 121–138, Jun. 2004.

[16] M. Cohen, S. J. Gortler, R. Szeliski, R. Grzeszczuk, and R. Szeliski, “The lumigraph”, in *Proc. of the 23th Annu. Conf. on Computer science*, New York, NY, USA, 1996, pp. 43–54.

[17] T. Ebrahimi, S. Foessel, F. Pereira, and P. Schelkens, “Jpeg pleno: Toward an efficient representation of visual reality”, *IEEE MultiMedia*, vol. 23, no. 4, pp. 14–20, Oct. 2016.

[18] M. B. de Carvalho, M. P. Pereira, G. Alves, E. A. B. da Silva, C. L. Pagliari, F. Pereira, and V. Testoni, “A 4D DCT-based lenslet light field codec”, in *25th IEEE Int. Conf. on Image Processing (ICIP)*, Athens, Greece, 2018, pp. 435–439.

[19] C. Perra and P. Assuncao, “High efficiency coding of light field images based on tiling and pseudo-temporal data arrangement”, in *2016 IEEE Int. Conf. on Multimedia Expo Workshops (ICMEW)*, Hamburg, Germany, pp. 1–4.

[20] A. T. Hinds, D. Doyen, P. Carballeira, and G. Lafruit, “Toward the realization of six degrees-of-freedom with compressed light fields”, in *2017 IEEE Int. Conf. on Multimedia and Expo (ICME)*, Hong Kong, pp. 1171–1176.

[21] S. E. Yuksel, J. N. Wilson, and P. D. Gader, “Twenty years of mixture of experts”, *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 8, pp. 1177–1193, Aug. 2012.

[22] P. Liang and D. Klein, “Online EM for unsupervised models”, in *Proc. of Human Language Technologies: The 2009 Annu. Conf. of the North American Chapter of the Association for Computational Linguistics*, Boulder, Colorado, 2009, pp. 611–619.

[23] X. Zhang, P. A. Chou, M. Sun, M. Tang, S. Wang, S. Ma, and W. Gao, “A framework for surface light field compression”, in *2018 25th IEEE Int. Conf. on Image Processing (ICIP)*, 2018, pp. 2595–2599.

[24] A. Kienle and F. Foschum, “250 years lambert surface: Does it really exist?”, *Opt. Express*, vol. 19, no. 5, pp. 3881–3889, Feb. 2011.

[25] P. Kamencay, M. Breznan, R. Jarina, P. Lukac, and M. Radilova, “Improved depth map estimation from stereo images based on hybrid method”, *Radioengineering*, vol. 21, no. 1, Apr. 2012.

[26] M. Dimitrievski, P. Veelaert, and W. Philips, "Semantically aware multilateral filter for depth upsampling in automotive lidar point clouds", in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 1058–1063.