Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS

Jan Kučera

Computational photography of light-field camera and application to panoramic photography

Department of Software and Computer Science Education

Supervisor of the master thesis: Ing. Filip Šroubek, Ph.D.

Study programme: Computer Science, Software Systems

Specialization: Computer Graphics

Prague 2014

I would like to express my thanks to my supervisor, Ing. Filip Šroubek, for his skilled and patient supervision of the thesis, to the Institute of South and Central Asia at Faculty of Arts for their support and understanding, and to the Prague City Hall for allowing and assisting me to take light field pictures from their premises. Finally, thanks to Pavel Bánský from Microsoft and Gus Issa from GHI Electronics, without whom this work would not happen, as they helped me to get the first Lytro camera into my hands.

St. Nicholas from the New City Hall, Prague
Photo by author, inspired by Jiří Turek

Název práce:	Výpočetní fotografie ve světelném poli a aplikace na panoramatické snímky
Autor:	Jan Kučera
Katedra (ústav):	Kabinet software a výuky informatiky
Vedoucí práce:	Ing. Filip Šroubek, Ph.D.
Abstrakt:	Digitální fotografie se neustále snaží dohnat svůj analogový protějšek a zaznamenávání směru světla se v poslední době stalo předmětem tohoto úsilí. První a doposud stále jediný fotoaparát pro běžné uživatele, který zaznamenává světelné pole – Lytro – se na trhu objevil v roce 2011. Tato práce seznamuje čtenáře s teorií světelného pole a jeho zaznamenáváním se zvláštním důrazem na ilustraci zmiňovaných principů ve 2D, shrnuje současný hardware a probíhající výzkum v této oblasti a předkládá analýzu Lytro fotoaparátu samotného. Nabízí popis uzavřených souborových formátů a používaných protokolů, otvírajíc tak prostor pro využití fotoaparátu v dalším výzkumu. Důležitým přínosem práce je přenosná .NET knihovna pro vývojáře, a součástí je i na ní založený editor souborů a program pro bezdrátovou komunikaci s fotoaparátem. Nakonec je popsaná teorie využita k diskusi jejích důsledků pro registraci světelných polí a lineární panorama.
Klíčová slova:	světelné pole, výpočetní fotografie, registrace obrazu, mikročočky, Lytro

Title:	Computational photography of light-field camera and application to panoramic photography
Author:	Jan Kučera
Department:	Department of Software and Computer Science Education
Supervisor:	Ing. Filip Šroubek, Ph.D.
Abstract:	The digital photography is still trying to catch-up with its analogous counterpart and recording light direction is one of the most recent area of interest. The first and still the only one light-field camera for consumers, the Lytro camera, has reached market in 2011. This work introduces the light-field theory and recording with special emphasis on illustrating the principles in 2D, gives an overview of current hardware and ongoing research in the area and analyses the Lytro camera itself, describing the closed file formats and protocols it uses so that further research can be conducted. An important contribution of the work is a .NET portable library for developers, supplemented by a file editor as well as an application for wireless communication with the camera based on the library. Finally, the theory is used to discuss implications for light-field registration and linear panoramas.
Keywords:	light field, computational photography, image registration, microlens, Lytro

Contents

Contents 6

Introduction 8

Disclaimer 9

1. Light Field Photography 10

1.1. History 11

1.2. Light Field 12

1.2.1. Definitions 13

1.2.2. Image formation 16

1.2.3. Direction sampling 16

1.3. Acquisition Techniques 18

1.3.1. Camera arrays 18

1.3.2. Microlens arrays 20

1.3.3. Emerging methods 23

1.4. Processing and Rendering 26

1.4.1. Rendering 27

1.4.2. Sensor equations 29

1.4.3. Parallax 32

1.4.4. Depth of field 34

1.4.5. Refocusing 35

1.4.6. Depth map 37

2. Commercial Products 38

2.1. Software Imitations 39

2.2. Pelican Imaging 39

2.3. Lytro 40

2.4. RayTrix 41

2.5. Light field in other applications 42

3. Lytro camera 44

3.1. Inside the Camera 45

3.1.1. Hardware 45

3.2. File Formats 47

3.2.1. Raw pictures (raw.lfp) 48

3.2.2. Prerendered pictures (stack.lfp, stacklq.lfp) 49

3.2.3. Depth maps (dm.lfp) 50

3.2.4. Calibration files (data.C.#, *.calib) 50

3.3. On the Air 50
3.3.1. Available services 51
3.3.2. Callback messages 52
3.3.3. Commands reference 60
3.4. Lytro Compatible Library 70
3.4.1. Working with files 70
3.4.2. Working with metadata 72
3.4.3. Working with images 72
3.4.4. Accessors 74
3.4.5. Communicating with camera 74
3.5. Supplementary Software 78

4. Panoramic Applications 80
4.1. Motivation 81
4.2. Derivation 82
4.3. Limitations 84
4.4. Super-resolution 85
4.5. Future work 86

Conclusion 87

Bibliography 88

Appendix A. List of acquired MLA configurations 91

Appendix B. List of files on Lytro camera 93

Appendix C. Accompanying media 95

Appendix D. Imprint 98

Introduction

When I was first reading through Ren Ng's dissertation DIGITAL LIGHT FIELD PHOTOGRAPHY [1], I was astonished at the innovativeness of the idea contrasting the simplicity of physics behind it and the fact that anybody can easily convert their digital camera into a digital light field camera, given a piece of glass with small lenses. As repeated many times through the history, claims that no one would ever need sensors with more megapixels were proven wrong again.

It turned out though, that the key piece of glass is extremely expensive, difficult to obtain and rarely reaching the dimensions required for this new application. Hence, I had to have one of the Ren Ng's cameras, the *Lytro camera*.

The Lytro camera is an affordable cutting-edge piece of hardware, but everything has its price — the platform is closed and the company is oriented towards consumer users. Lytro expressed their plans to "eventually provide open APIs for various parts of their picture experience"[1] and even software with editing capabilities[2] couple of months after the product release, but we still wait for this to happen.

Also the ideas around light field turned out to be over hundred years old, spanning researchers from U. S. through Europe to Russia. The main contribution of the dissertation is processing the light fields in Fourier domain, generalization of the light field camera and solving technical but very important difficulties to move from a research idea to the commercial product.

This thesis has two parts — theoretical and practical. In theory, it explains why things work the way they work, with strong emphasis on illustrative examples in 2D that should help understand the principles of light field to readers not skilled in the art. For the practical part, the aim was to try to understand the inner workings of Lytro camera and software so that the camera can be used for further research activities. The ultimate goal was to enable panoramic photography containing depth information of the scene.

It would not be fair at this point to conceal the work of Bricklbauer et al. [2] who have very recently published a solution to the light field panorama problem for 360° scenes. Given that and the considerable success of revealing the details of Lytro camera, the work focuses more on the first two contributions. Nevertheless, the last chapter uses the theory to get an insight of how light fields would work with a different type of panoramas (restricted to translational movement of the camera) and suggests future applications.

[1] http://support.lytro.com/entries/20552307 (originally posted on October 19, 2011)
[2] http://support.lytro.com/entries/20611761 (originally posted on October 27, 2011)

Disclaimer

Neither myself personally nor the Charles University are affiliated with or endorsed by Lytro, Inc. company. The information herein presented is a result of reverse engineering the camera, its software, firmware and my understanding of related patents and is neither official nor confirmed.

Using 3rd party software to communicate with the camera explicitly breaks its warranty[3] and can damage it irreversibly. Some of the commands presented in this work are not used in any official software and might be untested. Use at your own risk.

This camera has stopped working.

Please contact Lytro customer support.

[3] https://www.lytro.com/legal/warranty/

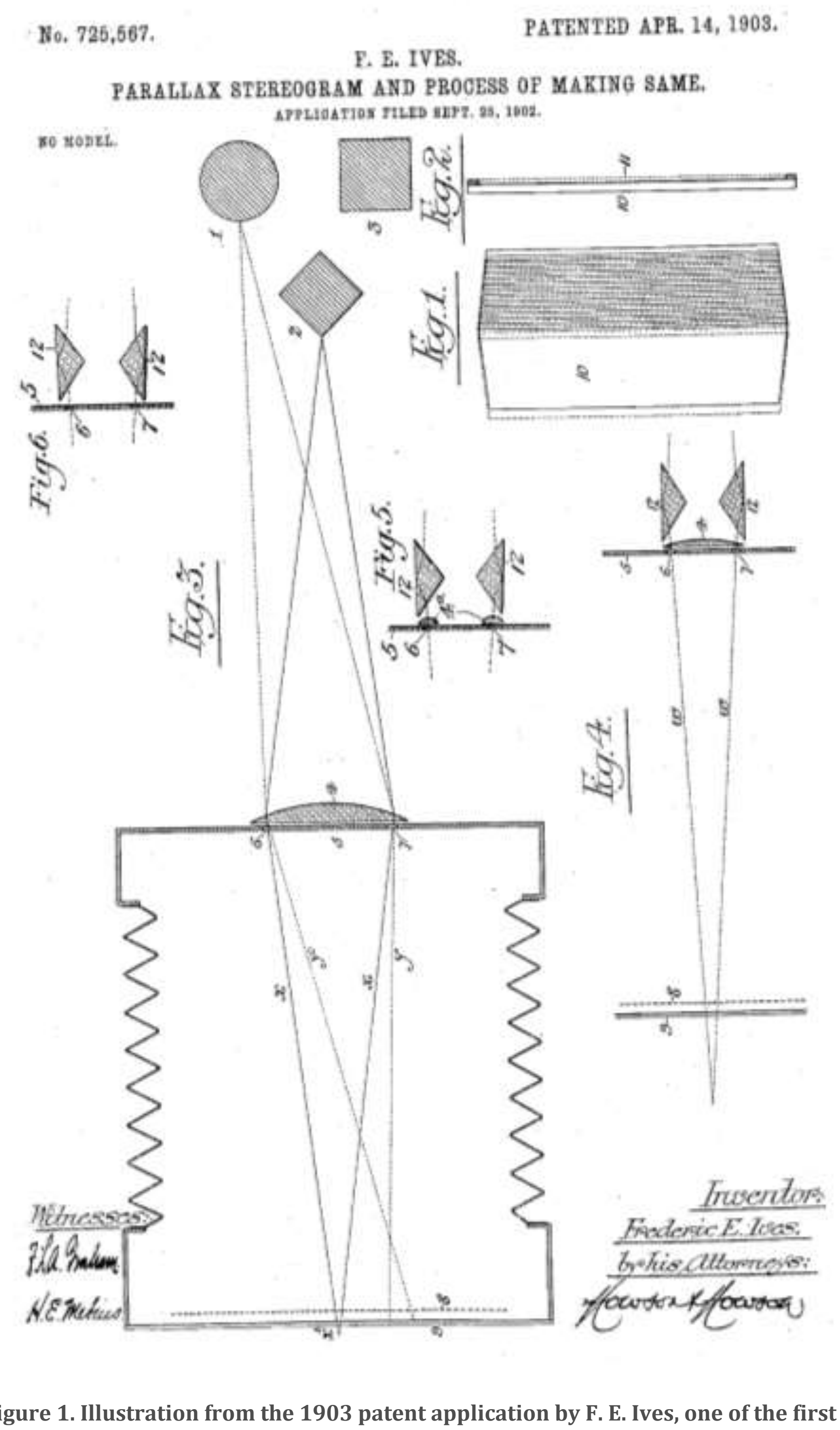


Figure 1. Illustration from the 1903 patent application by F. E. Ives, one of the first ones exploiting the light field principles.

1.1. History

The concept of light field photography, a recording of scene that contains not only intensity but also an information about the direction of individual rays, is over hundred years old. F. E. Ives, an U. S. invertor at Cornell University, patented his parallax stereogram in 1903. He uses vertical slivers to control which part of a photograph is viewed through which eye, allowing for an interleaved photograph to be percept as a three-dimensional image without any additional optical devices needed. The patent [3] also covers the process of making such stereograms based on the same principle, not unsimilar to the one used in present digital light field cameras (see Figure 1). It should be noted that stereograms were already popular at that time, but consisted of two side-by-side images and required a special device, a stereoscope, to view them [4].

Many of us might have met light field photography in everyday life, be it early prints of 3D postcards based on the very principle of F. E. Ives, lenticular sheets used in the more advanced *xographs* of 1970s, or 'animated' rulers and bookmarks for children still available today.[4] One of the most prominent use of the direction of incoming light is the focusing process in photography. In classical photography, the split-screen and microprisms present on the focusing planes use light from different parts of the image to convert misfocus to spatial translation (shown on Figure 2), making it easier to focus correctly.[5] Similar principle is used by the autofocus algorithms using phase detection, where image of the autofocus point hits from different directions a pair of single-line CCD or CMOS sensors and the relative shift of these images determines the amount of misfocus.[6]

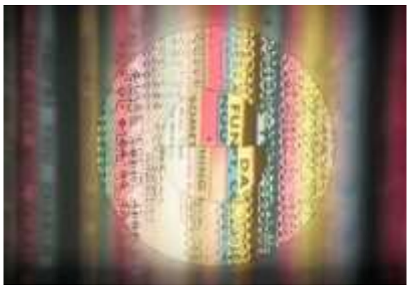


Figure 2. Split-screen and micro-prisms on the focusing plane of classical camera. Source: YouTube

All the applications mentioned above, however, use only few directions to supply an additional feature, while this work focuses on directional information being a fundamental part of the image itself. It was Gabriel Lipmann in 1908 who first came with the idea that lots of small images from slightly different place, when summed, reconstruct the original scene, and he named this method *integral imaging* [5]. At that time the light

[4] A comprehensive resource for postcard printing techniques and experiments through history would be the Metropolitan Postcard Club of New York City, particularly the Novelties chapter at http://www.metropostcard.com/techniques10.html.
[5] For more details on the optical background, see an article by Douglas A. Kerr available at http://dougkerr.net/Pumpkin/articles/Split_Prism.pdf.
[6] For interactive demonstration of this and other aspects of photography, check the applets of Stanford University available at http://graphics.stanford.edu/courses/cs178-10/applets.

field was being recorded on film and intended for viewing whole field by illuminating the film. Ives, Coffey, Dudnikov and others were later exploring and researching this idea further. This eventually led to the first digital light field camera in 1968 by Chutjian, seven years before the first classical digital camera by Kodak [6].[7] A deep walkthrough of the history of integral imaging is beyond scope of this work and would be reduplicating the great work already done by others. Starting points for interested readers would be Todor Georgiev's website[8], archive of University of Maryland[9] and INTEGRAL IMAGING slides by Stoykova & Sainov [7].

The Computer Science Department of Stanford University is pioneer in modern history of the light field, with Marc Levoy and Pat Hanrahan introducing the idea in their LIGHT FIELD RENDERING paper published in 1996 [8]. The rationale behind the paper is 'generating new views from arbitrary camera positions without depth information or feature matching, simply by combining and resampling the available images' [8]. At that time, the light field was of interest to Microsoft Research as well, in whose paper it was called *a Lumigraph* [9]. Both defined the light field as a function of four parameters and couple of papers followed exploring the choices of the parameterisation.

In 2006, Ren Ng submitted a Ph. D. dissertation to Stanford University titled DIGITAL LIGHT FIELD PHOTOGRAPHY [1], which is the basis for this thesis. In his work Ng suggests a simple model for the light field camera, analyses its performance and features. A microlenses array placed over standard digital camera sensor was used to build a working prototype of the camera and soon a start-up company, Lytro, Inc., was founded, producing the first light field cameras for consumer use.

1.2. Light Field

From physics point of view, different set of sources is usually cited referring to the light field. The major milestones would be Michael Faraday's THOUGHTS ON RAY VIBRATIONS where he proposes that light should be too interpreted as a field [10], Gershun's LIGHT FIELD, that introduces the term *light field* and formally defines it (although for the purposes of illuminating engineering) [11], and THE PLENOPTIC FUNCTION AND THE ELEMENTS OF EARLY VISION by Adelson and Bergen, which is a generic article defining a

[7] Technically the recording layer was still a light sensitive emulsion, but the image was computer generated.
[8] http://www.tgeorgiev.net/
[9] ftp://ftp.umiacs.umd.edu/pub/aagrawal/HistoryOfIntegralImaging/, most notable THE HISTORY OF INTEGRAL PRINT METHODS excerpt.

5D plenoptic function describing ‘everything that can be seen’ with references to psychophysical and physiological literature on vision [12].

For purposes of this thesis, however, light will be treated simply as a scalar value traveling along a ray, as is in the work of Ng.

1.2.1. Definitions

In all literature published so far, light field is explored in the context of a 4D space, which makes some principles difficult to illustrate and understand. For both educational and comprehensibility purposes, considerable attention will be paid to the light field in 2D space in this thesis too.

Light Field in 2D

As noted above, the light field photography is about recording direction from which the light is coming in addition to its intensity. Light field in 2D is therefore represented by one dimension for intensity and one for the direction. Based on the matrix methods in optics [13], the light ray hitting a plane is defined as

$$\vec{r} = \begin{bmatrix} x \\ \theta \end{bmatrix},$$

where x is the distance from the optical axis and θ is the angle from that axis, measured counter-clockwise (ref. Figure 3).

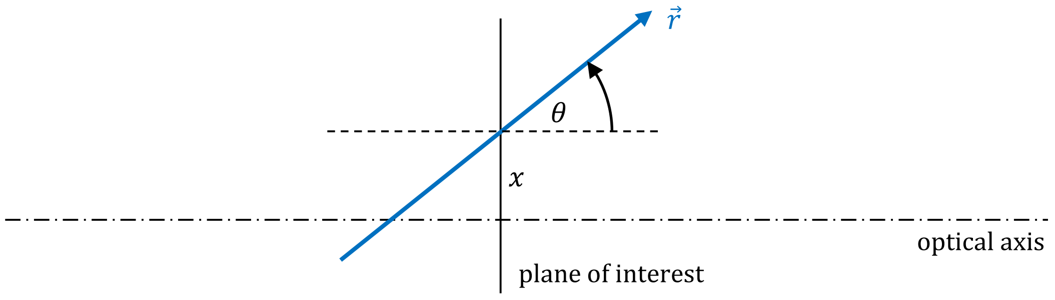


Figure 3. Light ray in 2D (angle parameterisation)

The angle of a ray is also defined by two points on two different planes. There are two planes of special interest in a camera – the lens plane and the sensor plane. So any individual ray is also defined by u, the distance from optical axis at lens plane, and x, the distance from optical axis at sensor plane (see Figure 4).

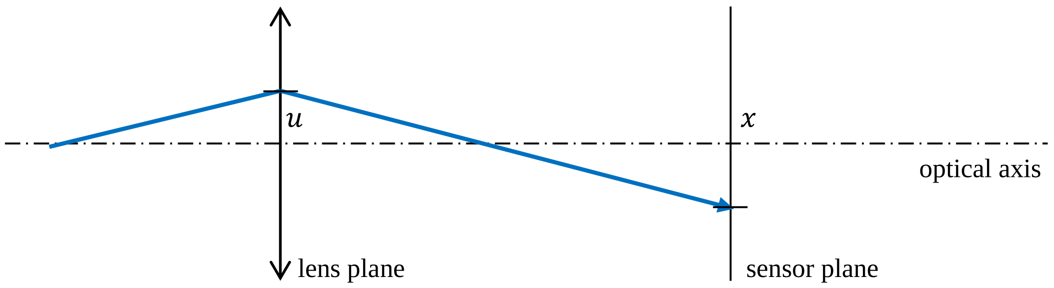


Figure 4. Light ray in 2D (planes parameterisation)

It is easy to see that for a ray hitting the sensor plane,

$$\theta = \tan^{-1}\frac{x-u}{d}$$

where d is the distance between the lens plane and the sensor plane.

Definition 1.1 (light field in 2D)

Let U and X denote two planes intersecting an optical axis with nonzero distance F between them. The **light field at the plane** X is defined as a function $L_F(x,u)$ giving the radiance of light ray coming through the plane U at the distance u from the optical axis and hitting the plane X at the distance x from the optical axis.

Light Field in 4D

The same principles apply to a 3D space with two dimensional planes intersecting the optical axis (cf. Figure 5).

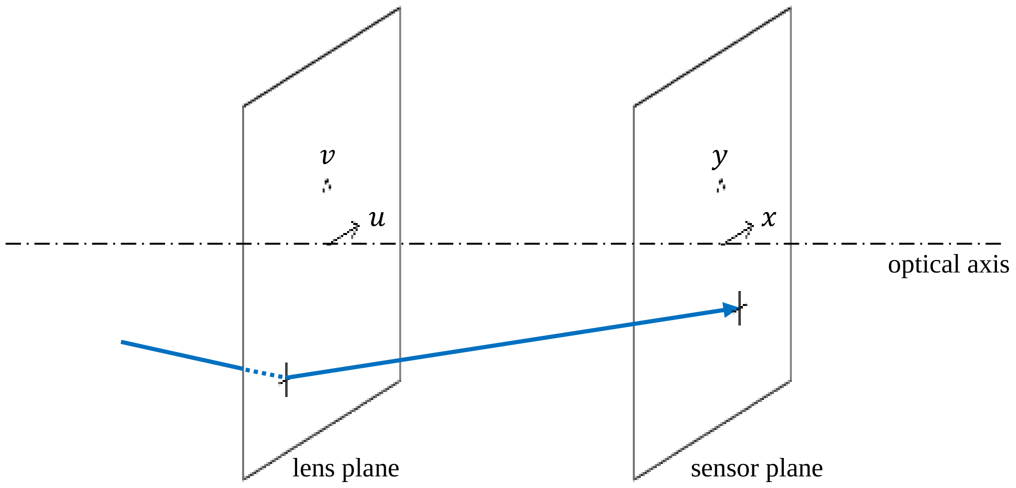


Figure 5. Light ray in 4D (planes parametrization)

Therefore, the definition of light field in 4D is analogous to the Definition 1.1.

Definition 1.2 (light field in 4D)

Let UV and XY denote two planes intersecting an optical axis with nonzero distance F between them. The **light field at the plane** XY is defined as a function $L_F(x, y, u, v)$ giving the radiance of light ray coming through the plane UV at the distance (u, v) from the optical axis and hitting the plane XY at the distance (x, y) from the optical axis.

Thin lens

Thin lens approximation of camera and other lenses will often be used through this work, so for reference, its basic equations follow:

$$\frac{1}{s_o} + \frac{1}{s_i} = \frac{1}{f} \qquad \text{(Gaussian formula)}$$

$$x_o x_i = f^2 \qquad \text{(Newtonian formula)}$$

$$\frac{y_i}{y_o} = -\frac{s_i}{s_o} = -\frac{f}{x_o} = -\frac{x_i}{f} \qquad \text{(lateral magnification)}$$

where

s_o is object distance from lens
s_i is image distance from lens
f is lens' focal length
$x_o = s_o - f$ and $x_i = s_i - f$
y_o is object distance from optical axis
y_i is image distance from optical axis

as depicted on Figure 6.

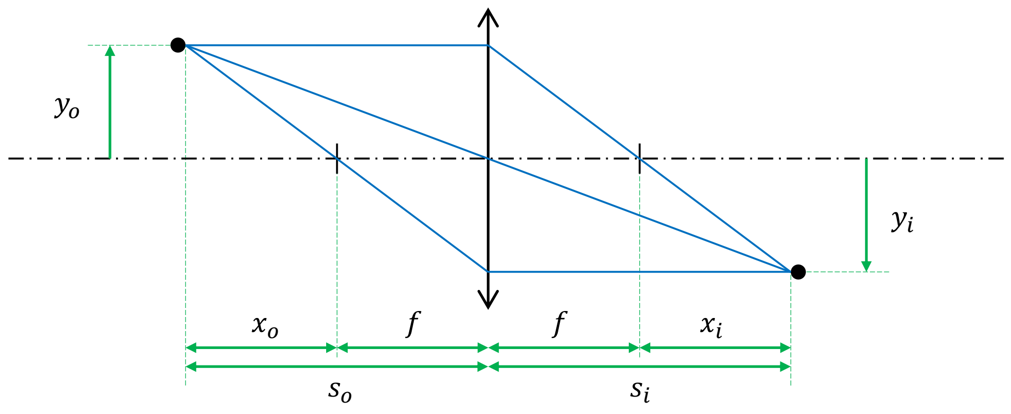


Figure 6. Thin lens

1.2.2. Image formation

Let's get some insight on how the light field looks inside. First, one might be interested how to render traditional image from it. In classical digital camera, where the direction of incoming light is not recorded, each pixel accumulates light from all directions.

For one-dimensional sensor, this means

$$E_F(x) = \int L_F(x,u)\, du$$

where $E_F(x)$ is the total irradiance of pixel x.[10]

Similarly for two-dimensional sensor,

$$E_F(x,y) = \iint L_F(x,y,u,v)\, du\, dv$$

where $E_F(x,y)$ is the total irradiance of pixel at (x,y).

1.2.3. Direction sampling

One way how to record light in different directions is to use a classical camera with narrow field of view and capture it from different locations. Figure 7 shows a single point displayed by two cameras modelled as thin lenses.

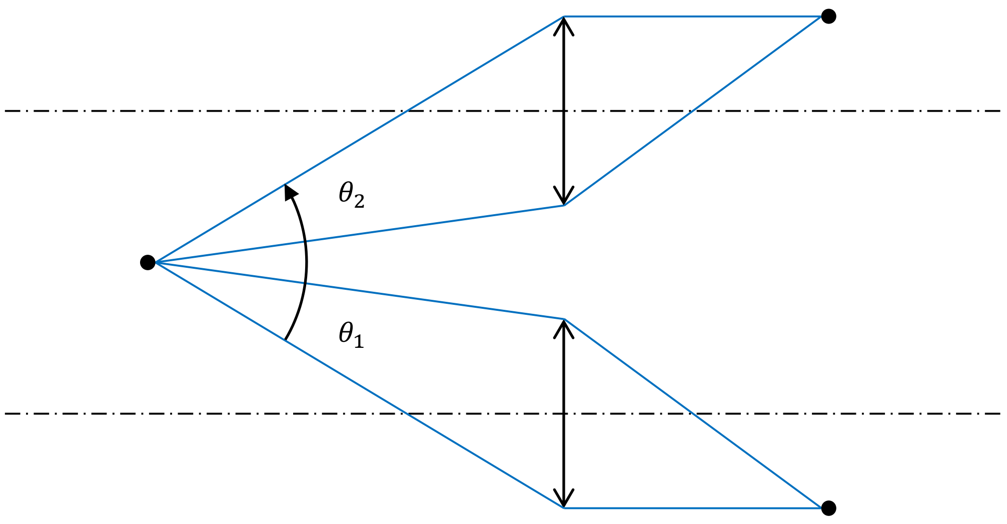


Figure 7. Single point from different locations

Another option would be to place the cameras in circle around the point, so that the angles θ_i would stay constant and the point would be observed from all the directions.

[10] Stroeber et al. shows that the image is in fact formed as $E_F(x) = \frac{1}{F^2}\int L_F(x,u)\cos^4\theta\, du$, but this is more a physical limitation. The cosine can either be absorbed into L_F [1] or eliminated by pixel microlenses [25] common on today's image sensors. The $\frac{1}{F^2}$ factor affects the overall image brightness only and is not interesting for the purposes of this work.

Although this approach is feasible for small objects and allows for their 3D models to be reconstructed [14], it is not available for everyday use in real-world photography.

The next important difference is the displayed image itself. If cameras were placed in a circle, all would display exactly the same image, a single point in the centre of the sensor. However, as clearly visible from Figure 7, in the case where cameras are arranged on a line, the point is displayed on different places of the sensor. This can be formulated more precisely: If a point O is at the distance y_o from the optical axis and at the distance s_o from the lens, then the lateral magnification formula gives distance

$$y_i = -\frac{s_i}{s_o} y_o = -\frac{f}{x_o} y_o$$

from the optical axis for point's image I on the sensor (assuming it is focused, i.e. the sensor is at distance s_i from the lens).

Now assume the point is fixed in space and only the camera is shifted in one direction. This offsets the point from optical axis of the camera in the opposite direction and therefore is equivalent to having the camera fixed and changing the point's y_o distance only.

Figure 8 shows the relationship given by lateral magnification formula.

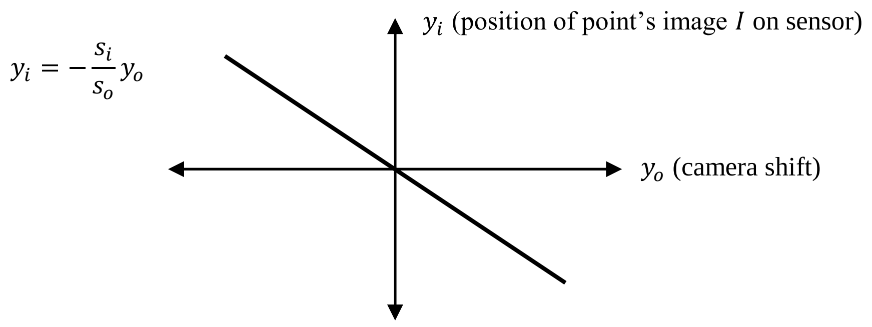


Figure 8. Position of single point on sensor from different camera locations

However, this gives us an important information about the point O. The sensor images allow us to compute *the distance of point* O *from the camera*. So given angle of the line φ with horizontal axis and the line equation $y = kx + q$ where $k = \tan\varphi$ is slope of the line, we have

$$\varphi = \tan^{-1} -\frac{s_i}{s_o}$$

$$\tan\varphi = -\frac{s_i}{s_o}$$

$$s_o \tan\varphi = -s_i$$

$$s_o = -\frac{s_i}{\tan\varphi}$$

the desired distance. Note that this expression might be a bit misleading, since s_i and s_o are in fact bound by the Gaussian formula and for a given lens, they cannot be manipulated independently. More clearly, for fixed f,

$$\frac{s_i}{s_o} = \frac{f}{s_o - f} = \frac{s_i - f}{f}$$

and

$$\tan\varphi = -\frac{f}{s_o - f}$$
$$\frac{1}{\tan\varphi} = \frac{f - s_o}{f}$$
$$s_o = f - \frac{f}{\tan\varphi}.$$

Alternatively,

$$\tan\varphi = -\frac{s_i - f}{f}$$
$$f\tan\varphi = f - s_i$$
$$s_i = f - f\tan\varphi$$

which gives us another important feature: if we want to move the sensor closer or farther to the lens (i.e. change s_i) it necessarily means a change to φ. Moving the sensor in this way actually means changing the focus of camera, so obviously to get an image focused at a different distance, one needs to change the slope of the line — more on this fundamental feature of light field later.

1.3. Acquisition Techniques

Before processing the light field, we will describe some methods of its recording in 4D space, where more dimensions bring more possibilities to the capture process.

1.3.1. Camera arrays

The natural extension of the suggestion given in the previous chapter is to arrange cameras on a plane. Since calibration and knowledge of the cameras' relative positions is important, this is much easier with multiple cameras fixed on a predefined grid. This configuration was used both by Stanford [8] and Microsoft Research [9]. An example of such camera array is depicted on Figure 9.

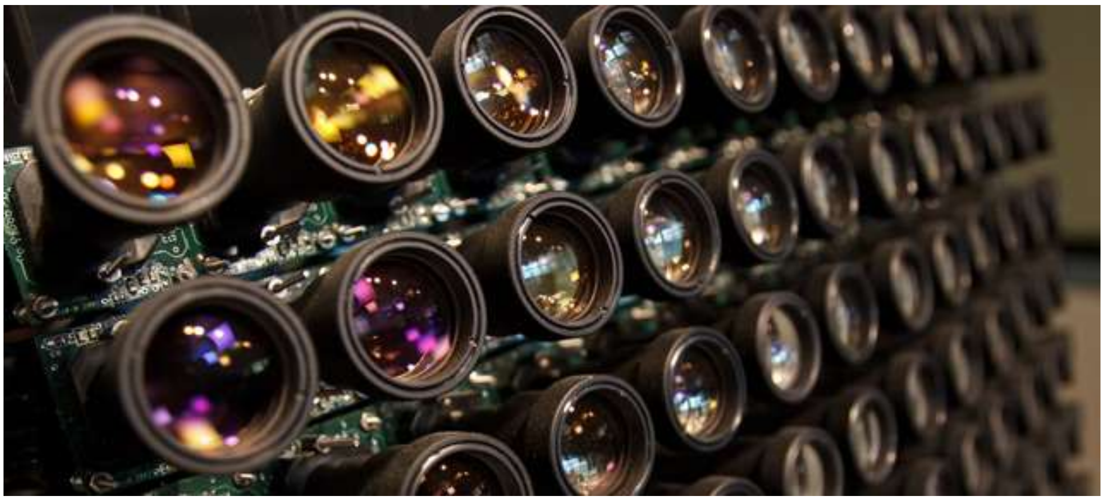


Figure 9. Stanford Multi-Camera Array. Photo by Eric Cheng. Source: [15]

The outcome of the array is an array of traditional photos, each capturing the scene from slightly different location, as illustrated by Figure 10. This figure (as well as figures 12 and 13) shows the complete 4D light field projected to 2D. In this case the outer axis is for the U and V dimensions (individual photos), while the inner axes show X and Y dimensions (pixels in each of the photos).

Therefore the individual photos, also known as *sub-aperture images*, will be referred to as **XY images**. While the spatial discretization is determined by the resolution of individual sensors, the directional information is discretized by the number of cameras themselves.

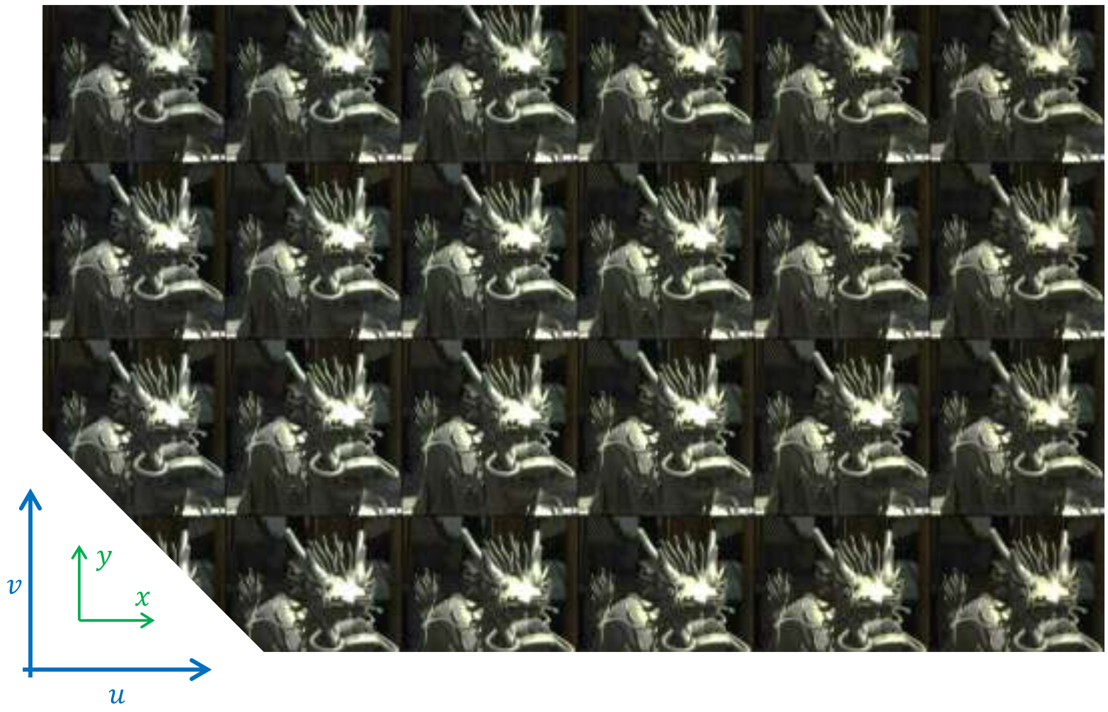


Figure 10. Output of the camera array. Sculpture in Forbidden City, Beijing, photo by author.

1.3.2. Microlens arrays

The same result can be obtained by replacing the cameras with single lenses and scaling them down to eventually cover one sensor. A piece of glass or plastics with thousands of small lenses, either engraved, moulded or eventually glued is called *a lenslet* or a microlens array (an example is on Figure 11).

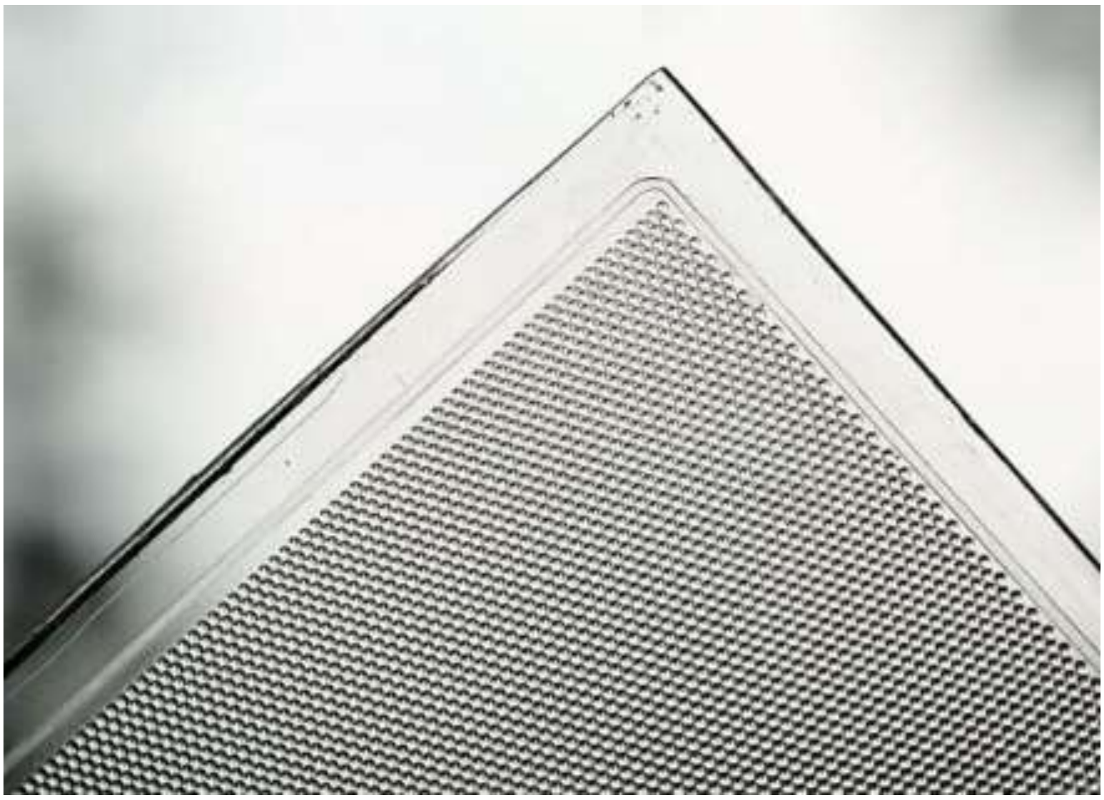


Figure 11. Microlens array. Manufactured and photographied by Mats Wernersson, with permission.

However, what if we wanted to sample the space the other way? What if, instead of all spatial information for one direction together, we wanted all directional information together for one quantum of spatial information? The trick to achieve this is to place the microlens array at the microlenses' focal length distance from the sensor, at most.[11]

This is the configuration chosen by Ren Ng. The overall image is still recognizable, but individual micro-images under each microlens can be observed (as seen on Figure 12). Here the outer axes denote X and Y dimensions (in hexagonal configuration in case of the Lytro camera) and the inner axes, within microlens images, U and V dimensions.

[11] Ng shows that the amount of directional information increases as the sensor distance approaches the focal length of the microlenses, with placing it immediately on the microlenses being equivalent to classical camera [1].

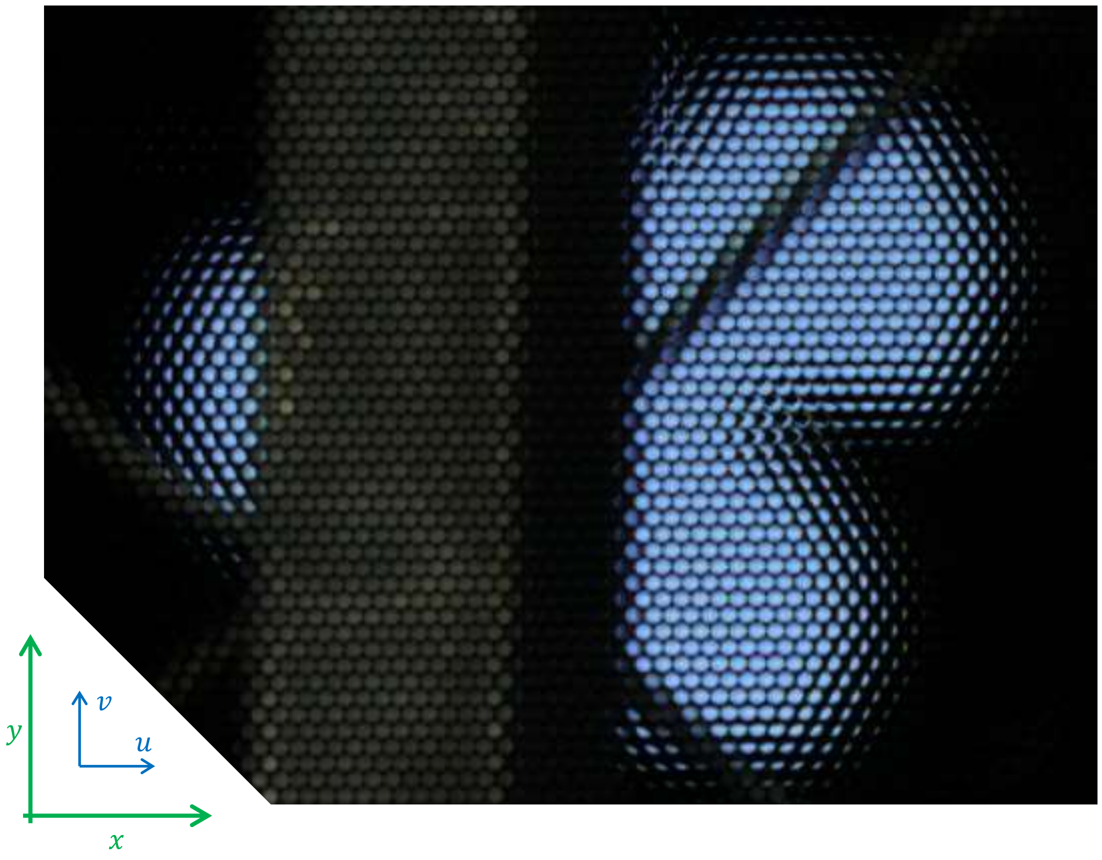


Figure 12. An enlargement of photo taken with hexagonal microlens array. A window at Cologne Cathedral, photo by author.

These individual images will be referred to as **UV images**. In this case the spatial resolution is given by the number of microlenses, while the directional information depends on the size of microlenses (discretized by the resolution, resp. number of pixels on the sensor underneath the microlens).

The relationship between UV images and XY images is rather simple and one can be easily converted to the other by simple transposition of pixels. The central XY image consists of central pixel under each microlens, the XY image right to the central one consists of pixel right to the central one under each microlens etc.

Having four dimensions, it might also be interesting to combine them mutually, i.e. to observe UX and VY images. These are known as *epipolar images* [1] and a sample is shown below.

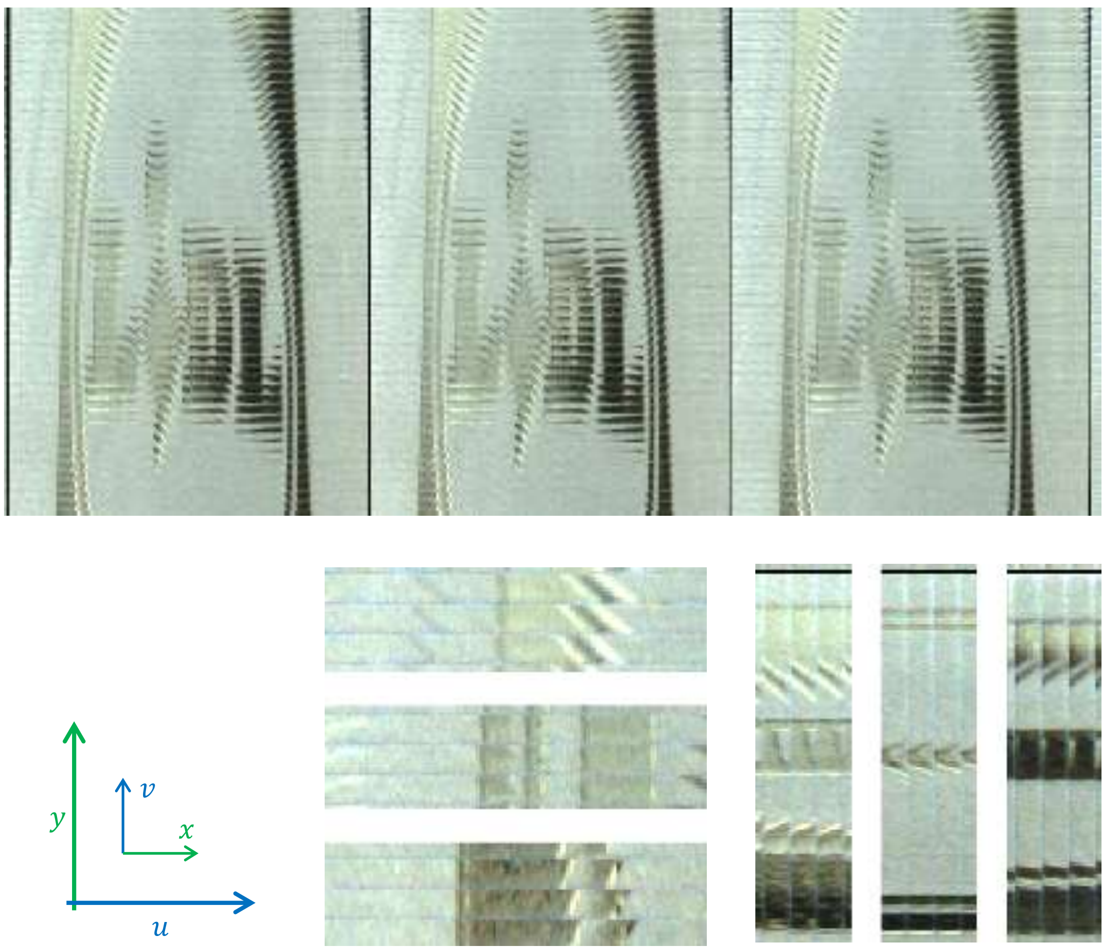


Figure 13. Cross-dimensional images. Lidl letterhead, photo by author.

A part of the UX image is shown on top of the picture. The number of columns corresponds to the size of microlenses (number of pixels they cover) and the number of rows in each column is equal to the number of microlenses. Similarly, in a VY image, the number of rows corresponding to size of the microlenses and number of columns in each row corresponding to their count in horizontal direction.

A detail of the UX image is bottom left and detail of a VY image bottom right. These individual rows and columns are 2D slices of the 4D space having one dimension spatial and one dimension directional just like it was introduced in the previous chapter. Note the changing slope of edges suggesting that different parts of the edge were at different distance from the camera, which is indeed the case due to the perspective composition (cf. Figure 14).

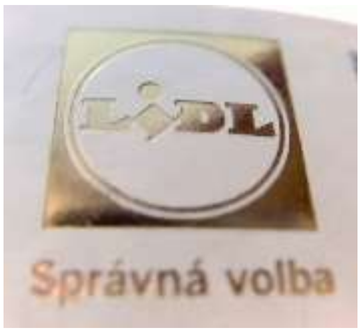


Figure 14. Rendered view of Figure 13

1.3.3. Emerging methods

The light field photography is currently experiencing boom in both consumer and academic communities and new technologies and techniques are being published monthly.

The biggest problem for light field photography today is trading the spatial resolution for the directional resolution. For example, for a 20 Mpx sensor with resolution of 5000×4000 pixels, which is above Canon's flagship digital camera[12], when directional resolution of 10 px is desired, it results in 500×400 pixels of spatial resolution, which is below the VGA standard and provides still only 10×10 pixels of directional information. Moreover, camera manufacturers basically stopped increasing the sensor resolution first because nobody needs that much and second because the lenses are actually at their resolution limits.[13]

Therefore the main interest is in capturing light fields without this trade-off. There are two ideas in this field worth mentioning. First is the patent of Panasonic Corporation, LIGHT FIELD IMAGE CAPTURE DEVICE AND IMAGE SENSOR [16], which places the microlens array behind the photosensitive layer, see Figure 15 for its schema. 'The micro lens layer is arranged so that light that has been transmitted through one of the photosensitive cells and then reflected from the reflective layer is incident on the same photosensitive cell again.' This allows the camera processor to associate depth information with the image, which, however, is 3D information only by definition, not a light field as defined in the field. On similar note, another patent uses electric voltage to adjust micolens focal length in order to turn them on and off, thus resulting again in two images, one full-resolution traditional photo and one low resolution in order to assign depth information to the individual pixels [16]. In order to achieve the refocusing effect, the processing software then needs to synthetically blur the part of images that shouldn't be focused. Popularity of this rather consumer feature made HTC introduce a first smartphone with a dual sensor camera, having one sensor dedicated for the depth information only.[14]

[12] Canon EOS-1D X has 18.1 Mpx as per Canon website.
[13] Resolving power of lens is out of scope of this work, some initial notes on this topic can be found e.g. at Michigan University's Physics Lecture Notes at http://www.pa.msu.edu/courses/2000fall/PHY232/. The important lesson is that the diffraction limit is a fundamental limit that cannot be solved by manufacturing process.
[14] Notes and some reviews on HTC One M8 available at http://lightfield-forum.com/tag/dual-camera/.

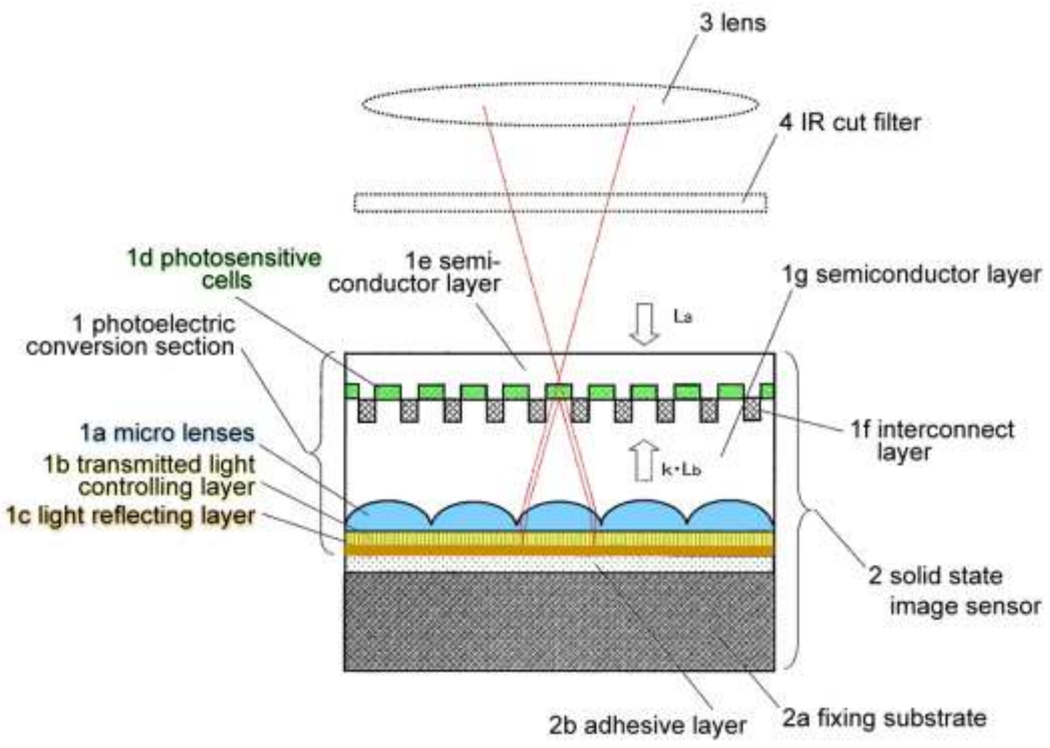


Figure 15. Panasonic's light field sensor. Colorized and annotated by Markus Nolf, with permission.

A slightly different approach was used by Boominathan et al. from Rice University to improve the spatial resolution [17] of captured light fields. They again use classical digital camera to capture high-resolution all-in-focus image of the same scene that was taken with low-resolution light-field camera, but in this case a pattern matching algorithm is used to replace the low-resolution pieces with those from the high-resolution photo, so the original light-field features are preserved within some error.

The second direction to be noted is called *compressive light field photography* and is developed by researchers at MIT Media Lab.[15] While the ideas above build on the Lippmann's idea of microlenses, MIT chose the model of blocking light introduced by Ives. A more or less random pattern on semi-transparent mask is placed before the sensor that causes various shadows to be cast over it which allows for reconstruction of the light field [18]. The interesting factor with this method is that everybody can print the mask for a fraction of the price that microlens array costs, and apply it to the camera themself (like on Figure 16).

[15] http://web.media.mit.edu/~gordonw/CompressiveLightFieldPhotography/

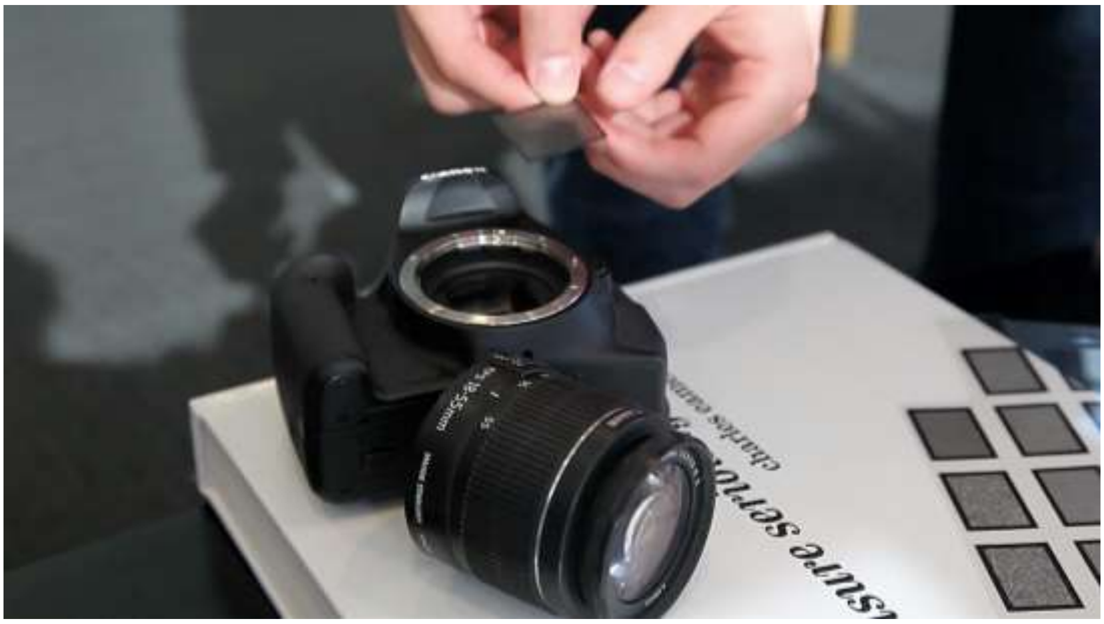


Figure 16. A printed coded mask being applied to a standard Canon camera. Photo courtesy by Kshitij Marwah, with permission.

The key observance here is that light field is highly redundant and can be decomposed into weighed sum of predefined base patches, much like how the cosine or Fourier transform works in image compression. So instead of recording the raw light field, only few coefficients for the directional information are recorded [19]. This approach has also further applications in processing, such as light field compression or denoising [18].

The Camera Culture Group to which this research belongs works on glasses-free 3D displays, which requires "completely integrated pipeline from live-action shooting to editing to display" and the discussed light field camera design emerged as one of the technologies for that pipeline [19]. Like with Lytro Inc., a spin-off company Tesseract Imaging[16] was started, with the design being named *FOCII*.

[16] http://tesseract.in/

1.4. Processing and Rendering

Now when we know how to record the light field, we can proceed to rendering it. Since principles of the Lytro camera is major topic of the thesis and also the only one available to the author, we will adhere to the microlenses as used in the Lytro camera (for corresponding 2D model see Figure 17).

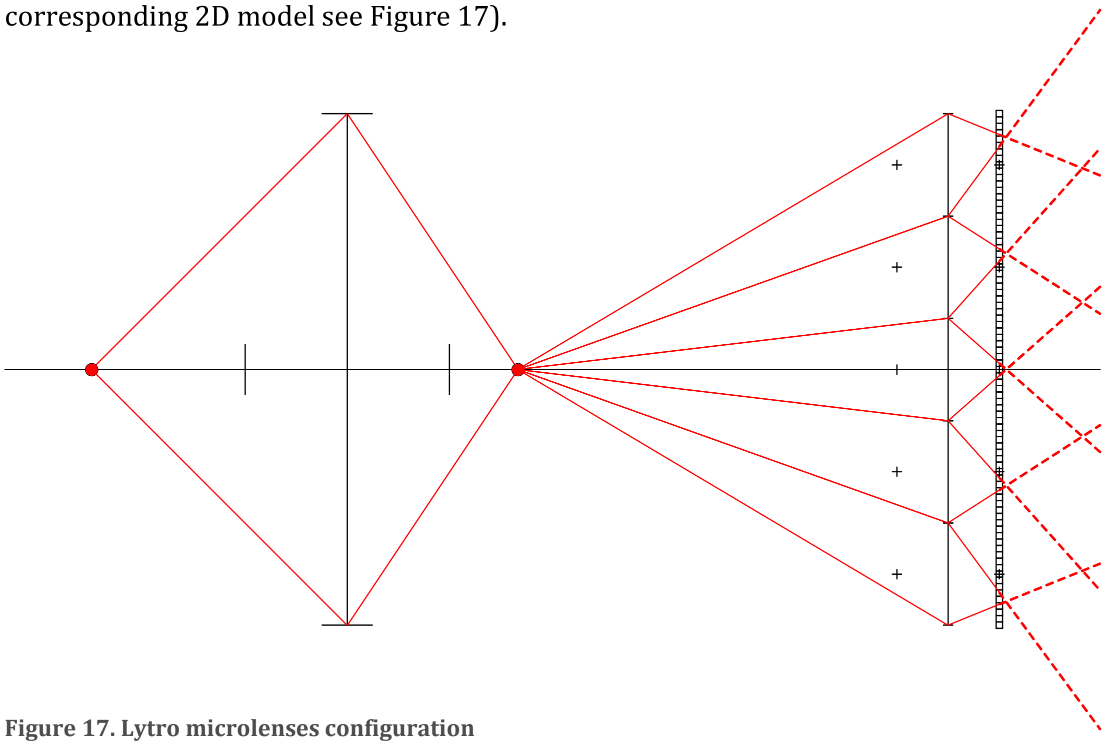


Figure 17. Lytro microlenses configuration

There are three planes of interest:

the *lens* plane, which is a thin lens approximation of the main camera lens;

the *mla* plane, which is a thin lens approximation of the microlens array;

the *sensor* plane.

We will use

f_{lens}	for the focal length of the lens;
f_{mla}	for the focal length of the microlenses;
d	for the distance between the *lens* plane and the *mla* plane;
Δ_{mla}	for the height (pitch) of individual microlenses;
Δ_{px}	for the height (pitch) of individual pixels on the sensor.

The distance between the *mla* plane and the *sensor* plane is fixed to f_{mla}.

In line with previous chapters, we introduce discrete indices

x to identify single microlens, zero being the one at optical axis;

u to identify single pixel under single microlens, zero being the centre one.

1.4.1. Rendering

The images under individual microlenses can be arranged next to each other to form the UX image. For example, a single point at optical axis focused on the microlens array will fall on pixels under the microlens at optical axis ($x = 0$), as in Figure 18.

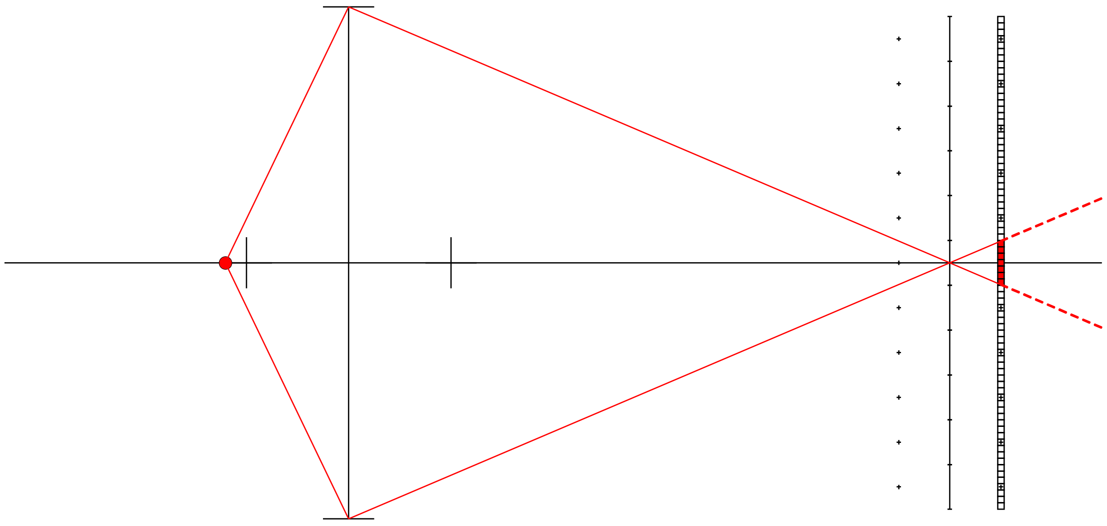


Figure 18. Focused point, overall 2D view

Note that in order to cover exactly all the pixels under given microlens, the following equation, where Δ_{lens} is height of the main lens, must hold:

$$\frac{\Delta_{lens}}{d} = \frac{\Delta_{mla}}{f_{mla}}$$

Putting the individual images next to each other will therefore result in a single vertical line as in Figure 19 top.

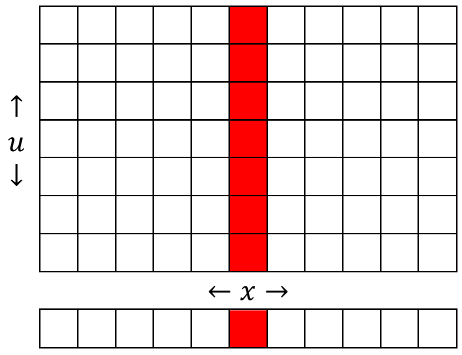


Figure 19. Focused point, UX view

As discussed in chapter 1.2.2 Image formation, the original 1D image can be restored by integration light from all the directions, which corresponds to summing all the pixels under individual microlenses, resulting in a single pixel representing the focused point (Figure 19 bottom).[17]

Similar in four dimensions, the original 2D image can be rendered by summing all the pixels under individual microlenses.

We have already seen that the slope of the line corresponds to the distance of the point from the lens. An example of single misfocused point on the optical axis is shown on Figure 20.

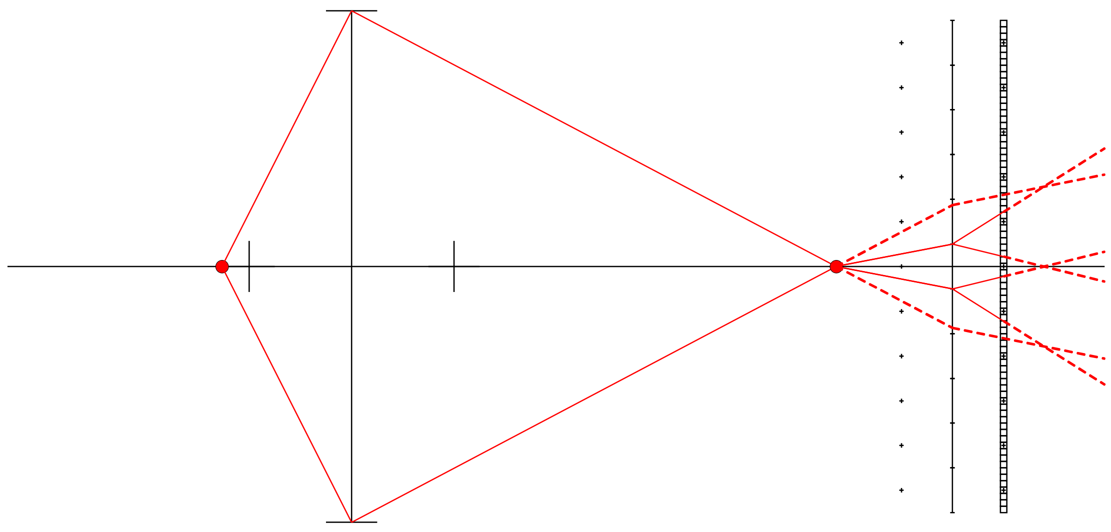


Figure 20. Misfocused point, overall 2D view

The corresponding UX image is at the top of the Figure 21, which illustrates the properties derived in the previous chapter.

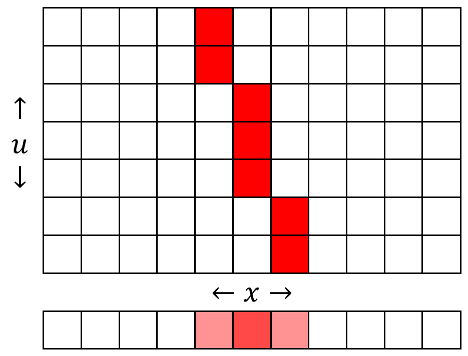


Figure 21. Misfocused point, UX view

When rendered as described above, it results in a blurred image of the point like at the bottom of Figure 21, as expected for a point out of focus.

[17] The sum must be normalized to fit into the sensor value range for further processing, which is equivalent to taking average value of the pixels under individual microlenses instead.

1.4.2. Sensor equations

Although the fact that UX image displays points as rasterized lines is suggested in Ren Ng's dissertation [1], we will take the steps to prove this critical feature. Two properties need to be proven – that the number of illuminated pixels under each microlens is equal and that such pixels under neighbouring microlenses are adjacent to each other.

Figure 22 shows a point displayed through a single microlens in the array, modelled by a thin lens, and a sensor (red plane) at fixed distance from the microlens.

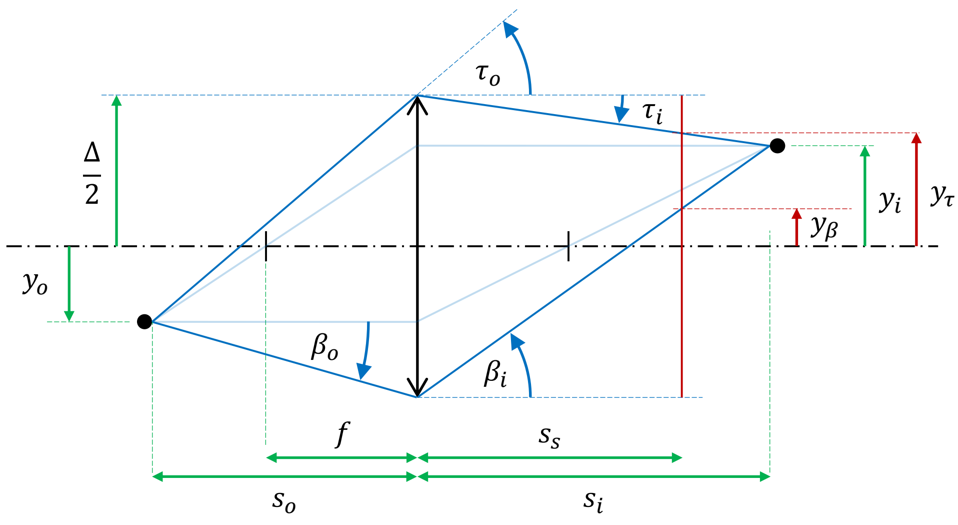


Figure 22. Image formation on a sensor

We denote

Δ	the height of the lens;
f	the focal length of the lens;
y_o	the distance of point from the optical axis;
s_o	the distance of point from the lens;
y_i	the distance of point's image from the optical axis;
s_i	the distance of point's image from the lens;
τ_o	the angle between the optical axis and ray coming from the point through the top of the lens (i.e. at distance $\Delta/2$ from the optical axis);
τ_i	the angle between the optical axis and ray coming from the top of the lens through the point's image;
β_o	the angle between the optical axis and ray coming from the point through the bottom of the lens (i.e. at distance $-\Delta/2$ from the optical axis);
β_i	the angle between the optical axis and ray coming from the bottom of the lens through the point's image;

s_s	the distance of the sensor from the lens;
y_τ	the distance from the optical axis where the ray coming from the top of the lens hits sensor;
y_β	the distance from the optical axis where the ray coming from the bottom of the lens hits sensor.

Specially, if the sensor has the point in focus, $s_o = s_s$ and $y_\tau = y_\beta = y_i$. If not, the point causes a line segment of $y_\tau - y_\beta$ length to appear on the sensor.

We want to show where a point at distance y_o from the optical axis and s_o from the lens will appear on the sensor at the distance s_s from the lens (i.e. find y_τ and y_β).

First we will derive the generic thin lens ray transfer formula, that is, if a ray from an object hits the lens under angle φ_o at the distance y from the optical axis, at what angle φ_i will the ray leave the lens (like on Figure 23).

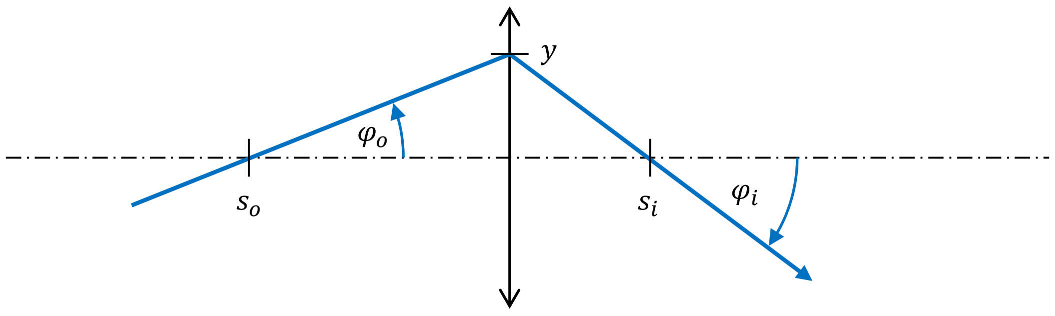


Figure 23. Thin lens ray transfer

It is easy to see that $\tan\varphi_o = \frac{y}{s_o}$ and $\tan\varphi_i = -\frac{y}{s_i}$ and the Gaussian formula gives us

$$\tan\varphi_i = -\frac{y}{\frac{f\,s_o}{s_o - f}} = \frac{y(f - s_o)}{f\,s_o} = \frac{y}{s_o} - \frac{y}{f} = \tan\varphi_o - \frac{y}{f}.$$

We will demonstrate how to use these formulæ to arrive at y_β. First, we note that

$$\frac{-\frac{\Delta}{2} - y_o}{s_o} = \tan\beta_o\,.$$

Then,

$$\tan\beta_i = \tan\beta_o - \frac{-\frac{\Delta}{2}}{f} = \frac{-\frac{\Delta}{2} - y_o}{s_o} + \frac{\Delta}{2f}$$

and again by expressing

$$\frac{\frac{\Delta}{2} + y_\beta}{s_s} = \tan \beta_i$$

we get

$$y_\beta = s_s \tan \beta_i - \frac{\Delta}{2} = s_s \left(\frac{-\frac{\Delta}{2} - y_o}{s_o} + \frac{\Delta}{2f} \right) - \frac{\Delta}{2} = \frac{s_s \left(-\frac{f\left(\frac{\Delta}{2} + y_o\right)}{s_o} + \frac{\Delta}{2} \right)}{f} - \frac{\Delta}{2}.$$

Analogously for y_τ we get

$$y_\tau = s_s \tan \tau_i + \frac{\Delta}{2} = s_s \left(\frac{\frac{\Delta}{2} - y_o}{s_o} - \frac{\Delta}{2f} \right) + \frac{\Delta}{2} = \frac{s_s \left(\frac{f\left(\frac{\Delta}{2} - y_o\right)}{s_o} - \frac{\Delta}{2} \right)}{f} + \frac{\Delta}{2}.$$

The latter forms make it easy to evaluate the special cases of $s_s = f$ and $s_o = f$.

Now the length of the line segment captured by the sensor is

$$y_\tau - y_\beta = s_s \left(\frac{\frac{\Delta}{2} - y_o}{s_o} - \frac{\Delta}{2f} \right) + \frac{\Delta}{2} - s_s \left(\frac{-\frac{\Delta}{2} - y_o}{s_o} + \frac{\Delta}{2f} \right) + \frac{\Delta}{2} =$$

$$= \frac{s_s \Delta}{2 s_o} + \frac{s_s \Delta}{2 s_o} - \frac{s_s y_o}{s_o} + \frac{s_s y_o}{s_o} - \frac{s_s \Delta}{2f} - \frac{s_s \Delta}{2f} + \frac{\Delta}{2} + \frac{\Delta}{2} =$$

$$= \frac{s_s \Delta}{s_o} - \frac{s_s \Delta}{f} + \Delta$$

pixels. The important observation is that the length does not depend on y_o, the distance of the point from optical axis, which also means it does not depend on the microlens position.

The centre of line segment is at distance

$$y_c = y_\beta + \frac{y_\tau - y_\beta}{2} = -\frac{s_s \Delta}{2 s_o} - \frac{s_s y_o}{s_o} + \frac{s_s \Delta}{2f} - \frac{\Delta}{2} + \frac{\frac{s_s \Delta}{s_o} - \frac{s_s \Delta}{f} + \Delta}{2} = -\frac{s_s}{s_o} y_o$$

from the optical axis (which is the same equation we arrived to in chapter 1.2.3 Direction sampling).

Note that if a point is at the focal length distance from the lens ($s_o = f$), its image will be in infinity, and will cover all the pixels under the lens, regardless of how far the sensor is located, which is in agreement with $y_\tau - y_\beta = \Delta$.

As described in the previous chapter, the Lytro camera has the sensor at the focal length distance ($s_s = f$), which further simplifies the equations:

$$y_\tau = \frac{f\left(\frac{\Delta}{2} - y_o\right)}{s_o} \qquad y_\beta = -\frac{f\left(\frac{\Delta}{2} + y_o\right)}{s_o} \qquad y_\tau - y_\beta = \frac{f\Delta}{s_o} \qquad y_c = -\frac{f}{s_o} y_o$$

In this case $y_c = -y_o$ for the point at focal length distance.

What remains to be proven is that the top ray from one microlens hits the sensor at the same location as the bottom ray from the microlens above, in other words y_τ for given y_o is equal to y_β for $y_o - \Delta$. But

$$y_{\beta-\Delta} = -\frac{f\left(\frac{\Delta}{2} + (y_o - \Delta)\right)}{s_o} = -\frac{f\left(-\frac{\Delta}{2} + y_o\right)}{s_o} = \frac{f\left(\frac{\Delta}{2} - y_o\right)}{s_o} = y_\tau$$

So if we have a microlens array and put the sensor images from each microlens side by side, the line segments will never overlap and at the point the segment from one microlens ends, the line segment from neighbouring microlens starts. [18]

Together with the fact that each microlens will display the point using the same amount of pixels, it allows us to treat the resulting image as rasterization of a line going through the centres y_c.

1.4.3. Parallax

The simplest view is to select one direction of interest and ignore light from all others, in our case that means to use data from single u only, picking the same one pixel under each microlens, which is called sub-aperture image and illustrated on Figure 24.

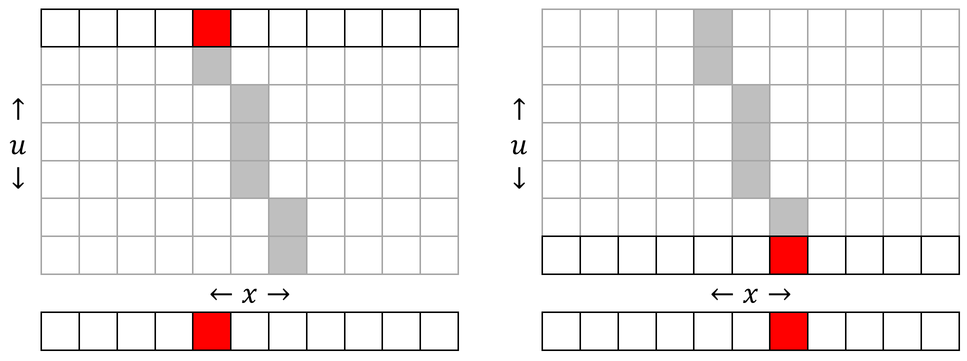


Figure 24. Subaperture image, UX view

[18] Assuming each mirolens has its own infinitely sized sensor. In practical embodiments, all microlens share a single sensor and care must be taken to ensure that rays from one microlens do not appear under another microlens. Apart from careful camera design this can be solved by mechanical barriers [25] or compensated in post processing [1].

Intuitively, this will render the scene from that direction only, resulting in the rendered point being at different locations in the image. If we had two points of various distance in the scene, they would be represented by two differently slanted lines, like on Figure 25. According to the theory, the red one (lighter) is nearer to the lens than the blue one (darker).

Similar to two aligned objects viewed from distinct locations, the relative position of the two points depends on the direction the observer is looking. When the observer is in line with the objects, the nearer object occludes the farther as in the case of $u = 0$ in the picture. This difference in the apparent position is known as *parallax*.

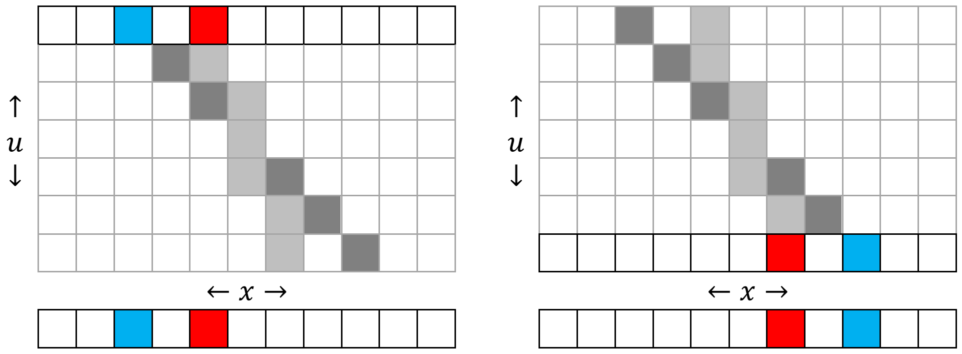


Figure 25. Subaperture image of two points, UX view

Figure 26 shows this effect rendered from a 4D light field — difference between the two pictures is most obvious on framing of the reflector relative to the cabin in background. The parallax can be used to generate 3D images using extreme left and right views, which readers can see on Figure 26 by crossing their eyes to merge the two views into one image.

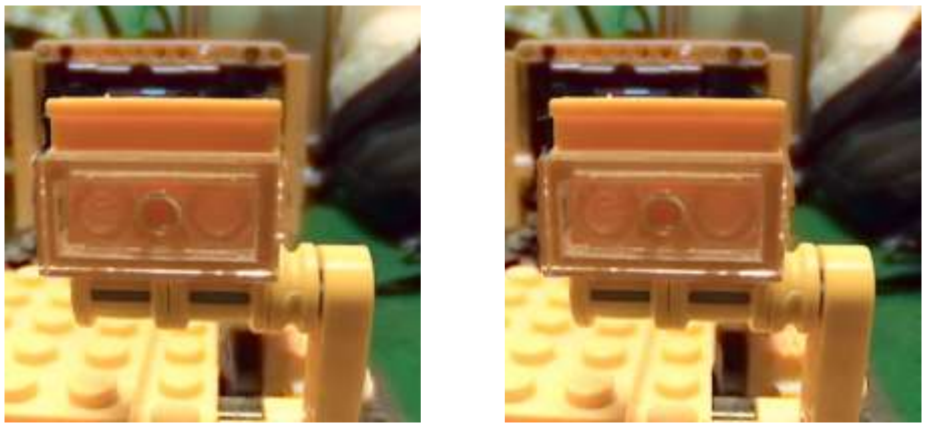


Figure 26. Parallax in 4D, detail of 8275 bulldozer by LEGO®, photo by author.

1.4.4. Depth of field

In traditional photography, the depth of field is controlled by the size of aperture. Reducing the aperture diameter effectively reduces the directional information coming through the camera. Smaller aperture in our model can be seen on Figure 27.

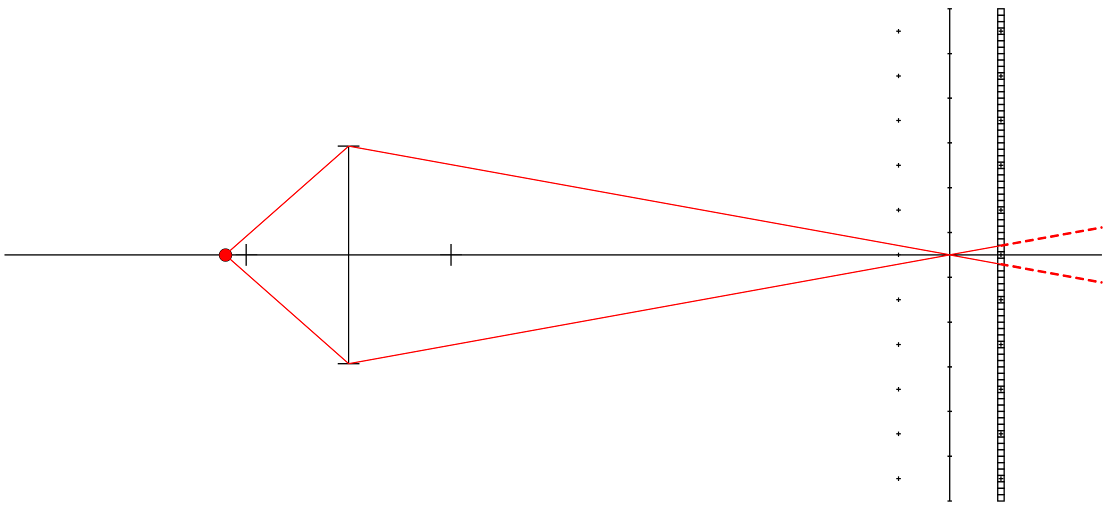


Figure 27. Small aperture, overall view

The corresponding UX view follows on Figure 28. Note that the total amount of light is decreased, too.

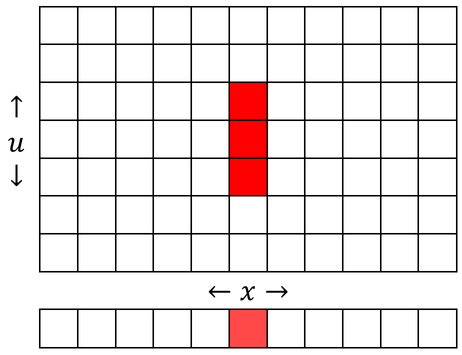


Figure 28. Small aperture, UX view

Specially,

$$px_{aperture} = \frac{\Delta_{lens} f_{mla}}{d \Delta_{px}}$$

is the number of pixels that cover the image of aperture. Hence the light field can be rendered with smaller aperture by simply taking only corresponding subset of the directional data, as shown in Figure 29 left.

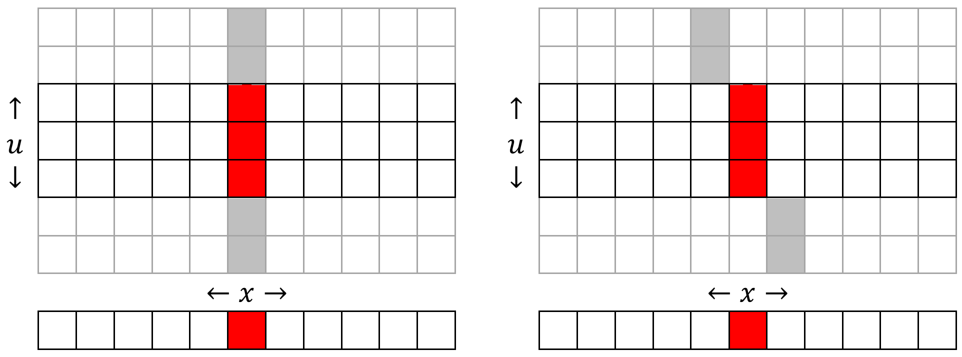


Figure 29. Small aperture synthetised, UX view

On the right side of Figure 29, the same process is applied to the misfocused point, where it helps to bring it into focus, as expected when using smaller aperture.

This effect is well known in the 4D case, an example of extending the depth of field is depicted on Figure 30.

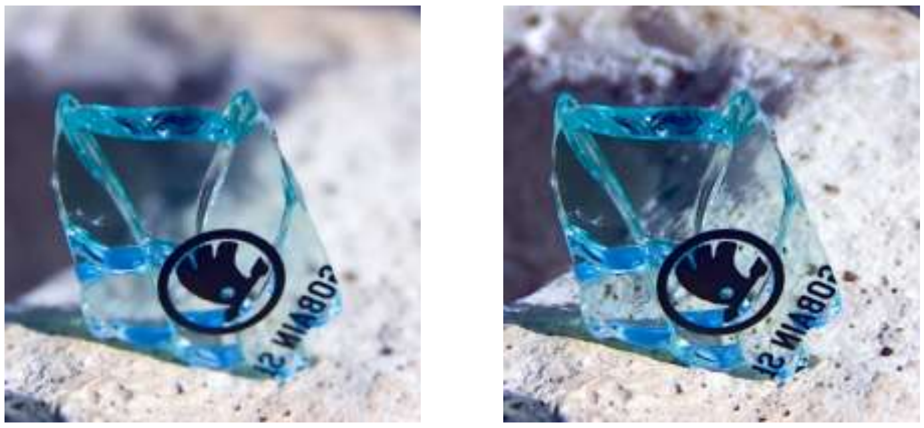


Figure 30. Narrow and extended depth of field in 4D, sliver of car window, photo by author.

Another observation is that while in classical photography the aperture size can be adjusted only in symmetrical way, with light field we can render it asymmetrical as well (e.g. using positive u only, or even non-contiguous bands of u).

1.4.5. Refocusing

It is obvious from the picture of misfocused point that in order to bring the point back into focus, we could skew the UX image, like on Figure 31.

This conforms to the theory developed in chapter 1.2.3 Direction sampling. More formally, the skewed imaging equation would have the form of

$$E_{deskewed}(x) = \sum_u L_d\left(x + \frac{u}{k}, u\right)$$

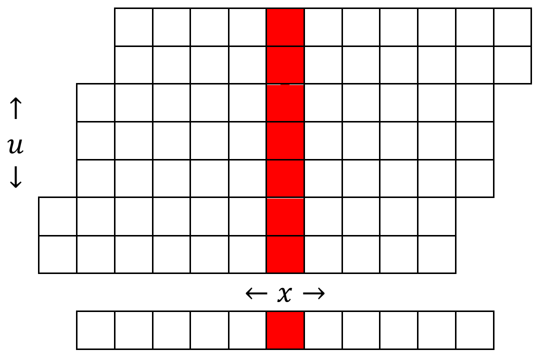


Figure 31. Refocused point, UX view

where k is the skewing factor, equal to slope of the line and to the number of pixels that the point falls on as we have shown with the sensor equations. By similar triangles, we can find the focused image point, i.e. the distance s_i from mla plane at which the point is focused:

$$\frac{\Delta_{mla} - \Delta_{px}k}{f} = \frac{\Delta_{mla}}{s_i}$$

Solving for k yields

$$k = \frac{\Delta_{mla} - \frac{f\Delta_{mla}}{s_i}}{\Delta_{px}}$$

and with the help of Gaussian formula we get the desired k to bring the plane at $d' = d - s_o$ into focus:

$$k = \frac{\Delta_{mla} - \frac{f\Delta_{mla}(s_o - f)}{fs_o}}{\Delta_{px}} = \frac{\Delta_{mla} - \frac{\Delta_{mla}s_o}{s_o} + \frac{\Delta_{mla}f}{s_o}}{\Delta_{px}} = \frac{f\Delta_{mla}}{s_o\Delta_{px}}$$

The same relationships applies to 4D, where

$$E_{d'}(x,y) = \sum_u \sum_v L_d\left(x + \frac{u}{k}, y + \frac{v}{k}, u, v\right).$$

For an example of refocused images rendered from 4D data, see Figure 32.

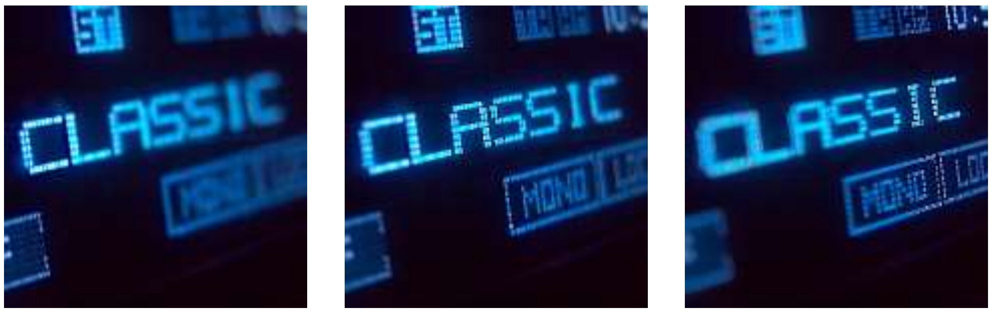


Figure 32. Refocusing in 4D, head unit by Pioneer, photo by author.

1.4.6. Depth map

A popular application of light field data is getting the depth map of the scene. This feature is requested so much that new imaging devices with separate depth sensors are emerging as an intermediate step between traditional and light field photography (see next chapter for details).

One of the easier methods to generate a depth map from the scene is taking the rendered pictures focused at different distances, and for each area of interest, determining the one that has the highest local contrast. An example of more sophisticated algorithm would be the one of Liang et al. using occlusion maps [20]. Figure 33 shows a depth map generated by Lytro software and using it to generate a 3D surface of the captured scene. Dark shades denote areas nearer to the camera, light shades the farther areas.

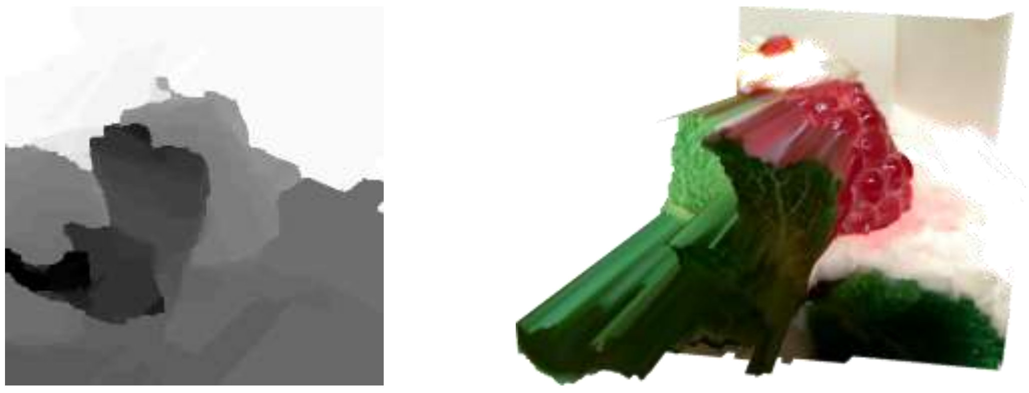
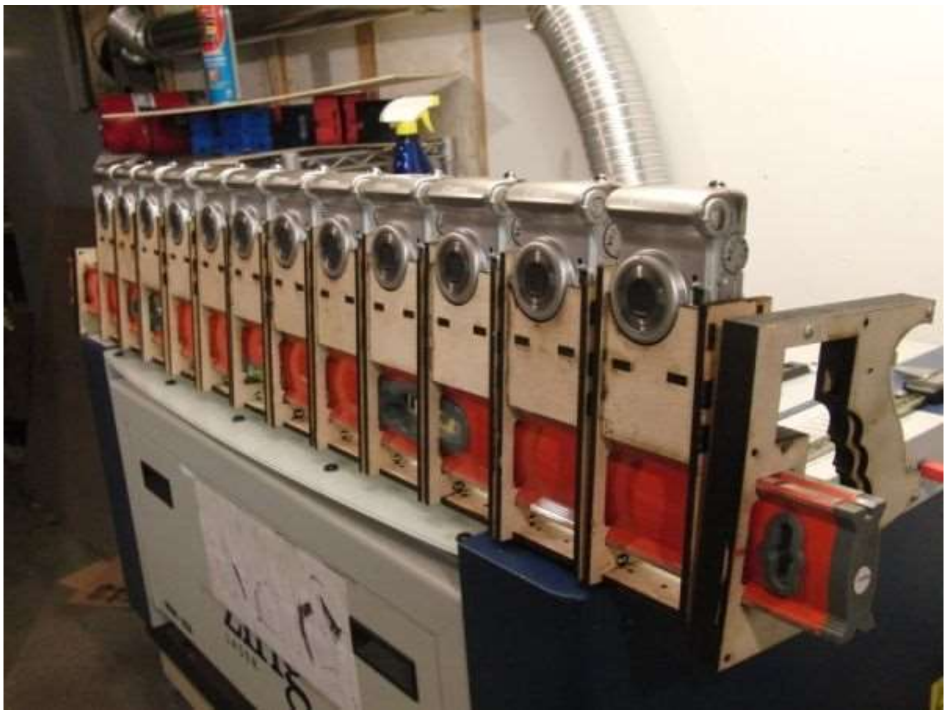


Figure 33. 2D depth map (left) and spatial data mapped to 3D (right), a raspberry cake, photo by author. Rendered by software accompanying the thesis.

Let me stress at this point that depth maps as researched in the field are an oversimplified problem. A scene can very easily contain points that can be focused at multiple depths, either due to translucent materials, or because of reflections, and 2D depth maps cannot cover these conditions, despite their everyday occurrence in the real world.

While depth maps can be built even from stereo images [21], the light field allows reconstructing true volumetric depth information about the scene. Extending our previous method, the third dimension would simply be the value of local contrast itself rather than finding its maximum. Note that this cannot produce a 3D model of the scene (neither for transparent objects), as both refraction and reflection cause images to appear at different locations than they come from, but light field software could take advantage of this information, for example when determining the distances that can be rendered.



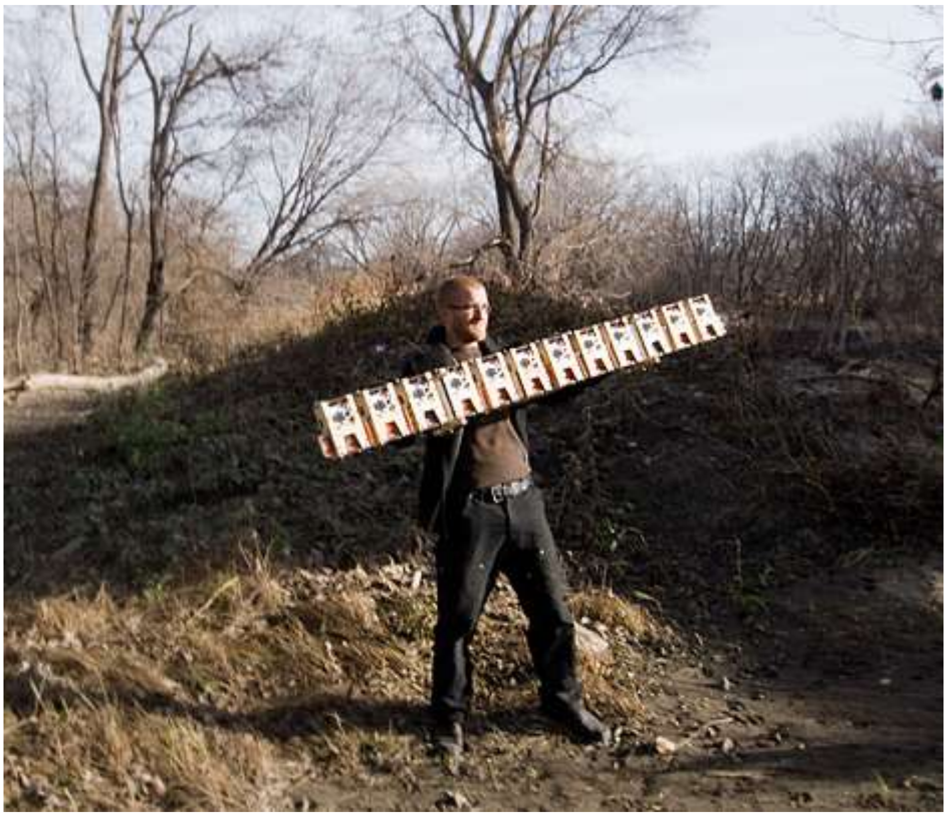


Figure 34. Daniel Reetz holding light field camera array they built together with Matti Kariluoma in 2009, with permission. Check www.futurepicture.org for more details on the camera.

Past couple of years, various commercial companies are announcing their interest in the light field technology for both recording and displaying, be it big established players like Adobe Systems or NVIDIA Corporation, fresh start-ups like Tesseract Imaging or Pelican Imaging or silicon vendors like Toshiba Corporation. It is impossible to give exhaustive and up-to-date listing here – interested readers can keep track of the technology on sites like www.lightfield-forum.com – but few noteworthy trends can be briefly discussed.

2.1. Software Imitations

The refocusing ability of light field photography is so popular that several phone manufactures decided to equip their devices with an imitation of software refocusing.

Different vendors did it in different ways. *Nokia Refocus*[19] captures couple of photos in fast succession, each focused at different plane and then offers refocusing by selecting the one frame of interest, or all-in-focus by blending all the frames. The main disadvantage of this solution is that each frame is captured at different time, so changes in the scene can be visible due to refocusing.[20] New types of camera modules are being manufactured to bring refocusing times to minimum, such as *mems|cam* by DigitalOptics Corporation.[21]

Another approach is to obtain depth information together with a single photo and then use it to synthetically blur parts of the image that are at different depth than the one to be focused. Toshiba Corporation has introduced a new dual camera module for this purpose[22], while HTC started to engage their phones with a dedicated depth sensors.[23]

Popular applications like separating the background from a photograph, applying different effects to the background and foreground or photographic collages are still possible with these solutions, but parallax view (that allows for 3D photography) or realistic out-of-focus appearance can't be delivered without true light field data.

2.2. Pelican Imaging

In 2013 Pelican Imaging Corporation published their new camera array module targeting especially mobile devices [22], which has couple of interesting properties. A photo of the module is on Figure 35.

[19] http://refocus.nokia.com/
[20] see e.g. http://www.cnet.com/news/nokia-vs-lytro-the-refocusing-challenge/
[21] http://www.memscam.com/
[22] TCM9518MD
[23] e.g. HTC One M8

Most importantly it is a true camera array supplying 4D light field data. The directional resolution is 4×4, the native spatial resolution is 1,000×750 pixels, increased by super resolution methods to 3,264×2,448 pixels.

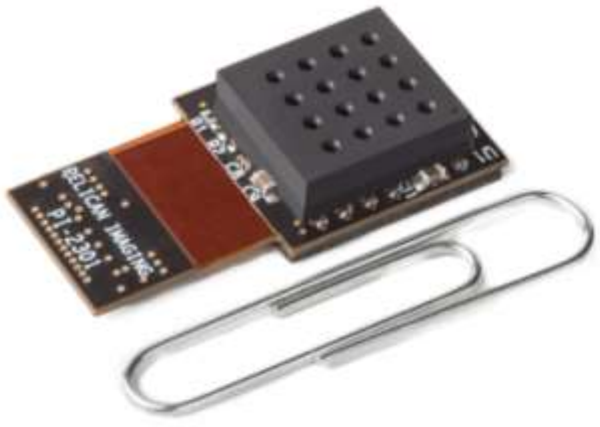


Figure 35. PiCam monolithic camera array, press release photo.

This module is notable especially for the technological challenges it addresses. The smaller aperture lens allowed to reduce physical height of the module, aberration errors, and shortened hyperfocal distance. Finally, each camera in the array is optically isolated and records single colour only, which not only provides better colour fidelity than traditional Bayern filter pattern, but also allows for higher performance optics due to the reduced bandwidth they need to transfer. [22]

2.3. Lytro

All the recent boom of products and investment into light field technology for consumer market started with Lytro[24] founded by Ren Ng after his Ph.D. research in 2006. Six years later they released first light field camera for consumers, called the *Lytro camera*.

Figure 36. First generation Lytro camera, press release pohoto.

[24] http://www.lytro.com/

The camera (on Figure 36) has about 10×10 directional resolution and roughly 328×380 pixels of spatial resolution in hexagonal configuration, increased by super resolution methods to 1080×1080 pixels. The release price was $499 for model with 16 GB memory and $399 for model with 8 GB memory. This camera came with 8× zoom with constant f/2.0 lens and is the main subject of chapter 3 in this work.

The common critique of the camera was the low resolution and poor quality of the photographs in comparison to traditional contemporary digital cameras. In July 2014, Lytro has released a second generation camera to address these issues and move from hobbyists more to the professional photographers, the *Lytro Illum camera*.

Figure 37. Lytro Illum camera, press release photo.

The price tag for Lytro Illum camera (on Figure 37) is $1,599. It also features 8× zoom with constant f/2.0. Unfortunately at the time of writing this thesis, the camera was not yet released and further technical details are not available.

2.4. RayTrix

RayTrix[25] is a company based in Germany, founded in 2008. They released their first camera in 2010, making it the first commercial light field camera on the market, called R11.

[25] http://www.raytrix.de/

However, RayTrix targets industrial segment in both features and prices. The release price of R11 camera (on Figure 38) was €30,000[26] and comes with various lens mounts (Nikon/F-mount, M58, Canon), up to 10 FPS video output over GigE/CameraLink, and SDK for developers. The R11 model has 40,000 microlenses.

Figure 38. Raytrix R11 camera, press release photo.

Over the time new models were introduced: R5, an entry-level camera with lower resolution but higher speed video and C-mount (€5,000[27]), and R29 with high resolution. They also offer modifying an existing customer's camera to become a light field one (RX).

The company does custom microlens arrays design and ships monochrome, colour and near infra-red versions of the cameras. Their cameras do not do any light field processing itself, the processing is offloaded typically to a standalone, high-performant computer.

The typical applications RayTrix is targeting is machine vision, surface and quality inspection, plant research, microscopy etc., most prominently 3D particle tracking.

2.5. Light field in other applications

Living photography and 3D reconstructions are not the only applications of light field. Other popular areas being researched include microscopy and light field displays.

[26] http://www.dradio.de/dlf/sendungen/forschak/1132822/
[27] ibid.

Again, Marc Levoy with his team at Stanford University are pioneering the area of light field microscopy. The microscopy has its own characteristic set of constraints. For example, microscope optics, being telecentric, produces only orthographic views, while light field allows to render new, perspective views of specimens. Being able to refocus or extended the depth of field from single capture makes it easy to inspect moving or light-sensitive objects as well [23]. Interested readers can find more information on Stanford website at http://graphics.stanford.edu/projects/lfmicroscope/.

As for light field displays, the break-through product is yet to come. The research is exploring several directions: NVIDIA attached microlens array to small displays in binocular configuration[28], researchers from the Institute for Creative Technologies at the University of Southern California are using high frame rate projector and rotating mirror to create 360° 3D scene with proper occlusions, while Camera Culture Group at Massachusetts Institute of Technology are using their compressive light field technology with a projector to build a 3D screen of larger dimensions.[29] Pamplona et al. suggested using light field displays to compensate for visual aberrations.[30]

Other interesting application that comes to my mind would be using a single strip of microlenses on 2D sensors in flatbed scanners. Not only that could compensate for rough surfaces or focus on individual layers of transparent materials like films, but it also could recognize and record protective patterns like those used on banknotes.

The biggest problem of getting light field technology into hands of students and researchers is the availability of microlens arrays. They are either too small (for example, Edmund Optics[31] offers arrays up to 1×1 cm only) or extremely expensive (or both, the small ones from Edmund Optics are for $550). Mats Wernersson published a way how one can make microlens array of acceptable performance themself, but it is quite a non-trivial task requiring equipment not everybody has access to. [32]

[28] https://research.nvidia.com/publication/near-eye-light-field-displays
[29] http://web.media.mit.edu/~gordonw/CompressiveLightFieldProjector/
[30] http://tailoreddisplays.com/
[31] http://www.edmundoptics.com/
[32] http://cameramaker.se/microlenses.htm

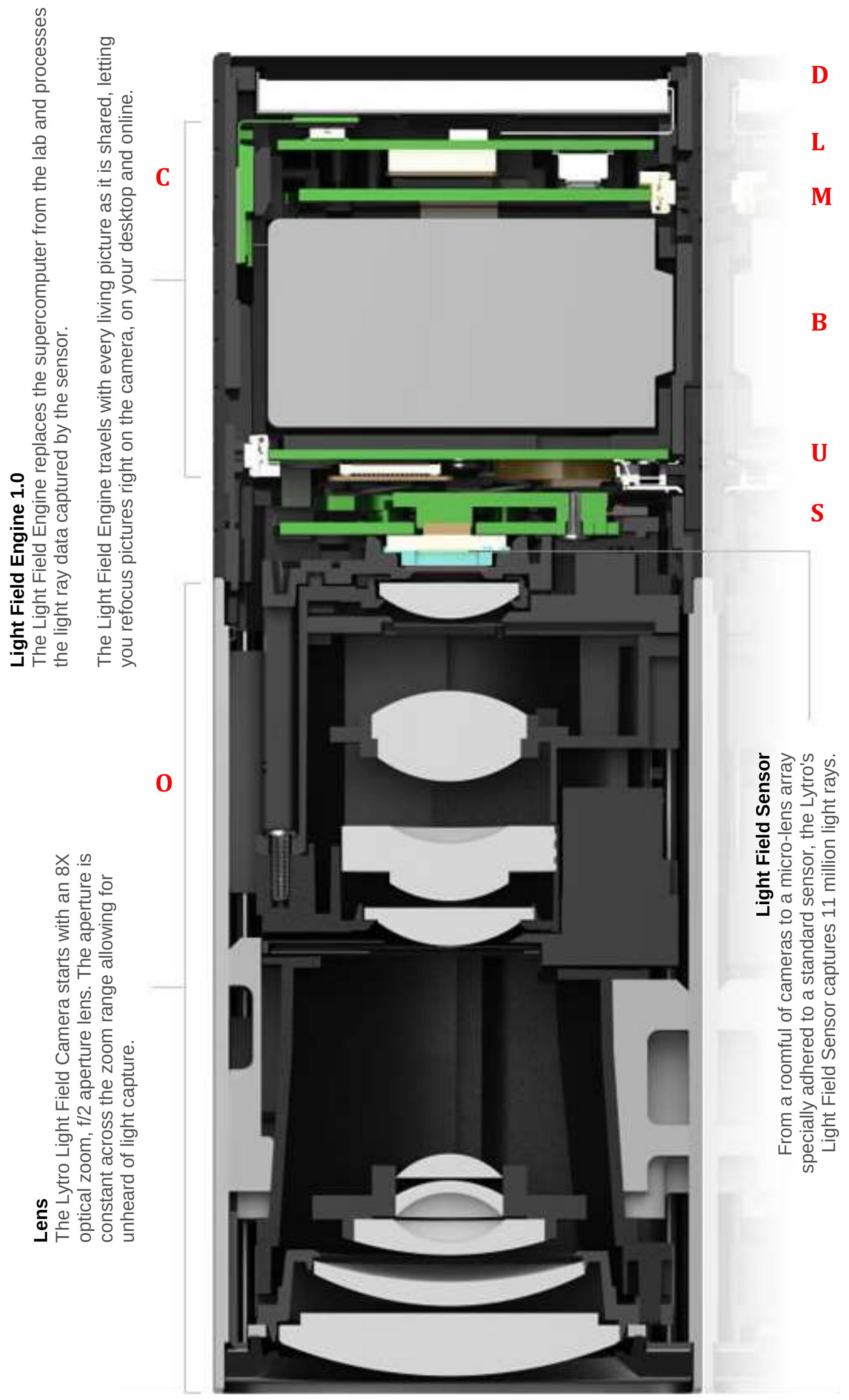

Light Field Engine 1.0
The Light Field Engine replaces the supercomputer from the lab and processes the light ray data captured by the sensor.
The Light Field Engine travels with every living picture as it is shared, letting you refocus pictures right on the camera, on your desktop and online.
C
Lens
The Lytro Light Field Camera starts with an 8X optical zoom, f/2 aperture lens. The aperture is constant across the zoom range allowing for unheard of light capture.
O
D
L
M
B
U
S
Light Field Sensor
From a roomful of cameras to a micro-lens array specially adhered to a standard sensor, the Lytro's Light Field Sensor captures 11 million light rays.

Figure 39. Inside cut through the first generation Lytro camera as published and described on Lytro's website [24]. Annotation mine.

All information in this chapter is based purely on my research of the camera, accompanying software and its behaviour. Lytro has neither supported nor confirmed any of the information herein presented. Use at your own risk.

Special notation is introduced to denote various sources:

SW	for software analysis
FW	for firmware analysis
FCC	for Federal Communication Commission materials
MET	for metadata produced by camera
PAT	for patents
3RD	for 3rd party source (manufacturer of the component in question)

3.1. Inside the Camera

The first generation Lytro camera's codename is *Firefly*SW,[33] and shipped in February 2012. The official model number is A1 as marked on the hardware.[34] It is a small, 41×41×112 mm camera with 8 GB or 16 GB of internal storage and couple of features not yet common on classical consumer cameras like wireless connectivity, touch screen and of course the microlens array.

3.1.1. Hardware

Having a wireless capabilities it had to be approved by Office of Engineering and Technology at Federal Communication Commission[35], which assigned Lytro a grantee code of *ZMQ*.FCC The commission publishes its measurements as well as some of the documents, which also includes photos of the product teardown.

Individual elements of the camera annotated on Figure 39 are described below. Photos with blue background come from the FCC exhibit, the ones with wooden background from my archive.

S **Sensor Board** (as of revision A6)
The sensor board contains a CMOS sensor with microlens array. The microlenses are arranged hexagonally, with rows being the major axis. All microlenses are of the same focal length, with the pitch of 13.89 μm and placed at 25 μm in front of the sensor.MET

[33] The codename for Lytro Illum camera is *Blitzen*.SW
[34] The model number of Lytro Illum camera is B5.FCC
[35] http://transition.fcc.gov/oet/

The sensor is Aptina MT9F002 14.4 Mpx 1/2.3” sensor (effective imaging area 6.14 × 4.6 mm) with 1.4 µm pixel size.[MET,3RD,36] The output frame size is, however, cropped to 3,280 × 3,280 pixels which gives about 10.7 Mpx at 12-bit resolution. There is a standard Bayer colour filter array (CFA) over the sensor to capture the colours with R,GR:GB,B pattern, blue being the top left pixel.[MET]

Finally, a 3-axis accelerometer is on board.[MET]

U **USB Board** (as of revision A6)

The USB board contains a Micro USB female connector at the bottom, the shutter button at the top and a piezzo buzzer, not enabled at the time of writing.

B **Battery**

The battery model is DC-A950 by FORMOSA, 3.7V⎓ 2100mAh 7.77Wh Li-ion. The calibration measurements are:[MET]

Working current:	-0.2670 A
Working voltage:	3.67 V
Power Consumption:	-990.8 mW
Charge current:	1.0210 A
Temperature:	27.4 C

M **Main Board** (as of revision A6)[FCC]

The main processing board of the camera. One the front side:

SAMSUNG NAND FLASH memory (8/16 GB)

On the back side:

Zoran‘s Camera On A Chip 32-bit RISC digital image processor (ZORAN ZR364246BGLG)

SK Hynix SDRAM memory

Temperature sensor

L **LCD Board** (as of revision A6)

The LCD board hosts Marvell‘s Avastar 88W8787A16 SoC[FCC] offering WiFi 802.11a/g/n, Bluetooth 3.0+HS, and FM radio with RDS and transmit capability on the back side.[3RD] At the time of writing, only WiFi functionality has been enabled.

36 http://www.aptina.com/products/image_sensors/

D Display

The display is 1.52” back-lit LCD with resolution of 128×128 pixels. There is a touch circuit on the back side of the display.

C Cap Slider

The zoom slider on top of the camera consists of 5 capacitive sensors.

O Lens

The manufacturer of lens is not known. It has 8 elements in 5 groups and focal length equivalent to 43—344 mm with constant f/2.0.[37] There is also a second temperature sensor on the lens.[MET]

3.2. File Formats

Lytro uses its own proprietary file format for their light field pictures. Its structure is divided into separate components laid one after other. The format of each component is described in Table 1. A software library that can parse and manipulate the light field picture format is part of this thesis.

```
89 L  F  _  0D 0A 1A 0A  VE VE VE VE  CL CL CL CL  (header, version, length)
s  h  a  1  -  00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 (name)
__ __ __ __ __ __ __ __ __ __ __ __ 00 00 00 00 (data, padding)
```

offset	length	notes
0	8	Fixed header (magic number), the fourth byte determines the component type, which can be one of these: **P** (package) always the first component in the file **M** (metadata) always the second component in the file and always the only one **C** (component) any other component
8	4	Version of the file format[SW], big endian integer. At the time of this work all Lytro files have 00 00 00 01 in the first (i.e. LFP) component. Other components must have 00 00 00 00.[SW]
12	4	Length of data in the component, big endian integer. This value can be zero, in which case this is the end of the component and the next

[37] https://www.lytro.com/camera/specs/gen1/

		component follows immediately. Currently the LFP component has always zero length.
16	80	Name of the component. The name can have up to 80 bytes, shorter names are padded with zeroes to the total length of 80. Several components can have the same name.SW
97	length	The actual data of the component.
*		Every component is padded with zeros so that the total length of the component (i.e. header + version + data length + name + data + padding) is multiple of 16. If the sum without padding is already a multiple of 16, then no padding is added.

Table 1. Light field component file format

The files do not have any special opening or closing and the file extension is **.LFP**.

When a Lytro camera is connected to the computer, the *Lytro Desktop* software is started and imports pictures from camera into the computer. During this process, several files per picture are created.

3.2.1. Raw pictures (raw.lfp)

Raw pictures correspond to RAW files in traditional cameras. They contain raw uncompressed sensor readings and need to be further processed.

The package metadata of raw pictures contain the metadata version, references to all the below components and whether the picture was marked as favourite, as well as whether the image is a dark or modulation frame (only calibration images do have these set).

Raw Sensor Data component

The Lytro A01 camera generates 3,280px × 3,280px × 12-bit data. To save space, the pixel values are packed together instead of padded, so there are 2 pixels (24 bits) stored in 3 bytes. Hence the length of this component in this case is always 16,137,600 bytes.

For example, if you have one white pixel (`FFF`) followed by a black one (`000`), there will be `FF F0 00` in raw data. Also, the values are stored in big endian order. So `12 34 56` need to be decoded into `03 12 06 45`. The first pixel is the top left one, continuing in rows.

The resulting image is grayscale with Bayer colour filter over it, so it needs to be demosaiced to obtain the colour information.

Frame Metadata component

Frame metadata contain all information needed to reconstruct the image. That includes the way how the raw sensor data component is encoded (so that data from other cameras/applications can be processed), readings from sensors (time in Zulu zone, temperatures of main board and lens, accelerometer, camera orientation), hardware configuration (parameters of sensor, microlenses, lens, camera maker and model), photo parameters (shutter, ISO, zoom, creative mode) and the firmware version.

As all metadata, this component is in the JSON text format.

Private Metadata component

Imaging sensor serial number and camera serial number is the only information in the private metadata component.

As all metadata, this component is in the JSON text format.

3.2.2. Prerendered pictures (stack.lfp, stacklq.lfp)

Since rendering the raw pictures is computationally very expensive and the files are large to share, the software generates files with some predefined views of the light field. It picks up to 12 depths and, if instructed, parallax or software filters and renders a stack of images that are small and easy to display.

The package metadata contain the metadata version, reference to the lookup table and references to all the prerendered images with a depth or parallax position they represent, so that the viewers can just show the correct image. The only other metadata included is whether the image was marked as favourite (and dark/modulation, see above), specially, there is no information about when the picture was taken, the caption that it was given, its ISO, shutter speed etc. in these files.

One or more prerendered images

In the case of low quality (stacklq.lfp) files, the prerendered images are stored as simple JFIF images, focused at different depths or with perspective shift. They are of 330×330 pixels resolution.

The higher quality variation (stack.lfp) has the images in 1,080×1,080 pixels resolution encoded into a H.264 Annex B stream.[38]

Depth Lookup Table component

In addition to the prerendered images, the files also contain a depth lookup table, which is an array of doubles representing the depths at which the picture should be refocused if user clicks at the corresponding position. The first depth is the top left area, continuing in rows.

[38] http://www.itu.int/rec/T-REC-H.264-201304-I/en

3.2.3. Depth maps (dm.lfp)

The last file that is generated is a separate depth map, with two components only, depth lookup table and depth confidence table.

Depth Lookup Table component

The depth lookup table is the same component as in the case of prerendered pictures.

Depth Confidence Table component

The depth confidence table has the same structure as depth lookup table, but the double values express the confidence ranging from 0 (not confident) to 1 (confident) of the depth value located at the same place in the lookup table. The depth values are computed from the light field so the accuracy varies.

3.2.4. Calibration files (data.C.#, *.calib)

When camera is connected to the computer for the first time, all the calibration data are backuped to the computer. They are packed in couple of `data.C.#` packages acting basically as containers for other files.

The package metadata contains list of file names present in the package and references to them, with an optional information which package file contains additional files. Then, the files themselves are the individual components of the package.

On Windows, these files can be found at `%LOCALAPPDATA%\Lytro\cameras`. On Mac, they are in the package at `Lytro.lytrolib\cameras`.

Calibration data for H.264 compression algorithm are stored in groups of bitmaps ibidem in `*.calib` files.

3.3. On the Air

Starting with camera firmware release v1.2, the on-board Wi-Fi is enabled. The connectivity is provided through Marvell's Avastar 88W8787A16 chip with MAC address in the range of Tayio Yuden Co., Ltd. (00:22:58). A software library that can communicate with the camera is part of this thesis.

The wireless communication must be explicitly enabled by user on the camera itself. This causes a network with SSID `lytro.camera.###` (where ### are the last three numbers of its serial number) to be broadcasted. The network is WPA2-CCMP protected with fixed keyphrase consisting of 8 arabic digits. A new keyphrase is generated when the camera is soft reseted, either manually from settings menu or when it hangs and resets itself.

3.3.1. Available services

Camera responds to ping messages. Wireless communication is power demanding and is turned off if not actively used for 5 minutes (even if a client is connected and the camera itself is being used).

Available UDP services

DHCP server, currently supporting one client at a time only (port 67).

`10.100.1.1`	is IP address of the camera (gateway),
`10.100.1.100`	receives the connecting client,
`255.255.255.0`	is the subnet mask.

DNS-Based Service Discovery, compatible with Multicast DNS (port 5353).

This system uses PTR requests to discover services on the network. Also, the service types and instances are broadcasted at start-up. The Lytro camera uses a `_lytro._tcp` service in the `local` domain. The DNS response/broadcast contains these answers:

```
PTR   _services._dns-sd._udp.local
      _lytro._tcp.local
      (service type announcement)

PTR   _lytro._tcp._local
      lytro-A#########._lytro._tcp.local
      (service instance announcement)

SRV   lytro-A#########._lytro._tcp.local
      lytro-A#########.local:5678
      (service endpoint)

TXT   lytro-A#########._lytro._tcp.local
      (empty)
      (named attributes)

A     lytro-A#########.local
      10.100.1.1
      (camera IP)
```

```
NSEC  lytro-A#########.local
      lytro-A#########.local
      (no more entries)
```

Open TCP ports

5677 (callback messages)

Once a client connects to this port, camera automatically starts sending various events back. Data are UTF-8 encoded strings, each event on its line. Individual messages and their parameters are described in chapter 3.3.1.

5678 and 5679 (Lytro service)

These ports use the same protocol, the mobile application uses port 5678 for control requests and port 5679 for downloading pictures. Communication is based on binary request-response pairs. Individual commands and their payload format are discussed in chapter 3.3.3.

3.3.2. Callback messages

Callback messages are sent automatically as soon as a client connects to the camera's port 5677. Data are UTF-8 encoded strings, each event on its line, ending with `CR LF` and a null character. The callback name is enclosed in square brackets, followed by parameter(s) separated by space.

```
[CallbackName] param1 param2 ...\r\n\0
```

HeartbeatTick

Generated automatically approximately every 100 ms if no other callback occurs.

Syntax: `[HeartbeatTick] tick`

Parameters:

parameter	value	description
`tick`	`string`	always 'tick'

Remarks:

This message is used as a watchdog to detect whether camera is still connected and responding. Sufficient amount of other messages will prevent this message coming, so the watchdog should be reset on any message received.

Examples:

```
[HeartbeatTick] tick\r\n\0
```

SelfTimerTick

Occurs every second during self-timer count-down.

Syntax: `[SelfTimerTick] state`

Parameters:

parameter	**value**	**description**

`state`	`integer`
	-or-
	`string`
		number of seconds remaining to trigger the shutter
		-or-
		‘Canceled’ when the user cancels the timer

Remarks:

The `SelfTimerTick` sequence ends either with `ShutterPressed` callback or cancelled.

Examples:

```
[SelfTimerTick] 2\r\n\0
[SelfTimerTick] Canceled\r\n\0
```

ShutterPressed

Occurs immediately after shutter is triggered.

Syntax: `[ShutterPressed] CLICK`

Parameters:

parameter	**value**	**description**
`CLICK`	`string`	always ‘CLICK’

Remarks:

The shutter can be triggered either manually by taking a picture, or using the self-timer. This callback does not distinguish between the two.

Examples:

```
[ShutterPressed] CLICK\r\n\0
```

NewPictureAvailable

Occurs when a picture taken is rendered and becomes available for download.

Syntax: `[NewPictureAvailable] id`

Parameters:

parameter	value	description
`id`	`string`	the id of picture available

Remarks:

A picture is not ready immediately when it is taken, it needs to be processed by the camera engine first. The id can be used to download the picture from the camera.

Examples:

```
[NewPictureAvailable] sha1-12345678901234567890123456789012345678 90\r\n\0
```

LikedChanged

Occurs when a user marks or unmarks a picture as a favourite.

Syntax: `[LikedChanged] id liked`

Parameters:

parameter	value	description
`id`	`string`	the id of picture available
`liked`	`integer`	‘1’ if marked as favourite -or- ‘0’ if unmarked as favourite

Remarks:

A picture is marked/unmarked as favourite by tapping the star icon.

Examples:

```
[LikedChanged] sha1-1234567890123456789012345678901234567890
\r\n\0
```

PictureDeleted

Occurs when a picture is deleted from the camera.

Syntax: `[PictureDeleted] id`

Parameters:

parameter	**value**	**description**
`id`	`string`	the id of picture deleted

Examples:

```
[PictureDeleted] sha1-12345678901234567890123456789012345678
90\r\n\0
```

AllPicturesDeleted

Occurs when all pictures are deleted from the camera at once.

Syntax: `[AllPicturesDeleted] all deleted`

Parameters:

parameter	**value**	**description**
`all`	`string`	always ‘all’
`deleted`	`string`	always ‘deleted’

Remarks:

This message is sent by the delete all command in the settings menu, regardless of the actual number of pictures deleted. Deleting the last picture individually does not send this message.

Examples:

```
[AllPicturesDeleted] all deleted\r\n\0
```

ZoomLevelChanged

Occurs when camera zoom is changed.

Syntax: `[ZoomLevelChanged] zoom`

Parameters:

parameter	value	description
`zoom`	`float`	the zoom level, ranging '1.0' to '8.0'

Remarks:

> This message is sent continuously as user swipes a finger to change the zoom factor, except when the General Control Dock is shown. In that case, the message is sent only once when the dock gets hidden again.
>
> The current shooting mode determines the possible range of zoom level. In standard mode, the range is '1.0' to '5.4' (the display shows 5.5x). In the Creative Mode, the zoom range is extended up to '8.0'. Switching between Everyday Mode and Creative Mode effectively changes the current zoom level, however, this does not result in any message.

Examples:

```
[ZoomLevelChanged] 2.0\r\n\0
```

CreativeModeChanged

Occurs when the current shooting mode changes.

Syntax: `[CreativeModeChanged] status`

Parameters:

parameter	value	description
`status`	`integer`	'1' if the new mode is Creative Mode -or- '0' if the new mode is Everyday Mode

Examples:

```
[CreativeModeChanged] 1\r\n\0
```

ShutterSpeedChanged

Occurs when the shutter speed setting changes.

Syntax: `[ShutterSpeedChanged] seconds`

Parameters:

parameter	**value**	**description**
`seconds`	`float`	the number of seconds the shutter is open -or- ‘0.000000’ if the shutter speed is set to automatic

Remarks:

This message occurs when user changes the Shutter Speed under manual controls. The callback happens only at the moment the Shutter Speed setting is opened (with current settings) and closed (with new settings), i.e. not as user continuously changes the setting. Changes due to automatic setting do not result in callback.

The shutter speed can be set manually to 8, 4, 6.4, 5, 4, 3.2, 2.5, 2, 1.6, 1.25, 1, 1/1.25, 1/1.6, 1/2, 1/2.5, 1/3.2, 1/4, 1/5, 1/6.4, 1/8, 1/10, 1/12, 1/15, 1/20, 1/25, 1/30, 1/40, 1/50, 1/60, 1/80, 1/100, 1/125, 1/160, 1/200 and 1/250 seconds. However, note that the automatic setting is not limited to these values and when opening the Shutter Speed setting with an automatic value, it can be any arbitrary number.

Switching to automatic control resets the shutter speed setting to automatic and causes this message to be sent, as soon as the checkbox in the menu is unchecked. Switching to manual control keeps the settings automatic, so no message sent in this case.

Examples:

```
[ShutterSpeedChanged] 0.000000\r\n\0
[ShutterSpeedChanged] 0.008158\r\n\0
[ShutterSpeedChanged] 0.008000\r\n\0
```

IsoSensitivityChanged

Occurs when the ISO sensitivity setting changes.

Syntax: `[IsoSensitivityChanged] value`

Parameters:

parameter	**value**	**description**
`value`	`float`	the ISO setting divided by 50.0 -or- ‘0.000000’ if ISO sensitivity is set to automatic

Remarks:

This message occurs when user changes the ISO Sensitivity under manual controls. The callback happens only at the moment the ISO Sensitivity setting is opened (with current settings) and closed (with new settings), i.e. not as user continuously changes the setting. Changes due to automatic setting do not result in callback.

The ISO sensitivity can be set manually to 3200, 2500, 2000, 1600, 1250, 1000, 800, 640, 500, 400, 320, 250, 200, 160, 125, 100, 80 and 75 (corresponding to values 64.0 to 1.5). However, note that the automatic setting is not limited to these values and when opening the ISO Sensitivity setting with an automatic value, it can be any arbitrary number. Also note that the ISO Sensitivity of 75 is incorrectly reported as 80 on the display.

Switching to automatic control resets the ISO Sensitivity setting to automatic and causes this message to be sent, as soon as the checkbox in the menu is unchecked. Switching to manual control keeps the settings automatic, so no message in this case.

Examples:

```
[IsoSensitivityChanged] 0.000000\r\n\0
[IsoSensitivityChanged] 56.509144\r\n\0
[IsoSensitivityChanged] 64.000000\r\n\0
```

NeutralDensityFilterChanged

Occurs when the neutral density filter is turned on or off.

Syntax: `[NeutralDensityFilterChanged] status`

Parameters:

parameter	**value**	**description**
`status`	`integer`	‘1’ if the filter was turned on -or- ‘0’ if the filter was turned off

Remarks:

This message occurs when user changes the ND filter setting under manual controls. The callback happens only at the moment the user taps the setting on the display. Changes due to automatic setting do not result in callback (which includes switching to automatic control or setting either Shutter Speed or ISO Sensitivity to Auto).

The ND filter has 4 stops.[MET]

Examples:

```
[NeutralDensityFilterChanged] 1\r\n\0
```

Other messages

The firmware suggests the following messages can be also generated but are not enabled at the time of this work.[FW]

- `BatteryLevelUpdated`
- `ManualControlModeChanged`
- `PictureCapacityUpdated`
- `USBStateChanged`

3.3.3. Commands reference

The camera accepts connections on TCP ports 5678 and 5679. Both requests and responses contain 28 bytes of header, optionally followed by payload data (content).

`AF 55 AA FA` `LE LE LE LE` `FL FL FL FL` `CM CM`	(magic number, content/buffer length, flags, command)
`PA PA PA PA PA PA PA PA PA PA PA PA PA PA`	(parameters)
`__ __ __ __ __ __ __ __ __ __ __ __ __ __`	(optional payload)

All numbers are little endian.

- `length` is 32-bit integer, representing either the numbers of bytes of payload attached, or the number of bytes of the receiving buffer (i.e. maximum allowed payload length of the response), depending on `flags`.
- `flags` two LSB observed only:
  - `xxxx xxx0` `length` is length of the payload
  - `xxxx xxx1` `length` is length of the buffer, no payload in the request
  - `xxxx xx0x` message is request
  - `xxxx xx1x` message is response
- `command` is 16-bit integer and determines the action the camera will execute and format of parameters and payload. Responses preserve the command and parameter values of requests.
- Each command has different number of parameters, not necessarily aligned. Unused bytes are zero.
- When payload is present, the `length` value contains its length in bytes. Format of the payload is different for each command.

Follows description of the observed commands, parameters and payload formats. Names are guessed and empty payload details are of unknown meaning.

Load hardware info (C2 00 00)

This command loads basic information about the camera.

Load file (C2 00 01)

This command loads a file from the camera storage.

Parameters:

offset	size	type	contents
`0x00`	`1`	`byte`	load type (1 = file)

Request payload:

offset	type	contents
`0x0000`	`string`	path to the file, null terminated

Response payload:

offset	type	contents
`0x0000`	`string`	path to the file, null terminated (same as in the request)

In case the requested file is not found, the response contains no payload.

Download payload:

Contents of the file.

Load picture list (C2 00 02)

This command loads a list of pictures available on the camera.

Parameters:

offset	size	type	contents
`0x00`	`1`	`byte`	load type (2 = picture list)

Request payload:

None.

Response payload:

None.

Download payload:

offset	size	type	contents
`0x0000`	4	`int`	
…	…	…	
`0x0058`	4	`int`	
then for each picture (128 bytes)			
`0x00`	8	`string`	folder name postfix, right padded with zeroes
`0x08`	8	`string`	file name prefix, right padded with zeroes
`0x10`	4	`int`	folder number
`0x14`	4	`int`	file number
`0x18`	4		
`0x1C`	4		
`0x20`	4		
`0x24`	4		
`0x28`	4	`int`	liked, 1 if picture marked favourite, 0 otherwise
`0x2C`	4	`float`	last lambda (at which user focused image in camera)
`0x30`	48	`string`	picture id, right padded with zeroes
`0x60`	24	`string`	date picture taken, ISO 8601 format
`0x78`	4		
`0x7C`	4	`int`	binary encoded rotation, 6 = 270°, 3 = 180°, 8 = 90°, 1 = 0° (counter-clockwise)

Load picture (C2 00 05)

This command loads a picture from the camera storage.

Parameters:

offset	**size**	**type**	**contents**
`0x00`	`1`	`byte`	load type (5 = picture)

Request payload:

offset	**type**	**contents**
`0x0000`	`string`	picture id followed by *picture format* digit, null terminated

Response payload:

offset	**type**	**contents**
`0x0000`	`string`	picture id followed by *picture format* digit, null terminated (same as in request)

In case the requested picture is not found, the response contains no payload.

Download payload:

4 bytes for length (`int`) followed by that amount of bytes containing the picture data, depending on the value of *picture format* digit.

When *picture format = '0'*, data contains a single JPEG file. The camera shoots in RAW+JPEG configuration. This is the JPEG part of it with compressed, colour microlens image. Data is equivalent to downloading `I:\DCIM\###PHOTO\IMG_####.JPG` file.

When *picture format = '1'*, data contains a single RAW file. The camera shoots in RAW+JPEG configuration. This is the RAW part of it. Data is equivalent to downloading `I:\DCIM\###PHOTO\IMG_###.RAW` file, and to the Raw Sensor Data component in the `raw.lfp` file.

When picture format = '2', data contains a single TXT file with metadata about the picture, including debug metadata which are otherwise inaccessible. Data is equivalent to downloading `I:\DCIM\###PHOTO\IMG_###.TXT` file.

When *picture format = '3'*, data contains a single thumbnail image with dimensions of 128×128 pixels. It is raw data, 16 bits per pixel, 4:2:2 YUY2 format. Data is equivalent to downloading `I:\DCIM\###PHOTO\IMG_###.128` file.

When *picture format = '4'*, data contains prerendered JPEG files with dimensions of 320×1280 pixels, each containing 4 frames of 320×320 pixels at different lambda. Data is equivalent to downloading `I:\DCIM\###PHOTO\IMG_###.STK` file and is laid as follows:

offset	**size**	**type**	**contents**
`0x0000`	4	`int`	total size of the file, including this field
`0x0004`	4	`int`	
`0x0008`	4	`int`	
`0x000C`	4	`int`	total number of frames
`0x0010`	4	`int`	total number of files
`0x0014`	4	`int`	width of frame
`0x0018`	4	`int`	height of frame
then for each file			
`0x0000`	4	`int`	length of the file
`0x0004`	4		(file contents)

Picture data are not guaranteed to exist even for valid picture IDs. In that case, payload for all picture formats contains only 4 bytes for the length (= 0).

Load calibration data (C2 00 06)

This command loads the calibration data minimum (set of files).

Parameters:

offset	**size**	**type**	**contents**
`0x00`	1	`byte`	load type (6 = calibration data)

Request payload:

None.

Response payload:

None.

Download payload:

offset	size	type	contents
for each file in the set			
`0x0000`	4	`int`	length of the file
`0x0004`	32	`string`	path to the file (on camera), right padded with zeroes
then for each file in the set, in the same order			
`0x0000`			(file contents)

Load compressed raw picture (C2 00 07)

This command loads a picture in the `rawPackedJpegCompressed` representation.

Parameters:

offset	size	type	contents
`0x00`	1	`byte`	load type (7 = compressed raw picture)

Request payload:

offset	type	contents
`0x0000`	`string`	picture id, null terminated

Response payload:

offset	type	contents
`0x0000`	`string`	picture id followed, null terminated (same as in request)

In case the requested picture is not found, the response contains no payload.

Download payload:

offset	size	type	contents
`0x0000`	4	`int`	length of metadata
`0x0004`	4	`int`	(not quite length of data)
`0x0008`			metadata of specified length followed by the data (a JPEG file)

Download (C4 00)

This command retrieves the content loaded by a load command above.

Parameters:

offset	**size**	**type**	**contents**
`0x00`	`1`	`byte`	(0 or 1)
`0x01`	`4`	`int`	offset

Request payload:

None.

Response payload:

Loaded content (see above commands for the data format), starting at specified offset. If content length is smaller than suggested buffer size, the response header contains the actual returned length. If it is larger, only the requested amount will be specified. The offset parameter can be used to retrieve the rest.

Specifying offset larger than content length (or not loading any content ahead) will result in no payload in the response. You can get the total content length using the query command as described below.

Requesting a buffer size that the camera cannot allocate will cause it to halt. Current software uses 2 MB buffer size.^{SW}

Upload (C5 00)

This command writes content to the active target.

Parameters:

offset	**Size**	**type**	**contents**
`0x00`	`1`	`byte`	(0)
`0x01`	`4`	`int`	offset

Request payload:

The content to be stored on the camera, starting at specified offset. Sending more data that the camera can allocate will cause it to halt. Current software uses 2 MB chunks.^{SW}

Response payload:

None.

Query content length (C6 00 00)

This command returns the loaded content length.

Parameters:

offset	size	type	contents
`0x00`	1	`byte`	query type (0 = content length)

Request payload:

None.

Response payload:

offset	size	type	contents
`0x0000`	4	`int`	content length

If no content was loaded, the returned length is zero.

Query camera time (C6 00 03)

This command returns current camera time.

Parameters:

offset	size	type	contents
`0x00`	1	`byte`	query type (3 = camera time)

Request payload:

None.

Response payload:

offset	size	type	contents
`0x0000`	2	`short`	year
`0x0002`	2	`short`	month
`0x0004`	2	`short`	day
`0x0006`	2	`short`	hour
`0x0008`	2	`short`	minute
`0x000A`	2	`short`	second
`0x000C`	2	`short`	millisecond

Milliseconds are currently not reported (the value is zero).

Query battery level (C6 00 06)

This command returns current battery level (as percentage).

Parameters:

offset	size	type	contents
`0x00`	1	`byte`	query type (6 = battery level)

Request payload:

None.

Response payload:

offset	size	type	contents
`0x0000`	4	`float`	battery level (percentage)

Take a picture (C0 00 00)

This command triggers the camera shutter.

Parameters:

offset	size	type	contents
`0x00`	1	`byte`	set type (0 = shutter)

Request payload:

None.

Response payload:

None.

(C0 00 02)

This command finalizes a firmware update. It might mean *end of upload* or *apply firmware update*.

Request payload:

None.

Response payload:

None.

Set camera time (C0 00 04)

This command sets current camera time.

Parameters:

offset	size	type	contents
`0x00`	1	`byte`	set type (4 = camera time)

Request payload:

offset	size	type	contents
`0x0000`	2	`short`	year
`0x0002`	2	`short`	month
`0x0004`	2	`short`	day
`0x0006`	2	`short`	hour
`0x0008`	2	`short`	minute
`0x000A`	2	`short`	second
`0x000C`	2	`short`	millisecond

Response payload:

New camera time, same format as in the request. Using this command can be logged to `I:\RTCERROR.LOG` file on the camera.

Milliseconds are currently not reported (the value is zero), but the written value is used.^{FW}

Write firmware (C1 00)

This command initiates a firmware upload. In the command terminology, it selects firmware update as the active target. To be followed by the Upload (C5 00) command.

Parameters:

offset	Size	type	contents
`0x00`	1	`byte`	(0)
`0x01`	4	`int`	firmware length

Request payload:

None.

Response payload:

None.

3.4. Lytro Compatible Library

A Lytro compatible library is part of this thesis, providing access to the functionality described above. The overall structure of the library features can be seen on Figure 40.

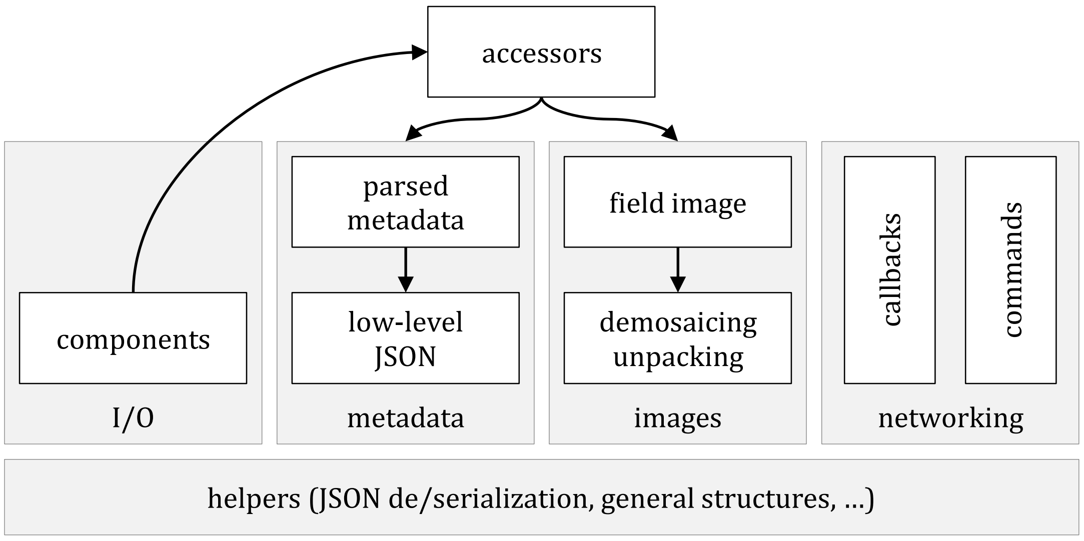


Figure 40. Structure of library features

The library was created as a .NET Portable Class Library, allowing it to be referenced by .NET Framework 4.0, Windows Store and Universal applications, Silverlight 4, Windows Phone 7, Xbox 360 and higher projects (Profile 1)[39]. This is rather limiting profile without support for asynchronous code, file paths, network sockets and other features that might be useful, so for these cases a full desktop library project is available, from which a subset of files is compiled as the portable library.

For detailed class reference, see documentation on the accompanying media. Samples in this chapter contain little or no error checking for clarity.

3.4.1. Working with files

The core classes for working with the light field picture files are `LightFieldComponent` and `LightFieldPackage` deriving from it (their diagram is on Figure 41). We have seen in chapter 3.2 File Formats that the files are collections of named components, the first one of type **P** and being empty. The collection of components is hold by `LightFieldPackage`, which itself represents the **P** component and is the first one in the collection. To ensure maximum flexibility at this level and due to lack of the official

[39] The profile's metadata must be modified to allow targeting all these platforms together in Visual Studio 2013 and above, refer to the accompanying media for instructions.

documentation, its position and count is not enforced and developers are responsible for keeping the **P** component part of the component collection.

Figure 41. Classes for working with files

When the LightFieldPackage reads a stream of components (usually a file stream), their names are stored in a dictionary for fast lookup, and those of type **M** (metadata) are noted separately. Note that the components do not need to have unique name and there is no specification available limiting metadata components to single instance per package, so users must be able to consume a list of components for given name or type.

The typical scenario called *splitting* that extracts all components into separate files can be carried out as in Listing 1.

```
string path = "IMG_0000.lfp";
LightFieldPackage package = new LightFieldPackage(File.OpenRead(path));

for (int i = 1; i < package.ComponentsCount; i++)
    File.WriteAllBytes(path + "." + i, package.Components[i].Data);
```

Listing 1. Splitting light field packages

The ComponentCollection of LightFieldPackage is accessible through its Components property and is read-write. The components can then be serialized again to a stream using the WriteTo method.

The non-portable version can create LightFieldPackages from files obtained from camera storage using the static `FromCameraFiles` method, which allows downloading light field pictures from Lytro camera without physically attaching the camera to the computer. The communicator application utilizes this function.

3.4.2. Working with metadata

There are two levels of abstraction for metadata available in the library. The low-level structures directly reflecting metadata as written in the components are nested in the static `Json` class. Users can directly parse the metadata onto these structures or use them to generate JSON strings. A sample code checking popularity of a picture is shown in Listing 2.

```
LightFieldComponent metadata = package.GetMetadata().First();

Json.Master packageMetadata = new Json.Master();
packageMetadata.LoadFromJson(metadata.GetDataAsString());

// assuming the property is always there and the first one
bool isFavorite = (packageMetadata.Picture.ViewArray[0].VendorContent
                   as Json.LytroStars).Starred;
```

Listing 2. Accessing low-level metadata

For easier access, metadata manipulation and better type safety, a higher level of abstraction for dealing with common metadata is available by means of `PictureMetadata`, `FrameMetadata`, `FilesMetadata` and `DebugMetadata` classes. The same information as above can be obtained using couple of lines of Listing 3.

```
PictureMetadata pictureMetadata =
                                 new PictureMetadata(packageMetadata.Picture);

isFavorite = pictureMetadata.IsStarred;
```

Listing 3. Accessing high-level metadata

The higher-level classes use the lower JSON classes as backing storage.

3.4.3. Working with images

As described in chapter 3.2.1 Raw pictures (raw.lfp), the raw pictures are bit packaged, with Bayern filter superposed on it. The library offers classes to decode and interpret the images step by step, summarized in Figure 42.

All image representations in the library derive from `ISampled2D<T>` or `IContinuous2D<T>` providing access to the underlying 2D data through indexer. The basic steps need to be done with the raw data are *unpack* ➞ *demodulate* ➞ *interpolate*.

Figure 42. Classes for working with images

The class structure reflects the process closely. The `LightFieldComponent` containing raw data is passed to the `RawImage`, which can be passed to the `DemosaicedImage`, which in turn can be used to initialize `InterpolatedImage`.

To improve performance when only subset of pixels is needed, the demosaicing is performed lazily. The library implements high-quality linear interpolation algorithm by Malvar et al. [25], but can be recompiled to traditional averaging by undeclaring the `USEFILTER` preprocessor variable. If lazy evaluation is not desired, developers can use the `Demosaic` method to perform complete demosaicing in advance. Finally, interpolation uses standard bi-linear approach.

Since working with the images is memory intensive (128 bits per pixel are used) and the scenario to get various stages progressively is common, the library features the high-level `FieldImage` class to encompass the process, instantiating the chain of classes as required. Additionally, it provides access to the XY and UV images through the `GetSubapertureImage` and `GetMicrolensImage` methods, respectively.

The non-portable version then allows for converting the images into standard BitmapSources to be displayed in UI and used in common image processing.

3.4.4. Accessors

So far, the developer needs to process an arbitrary LightFieldPackage, get the metadata component, parse it and find the other components containing the data of interest. Classes deriving from the abstract PackageAccessor are intended to help with this technical and error-prone routine for packages with known structure, to get the developer directly to the data he needs. For example for the raw image files, one can use the RawFieldAccessor as suggested in Listing 4 to access common metadata and the central sub-aperture image.

```
string path = "IMG_0001.lfp";
LightFieldPackage package = new LightFieldPackage(File.OpenRead(path));

RawFieldAccessor raw = package.AccessRaw();
bool isFavorite = raw.GetPictureMetadata().IsStarred;

XYImage central = raw.GetFieldImage().GetSubapertureImage(0, 0);
BitmapSource centralBitmap = central.ToBitmapSource();
```

Listing 4. Using the accessors

3.4.5. Communicating with camera

We have learned in chapter 3.3 On the Air two interactions the camera allows using wireless networking – callback messages and commanding. The callbacks are exposed through the LytroCallbackSink class, while for bi-directional communication and commands, the LytroNetClientPortableBase would be the starting point.

However, the portable class library does not support managing network connections and socket communication, which complicates design of these features. Therefore, for the portable case, developers need to establish the connection with the camera on their own and pass the network stream to the library. This is easy for the callbacks which are just streams of events, but the full Lytro networking client must ensure minimum robustness and integrity of the connection.

Receiving callbacks

The LytroCallbackSink class is designed for maximum performance, keeping in mind that once connected, the camera continuously sends messages at 100 ms rate (in the idle case), and optimized for the currently known callbacks in terms of buffer sizes, yet still adaptive for new challenges.

The class offers strongly typed events for each of the callbacks documented above, such as NewPictureAvailable or ZoomChanged. The developer is expected to subscribe to the callbacks he is interested in and start processing the callback stream (or the other way, in which case some callbacks might be missed), as in Listing 5.

```
LytroCallbackSink sink = new LytroCallbackSink();
sink.SelfTimerTick += (sender, c) =>
                  Console.WriteLine("Smile, {0} seconds to go!", c.Seconds);

sink.Process(stream); // connected network stream
```

Listing 5. Receiving callbacks in portable library

In the portable scenario, processing the stream is a synchronous call. In order to stop it, the stream must be closed or the processing stopped via StopProcessing call from another thread. Note that processing is stopped co-operatively, i.e. the callback stream must return from the read request in order for the process to be stopped. If waiting for that to happen is not desired, users can use StopProcessingAsync method instead and check the IsProcessing property for the status. The non-portable version then takes care of the connection and allows for asynchronous processing (Listing 6).

```
LytroCallbackSink sink = new
                      LytroCallbackSink(LytroCallbackSink.DefaultEndPoint);

sink.SelfTimerTick += (sender, c) =>
                  Console.WriteLine("Smile, {0} seconds to go!", c.Seconds);

// either
sink.Process(); // the sink connects to the camera itself
// or
sink.ProcessAsync(CancellationToken.None);  // asynchronous calls available
```

Listing 6. Receiving callbacks in non-portable library

The extra performance is gained by storing the callback handlers in a dictionary and most importantly by not parsing the callback parameters if there is no handler registered to receive it. There is a generic CallbackReceived event to record or pre-process all incoming callbacks and to handle callbacks that are not well-known. For complete description of the individual callbacks and usage remarks, see chapter 3.3.2 Callback messages.

Sending commands

The architecture of LytroNetClient is based on the principles of WebClient, built on the same layers of abstraction, so anyone familiar with the infrastructure for HTTP communication should be able to communicate with the camera with ease. Overview of the architecture is shown on Figure 43.

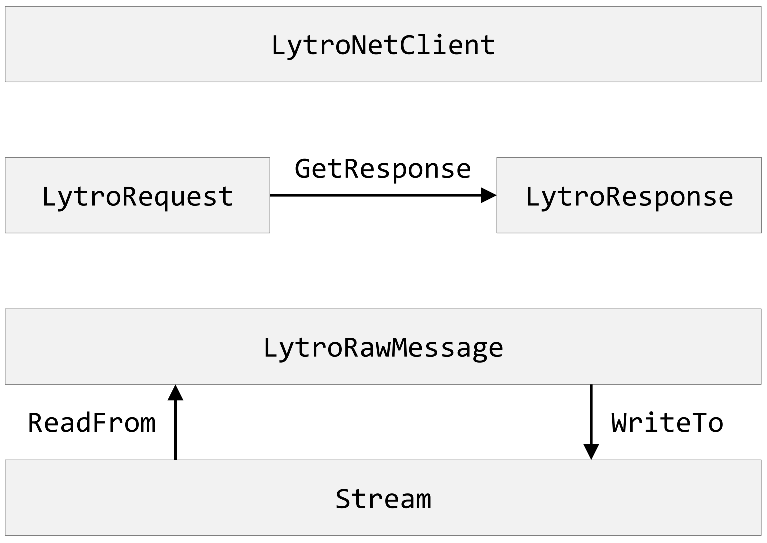


Figure 43. LytroNetClient architecture

At the top most level, there is the `LytroNetClient` class. As already noted, consumer of the portable library must supply the connection management. Listing 7 presents the most primitive implementation. A more polished version with connected/disconnected events is part of the non-portable version.

```
public class LytroNetClient : LytroNetClientPortableBase
{
    private Stream _stream;
    private TcpClient _client;

    protected override Stream GetStream()
    {
        if (_stream == null || !_stream.CanRead || !_stream.CanWrite)
        {
            _client = new TcpClient();
            _client.Connect("10.100.1.1", 5678);
            _stream = _client.GetStream();
        }

        return _stream;
    }

    protected override void OnException(Exception e, Stream stream)
    {
        if (_client != null)
            _client.Close();
    }
}
```

Listing 7. Simple LytroNetClient implementation

With the client, taking pictures is as easy as calling `new LytroNetClient().TakePicture()`. Most of the commands described in chapter 3.3.3 Commands reference are accessible through methods of the client, see Figure 44.

Figure 44. Classes for working wireless

The client also reports progress whenever data is being downloaded or uploaded, as Listing 8 shows for the case of downloading files.[40]

```
LytroNetClient lytro = new LytroNetClient();
lytro.DownloadBufferSize = 512; // only to demonstrate progress monitoring,
                                   this is a small file!

lytro.ProgressChanged += (sender, e) => Console.WriteLine(
                       "{0:P0} % completed",
                       (float)e.BytesTransferred / e.TotalBytesToTransfer);

byte[] data = lytro.DownloadFile("A:\\VCM.TXT");

File.WriteAllBytes("VCM.TXT", data); // save to disk
```

Listing 8. Downloading files with progress monitoring

Underneath, the `LytroNetClient` uses `LytroRequest` and `LytroResponses` classes (corresponding to `WebRequest` and `WebResponse`). When working at this layer, developers need to create the request and then send it using `GetResponse` method over the stream. However, in contrast to the HTTP protocol, the request must contain expected length of the response content, so that must be passed in as well. Check the client's source code for extensive use of requests and responses, a simple example of

[40] See Appendix B for list of files that can be found on the Lytro camera.

querying current battery level is pointed out in Listing 9. Again, the non-portable version offers an asynchronous `GetResponseAsync` call.

```
LytroRequest request = LytroRequest.Create(LytroCommand.QueryBatteryLevel);
LytroResponse response = request.GetResponse(stream, 4
                                        /* expected response payload */);

Debug.Assert(response.ContentLength == 4);

return BitConverter.ToSingle(response.Content, 0);
```

Listing 9. LytroRequest and LytroResponse example

At the lowest level, both responses and requests work with the `LytroRawMessage` class, which represents the raw data being sent over network, and enables two additional, advanced scenarios. First one is constructing and observing commands that weren't documented yet, and the other one is building the server side of the network service. The other noteworthy functionality of raw messaging are the static `TraceHeader` and `TraceData` events that can be subscribed to for diagnostic and logging purposes of the overall communication that is taking place.

3.5. Supplementary Software

Two desktop applications that build on top of the library are part of this thesis, the Lytro Compatible Viewer (Figure 45) and Lytro Compatible Communicator (Figure 46).

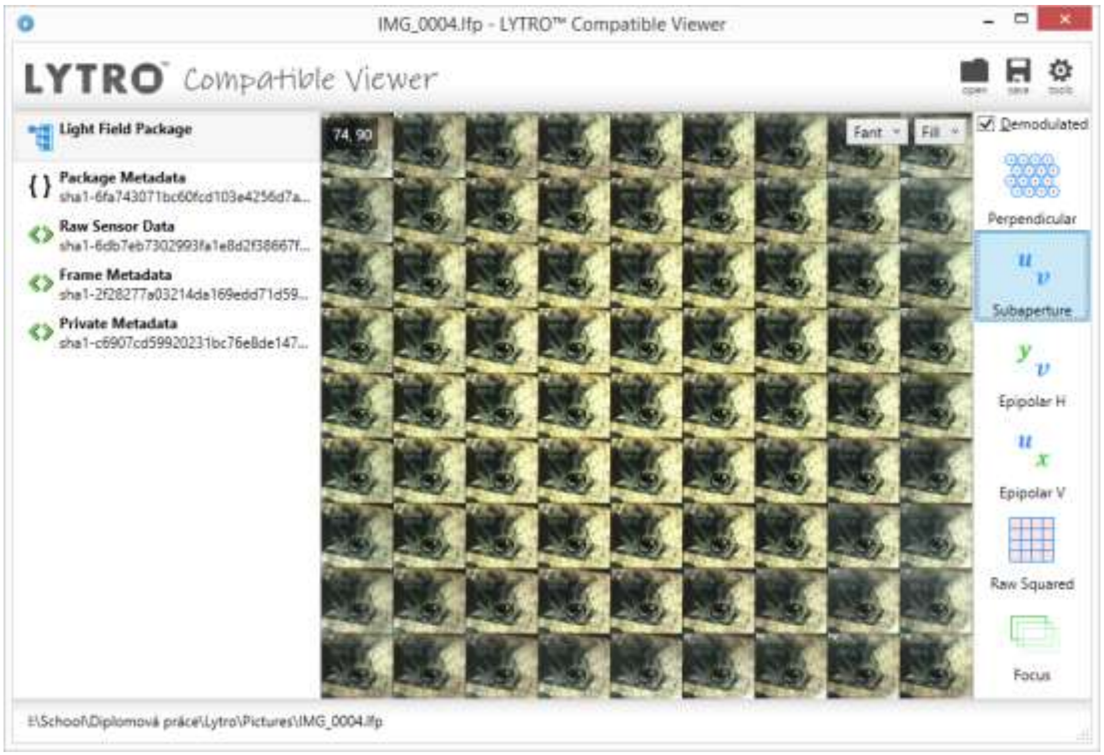
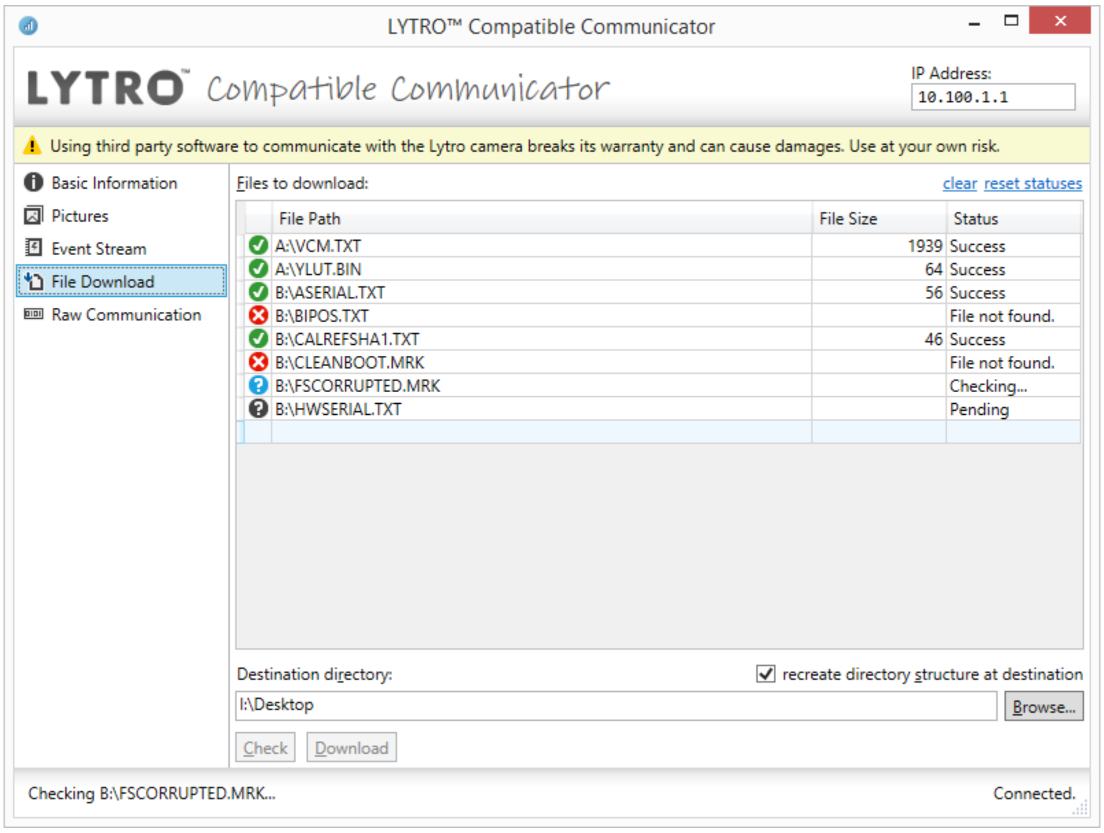

IMG_0004.lfp - LYTRO™ Compatible Viewer
LYTRO Compatible Viewer
open
save
tools
Light Field Package
Package Metadata
sha1-6fa743071bc60fcd103e4256d7a...
Raw Sensor Data
sha1-6db7eb7302993fa1e8d2f38667f...
Frame Metadata
sha1-2f28277a03214da169edd71d59...
Private Metadata
sha1-c6907cd59920231bc76e8de147...
74, 90
Fant
Fill
Demodulated
Perpendicular
Subaperture
Epipolar H
Epipolar V
Raw Squared
Focus
E:\School\Diplomová práce\Lytro\Pictures\IMG_0004.lfp

Figure 45. Lytro Compatible Viewer


LYTRO™ Compatible Communicator

LYTRO™ Compatible Communicator

IP Address: 10.100.1.1

Using third party software to communicate with the Lytro camera breaks its warranty and can cause damages. Use at your own risk.

Files to download:

	File Path	File Size	Status
✓	A:\VCM.TXT	1939	Success
✓	A:\YLUT.BIN	64	Success
✓	B:\ASERIAL.TXT	56	Success
✗	B:\BIPOS.TXT		File not found.
✓	B:\CALREFSHA1.TXT	46	Success
✗	B:\CLEANBOOT.MRK		File not found.
?	B:\FSCORRUPTED.MRK		Checking...
?	B:\HWSERIAL.TXT		Pending

Destination directory: I:\Desktop

[✓] recreate directory structure at destination

Checking B:\FSCORRUPTED.MRK...

Connected.

Figure 46. Lytro Compatible Communicator

The viewer can open and edit Lytro's light field pictures and data contained in them, manipulate the individual components, render different views of the light field etc. It also provides shell integration so users can see picture thumbnails in the file explorer.

The communicator utilizes most of the commands listed above, features downloading pictures from the camera without additional software, remotely triggering the shutter or raw communication terminal.

Unfortunately implementation details of these applications are greatly out of topic of this work, but readers are welcome to study the enclosed source code in case of doubts about the library usage. User manuals are available on the media.

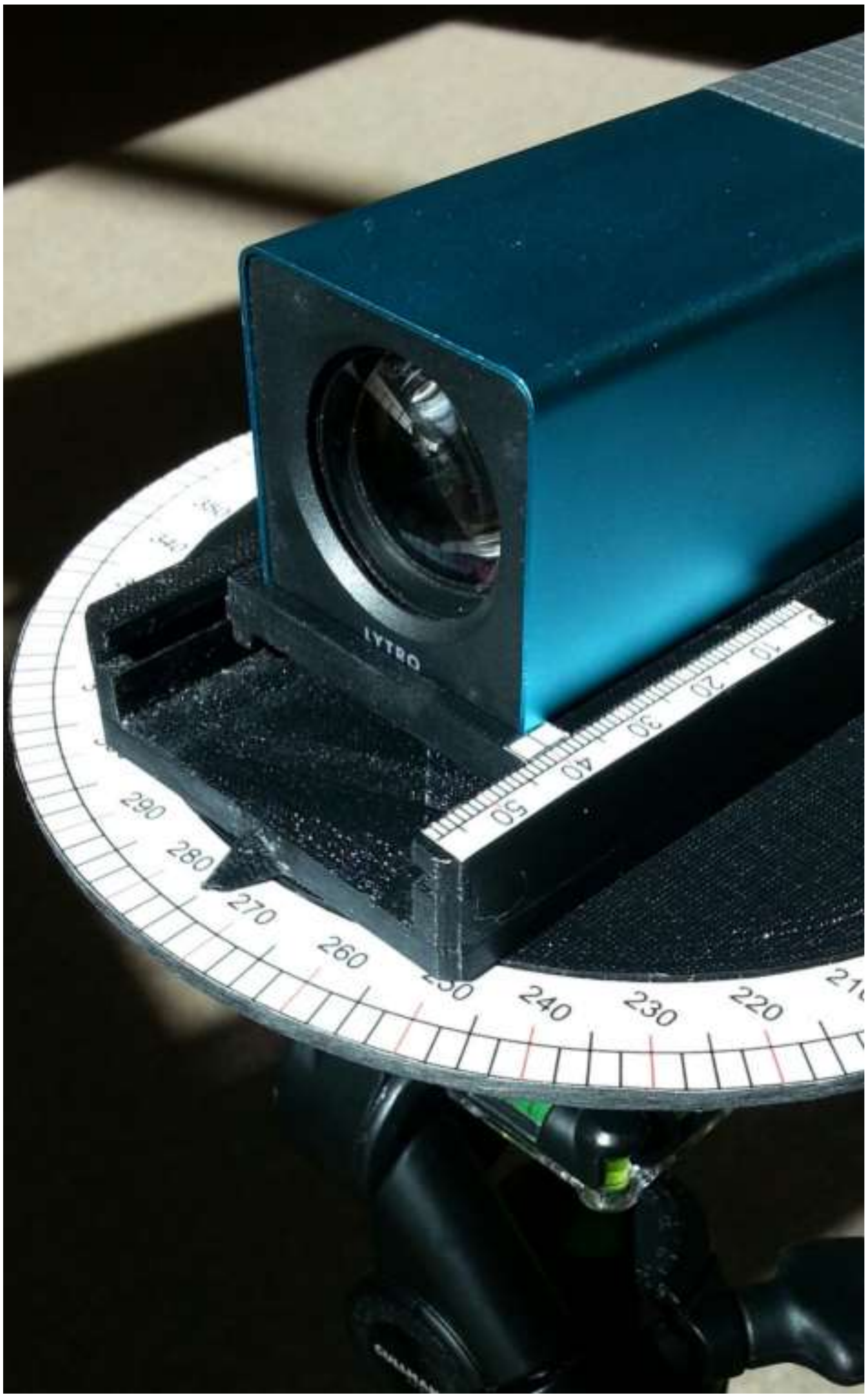


Figure 47. A Lytro camera mounted on panoramic tripod head designed and 3D printed at Johannes Kepler University, with permission. Source: [2]

While Lytro Desktop reveals the basics of light field features, only few image-processing algorithms are available in the field. For the panorama, i.e. merging slightly overlapping images, the trivial solution is taking individual images rendered at given focus planes and merging those using classical methods available for stitching 2D images. Institute of Computer Graphics at Johannes Kepler University in Linz[41] first published results using this method in 2012 [26].

However, stitching prerendered images does not preserve the light field itself, i.e. the directional information captured in the pictures. This method suffers from all the problems the classical image stitching has and introduces new artefacts e.g. during refocusing [2]. The same authors recently published paper on true light field merging applied to 360° panorama with constant roll and pitch angles of the camera [2], for which they designed and published a panoramic tripod mount (Figure 47).

Let us focus on how the light field behaves on *linear panoramas*, that is with the same constraints on roll and pitch angles, but with translating the camera along one axis.

4.1. Motivation

Imagine three points on the optical axis, at various distance from the main lens. From the theory in the first chapter, we know that the image on sensor will be similar to the one depicted on Figure 48.

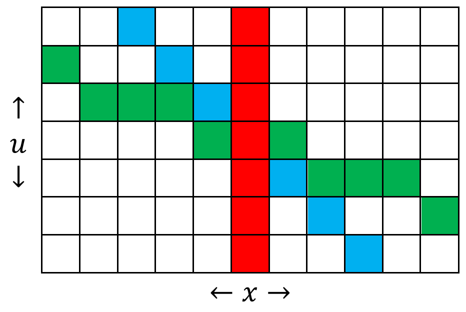


Figure 48. Three points, base UX view

Simulations show that if the sensor moves in the direction perpendicular to the optical axis (i.e. the distances between planes containing the points and the lens plane remain constant), a shifted images will be captured as shown on Figure 49.

[41] http://www.jku.at/cg/

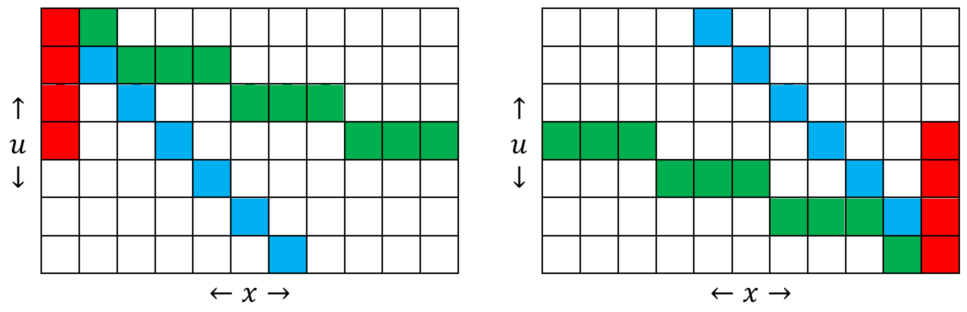


Figure 49. Three points, shifted UX view

Standard methods such as cross-correlation could then be used to register the images together, the desired result being Figure 50.

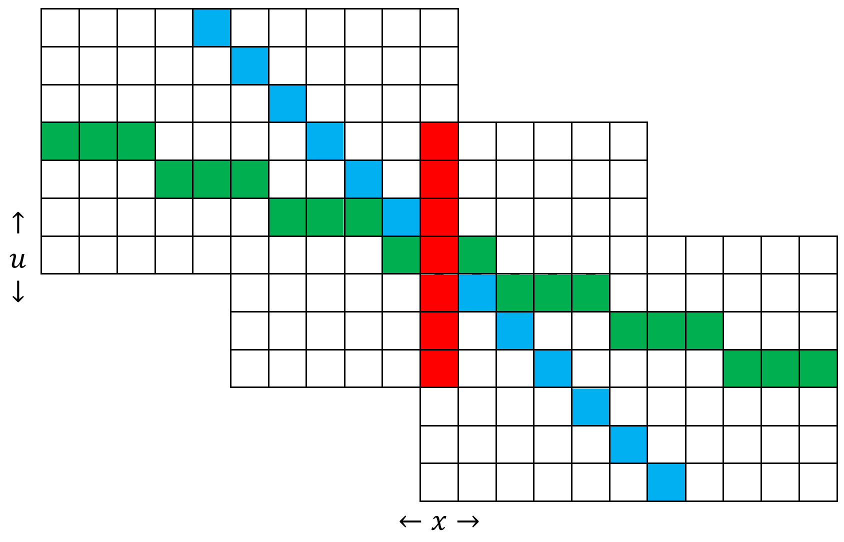


Figure 50. Three points, merged UX view

Couple of interesting things are apparent from the merged view. The pixels at top right and bottom left cannot be obtained when moving the sensor as described. They represent rays coming from directions that would require either bigger aperture and/or sensor, or camera rotation. On one side, this could help reducing the space that needs to be searched when registering images, on the other side it limits what features will the merged light field offer.

4.2. Derivation

We have shown in chapter 1.2.3 Direction sampling that individual points in the scene form lines with constant slope and in chapter 1.4.2 Sensor equations that we can treat the image under micro lens array as rasterization of that line. Although the equation for line was derived for a point on the optical axis only, it immediately follows that for off-axes points, the line will be offset as well (for illustration see Figure 51).

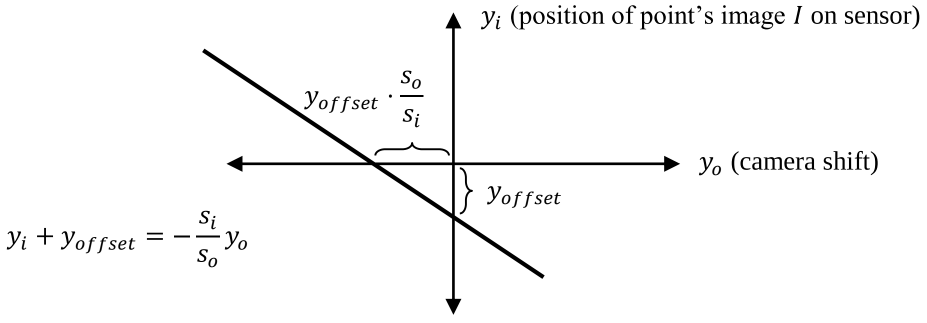

y_i (position of point's image I on sensor)
$y_{offset} \cdot \frac{s_o}{s_i}$
y_o (camera shift)
y_{offset}
$y_i + y_{offset} = -\frac{s_i}{s_o} y_o$

Figure 51. Position of single off-axis point on sensor from different camera locations

The fact that the UX view consists of overlapping lines whose relative position and slope does not depend on the camera translation in question makes registering the respective UX images a well-defined operation using translation transformation only.

Imagine rays coming from infinity through the main lens (Figure 52).

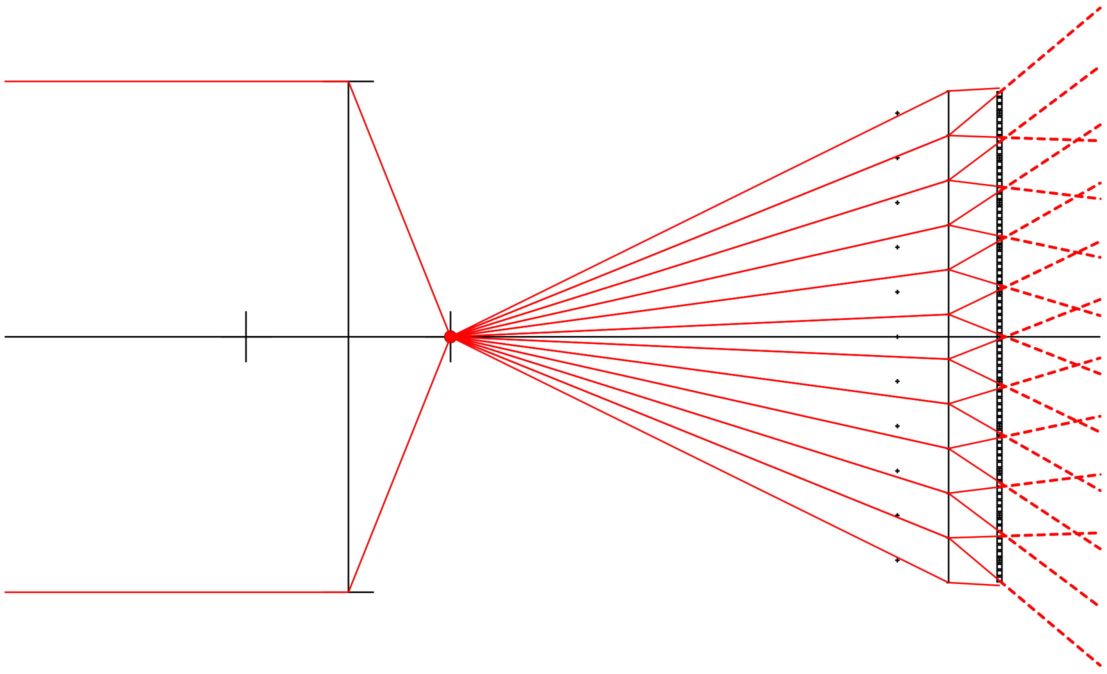


Figure 52. Rays from infinity

All rays coming from infinity meet at focal point of the lens. This point is then imaged through the microlenses and we already know this results in a line in the UX view. Regardless of how the camera is moved in direction perpendicular to the optical axis, the rays from infinity still meet at the same point. Indeed, in traditional photography, if camera is being moved e.g. horizontally, objects near the camera move faster in the image than more distant objects, and eventually the scene at infinity stays identical all the time.

Back to our single point example, rays coming from infinity form the same line, with the same slope and same position, regardless the camera movement. It follows that if we want to register such two images, these lines must come aligned. Hence, in case of camera translation, all images must lie on this *line of infinity*, leaving only one dimension to search. For given camera parameters, we even know the line in advance:

$$k = \tan \varphi = -\frac{s_i}{s_o} = -\frac{f_{mla}}{d - f_{lens}}$$

It is also easy to see from Figure 51 that we can map the translation of camera y_{offset} to the shift in images y_k along the line using Pythagorean theorem,

$$y_k = \sqrt{y_{offset}^2 \cdot \frac{s_o}{s_i} + y_{offset}^2} = y_{offset} \sqrt{\frac{d - f_{lens}}{f_{mla}} + 1}\,.$$

The above steps apply in 4D too, reducing the search space to one or two dimensions only, depending on whether we allow translations in horizontal or vertical directions only, or in both.

4.3. Limitations

For linear panoramic applications, the missing data restricts available rendering options of the complete scene.

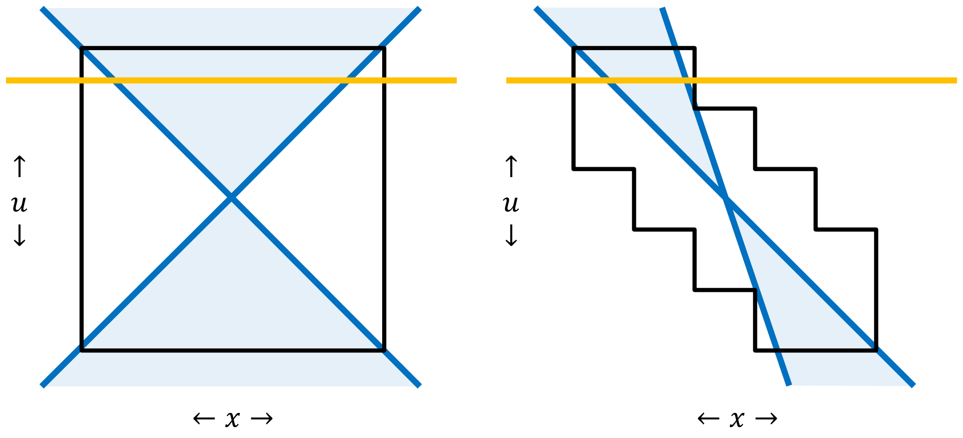


Figure 53. Linear panorama performance

For example, the yellow lines on Figure 53 show a row needed for chosen parallax view (cf. Figure 25) for full light field data on the left and for the light field data of linear panorama on the right. The combined light field does not provide views of whole panorama from different directions, because the rays coming from acute angles were not captured.

Similarly, we know the slope of line corresponds to the distance in front of camera. The blue lines denote the nearest and farthest distance at which the image can be focused without any loss of information.

Note that if we were interested only in a subset of the panorama (e.g. user panning through it), full features might still be available, subject to size of the viewport. More interestingly, we have shown that the line of infinity depends on camera parameters, most notably on the distance between main lens and the microlens array. Therefore, if the camera allows changing this distance (either by choosing the initial focused distance or by zooming, both of which Lytro camera provides), we can obtain panorama with different slope and then register the two panoramas to cover some of the missing pieces, see Figure 54 for illustration of this principle.

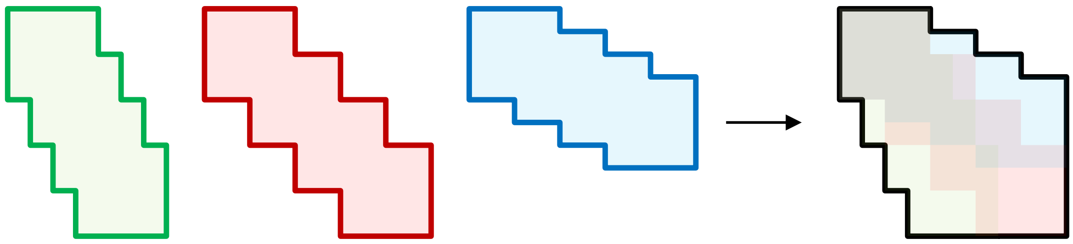


Figure 54. Registering multipe panoramas

It is important to note that we have built the assumption of well-defined registration based on ideal optics and ray projections, while in real cameras, the UX grid will be non-trivially deformed. See the Ren Ng's work [1] for methods of correcting such errors.

4.4. Super-resolution

While the results render the linear panorama as less than ideal application, the super-resolution algorithms could be applied to light fields with promising success. In traditional digital photography, super-resolution or *super-resolution image reconstruction* is a problem of obtaining a higher-resolution image from multiple low-resolution images of the scene, with the key assumption that there is a shift between the low-resolution images and we are able to detect this shift with subpixel accuracy [27]. Formally, the observation model is defined as

$$y_k = \boldsymbol{D}\boldsymbol{B}_k\boldsymbol{M}_k\vec{x} + \vec{n}_k \quad \text{for } 1 \leq k \leq p\,,$$

where $\vec{x}$ is the ideal high-resolution image, modified by the warp matrix $\mathbf{M}_k$ (representing translation, rotation etc. of the camera), degraded by the blur matrix $\mathbf{B}_k$ (motion blur, optical errors etc.) and downsampled by the sensor as modelled by the matrix $\mathbf{D}$. $\vec{n}$ represents an additive noise and k denotes the individual low-resolution images from the set of p images available. The task is to find $\vec{x}$ given set of such equations for $1 \leq k \leq p$. An overview of various approaches for solving this problem is given in the IEEE Signal Processing magazine [27].[42]

The light field photography, inherently capturing a scene from multiple directions, is a great candidate for increasing spatial resolution by means of the super-resolution methods, and already has been subject to an active research in the field [28]. Lytro must be also taking advantage of this technique, bringing 330 microlenses to the final resolution of 1080 pixels.

To my knowledge though, no publication so far considered registering multiple light fields as suggested in the previous chapter, to increase both spatial and directional information. While improving the spatial resolution has well-known effect of providing more detail in the scene, increasing the directional resolution would allow finer control of the focused distance and enable further decrease of the depth of field.

4.5. Future work

Another not really well described area are the visual effects of lower resolution or less data in either spatial or directional domain but not in the other. If there are parts in the image with less or more information in one of the domains, what artefacts will appear in the image and when? Can microlens arrays with lenses of variable size, focal length and other parameters provide additional value?[43]

Aberrations and other errors prevalent in real-world acquisition of light fields complicate theoretical research and reproducible experiments in the field. Synthetic light fields could help, but little options are currently available to public. For consuming light fields, the Stanford Computer Graphics Laboratory published its LFDisplay software.[44] The MIT Media Lab made available couple of POV-Ray generated light field pictures[45] but has not published the script itself. Implementing light field cameras in popular rendering engines would definitely be useful and challenging task.

[42] The whole May 2003 issue of the magazine is dedicated to the topic of super-resolution.
[43] Lytro suggests some configurations in one of their patents [32] and Raytrix uses different focal lengths in their *multi-focus plenoptic camera* to extend the depth of field [33].
[44] http://graphics.stanford.edu/software/LFDisplay/
[45] http://web.media.mit.edu/~gordonw/SyntheticLightFields/

Conclusion

The aim of the work was to become familiar with the physics of light field photography and various rendering techniques needed to process the digital light field images. The work describes in detail how the light field looks like and proves that recording it through micro lens array is well defined and can be used the way that it is being used in other, advanced works. The most common rendering operations (refocusing, depth of field and parallax) are explained and illustrated. Overall, the work uses 2D cases where possible to demonstrate the topics discussed, which is also rarely seen in related works. Latest and emerging hardware was summarized as well as the ongoing research in other usages of the light field, with couple of novel ideas suggested, too.

The main contribution of this work is the description of the Lytro camera, the file formats and communication protocols it uses, packaged into a .NET portable class library. Two desktop applications built on the library are included, the light field picture viewer with editing capabilities and the communicator to interact with the camera over WiFi, with the source code available on the accompanying media.

This is the first time similar software and developer library for the Lytro camera is released. Some of the features (e.g. remotely triggering the shutter) are not available in the official software at all, despite their importance in research applications. Similarly, the camera's firmware image was obtained and is for the first time available on the accompanying media for additional examination together with instructions for manual firmware update, which opens completely new area of possibilities.

This work is the first step of enabling the use of consumer light field cameras in further research and the response and support from various universities and research institutions over the world so far suggests it is a welcome contribution. Readers can find up-to-date software, documentation and analysis at http://lytro.miloush.net/.

Finally, the options of linear panoramas were analysed. It is apparent that the impact of light fields in these panoramas is limited. Still, considerable performance improvements are available for the registration, which calls for effective, multiple light field super-resolution algorithms to be developed.

Light field technology is currently experiencing boom in both consumer market and research. I hope that the hardware manufacturers will catch up soon and users will find value in the core essence of light fields – capturing as much information in the scene as possible – rather than celebrating cheap visual tricks like refocusing and depth maps.

Bibliography

[1] R. Ng, *Digital Light Field Photography,* Stanford University, 2006.

[2] C. Birklbauer and O. Bimber, “Panorama Light-Field Imaging,” in *Eurographics 2014 Proceedings*, Strasbourg, 2014.

[3] F. E. Ives, “Parallax stereogram and process of making same”. United States of America Patent 725567, 14 April 1903.

[4] J. Spoel, Artist, *Damesgezelschap bekijkt stereoscoopfoto's.* [Art]. Rijksmuseum Amsterdam, 1868.

[5] G. Lippman, “Épreuves réversibles. Photographies intégrales.,” *Comptes Rendus de l'Académie des Sciences,* vol. 9, no. 146, pp. 446-451, 2 March 1908.

[6] T. Georgiev, “100 Years Light-Field,” 2008. [Online]. Available: http://www.tgeorgiev.net/Lippmann/100_Years_LightField.pdf. [Accessed 27 May 2014].

[7] E. Stoykova and V. Sainov, “Institute of Optical Materials and Technologies,” [Online]. Available: http://www.clf.bas.bg/page_06/Integral_Imaging.pdf. [Accessed 6 May 2014].

[8] M. Levoy and P. Hanrahan, “Light Field Rendering,” in *SIGGRAPH '96*, 1996.

[9] S. J. Gortler, R. Grzeszczuk, R. Szeliski and M. F. Cohen, “The Lumigraph,” in *SIGGRAPH '96*, 1996.

[10] M. Faraday, “Thoughts on Ray-vibrations,” *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science,* vol. 28, no. 188, pp. 345-350, May 1846.

[11] A. Gershun, The Light Field, Moscow, 1939.

[12] E. H. Adelson and J. R. Bergen, “The Plenoptic Function and the Elements of Early Vision,” *Computational Models of Visual Processing,* pp. 3-20, 1991.

[13] A. Gerrard and J. M. Burch, Introduction to Matrix Methods in Optics, Wiley Series in Pure and Applied Optics ed., London: J. Wiley, 1975.

[14] J. Košecká, Y. Ma, S. Soatto and R. Vidal, *Multiple-View Geometry for Image-Based Modeling,* Los Angeles: Siggraph Course #23, 2004.

[15] Lytro, Inc., "The Science Inside Living Pictures," 5 December 2011. [Online]. Available: http://blog.lytro.com/post/57720255142/the-science-inside-living-pictures. [Accessed 30 May 2014].

[16] M. Hiramoto, Y. Ishii and Y. Monobe, "Light Field Image Capture Device and Image Sensor". United States of America Patent US 2014/0078259, 20 March 2014.

[17] V. Boominathan, K. Mitra and A. Veeraraghavan, "Improving Resolution and Depth-of-Field of Light Field Cameras Using Hybrid Imaging System," in *IEEE International Conference on Computational Photography*, Santa Clara, 2014.

[18] K. Marwah, G. Wetzstein, Y. Bando and R. Raskar, "Compressive Light Field Photography using Overcomplete Dictionaries and Optimized Projections," in *SIGGRAPH 2013 Conference Proceedings*, New York, 2013.

[19] L. Hardesty, "Multiview 3-D photography made simple," MIT News Office, 19 June 2013. [Online]. Available: http://newsoffice.mit.edu/2013/multiview-3d-photography-made-simple-0619. [Accessed 27 May 2014].

[20] C.-K. Liang, T.-H. Lin, B.-Y. Wong, C. Liu and H. H. Chen, "Programmable Aperture Photography: Multiplexed Light Field Acquisition," *ACM Transactions on Graphics,* vol. 27, no. 3, 2008.

[21] J. Sun, N.-N. Zheng and H.-Y. Shum, "Stereo Matching Using Belief Propagation," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 25, no. 7, pp. 787-800, 2003.

[22] K. Venkataraman, D. Lelescu, J. Duparré, A. McMahon, G. Molina, P. Chatterjee, R. Mullis and S. Nayar, "PiCam: An Ultra-Thin High Performance Monolithic Camera Array," in *SIGGRAPH 2013 Conference Proceedings*, Hong Kong, 2013.

[23] M. Levoy, R. Ng, A. Adams, M. Footer and M. Horowitz, "Light Field Microscopy," in *SIGGRAPH 2006 Conference Proceedings*, Boston, 2006.

[24] Lytro Inc, "Science Inside | Lytro," [Online]. Available: http://www.lytro.com/science_inside. [Accessed 27 4 2012].

[25] H. S. Malvar, L.-W. He and R. Cutler, "High-Quality Linear Interpolation for Demosaicing of Bayer-Patterned Color Images," in *International Conference of Acoustic, Speech and Signal Processing*, 2004.

[26] C. Birklbauer and O. Bimber, "Panorama light-field imaging," in *ACM SIGGRAPH 2012 Talks*, Los Angeles, 2012.

[27] S. C. Park, M. K. Park and M. G. Kang, "Super-Resolution Image Reconstruction: A Technical Overview," *IEEE Signal Processing Magazine,* vol. 20, no. 3, pp. 21-36, 2003.

[28] T. Georgiev, G. Chunev and A. Lumsdaine, "Superresolution with the Focused Plenoptic Camera," in *SPIE Electronic Imaging*, San Francisco, 2011.

[29] D. G. Dansereau, O. Pizarro and S. B. Williams, "Decoding, Calibration and Rectification for Lenselet-Based Plenoptic Cameras," in *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, Portland, 2013.

[30] H. Choi, S.-W. Min, S. Jung, J.-H. Park and B. Lee, "Multiple-viewing-zone integral imaging using a dynamic barrier array for three-dimensional displays," *Optics Express,* vol. 11, no. 8, pp. 927-932, 2003.

[31] M. Rose, M. Baierl, A. Koch and K. Bobey, "Compact Digital Pinhole Camera," in *Deutschen Gesellschaft für angewandte Optik Proceedings 2012*, 2012.

[32] Y.-R. Ng, P. M. Manrahan, M. S. Levoy and M. A. Horowitz, "Imaging arrangements and methods therefor". United States of America Patent US 7,936,392 B2, 3 May 2011.

[33] C. Perwass and L. Wietzke, "Single lens 3D-camera with extended depth-of-field," in *Proceedings SPIE 8291, Human Vision and Electronic Imaging XVII, 829108*, California, 2012.

Appendix A. List of acquired MLA configurations

The critical part when working with light field pictures is calibration – in the case of microlens arrays, determining the location of individual microlenses. Scientists having access to the camera can calibrate it manually by taking an image through white diffuser or of white scene, or using such images found in the Lytro calibration data set [29].

However, this approach is unavailable for processing images from unknown cameras, where the calibration data is not available, yet still the Lytro Desktop can render these images without noticeable loss of quality. The software that is part of this thesis therefore focused on interpreting the image metadata in a way that would allow to obtain microlens positions within reasonable accuracy.

To verify the results, a test set of images from several Lytro users and few online sites were collected. The test set, anonymized, is available on the accompanying medium. A list of microlens array parameters as reported by the metadata follows:

picture	rotation [$mrad$]	offset [μm]			pitch [μm]		scale	
		x	y	z	lens	pixel	x	y
01.1	**-6.0540**	**+3.6736**	+2.3655	25	13.899	1.400	1	1.0004
01.2	-6.0491	+3.6726	+2.3695	25	13.899	1.400	1	1.0005
01.3	-6.0389	+3.6682	+2.3503	25	13.899	1.400	1	1.0004
02	-4.8322	-1.3038	-4.6239	25	14.000	1.400	1	1.0002
03	-2.6991	-2.0778	-11.221	25	13.899	1.400	1	1.0005
04	-2.5191	-3.9254	-01.008	25	13.899	1.400	1	1.0004
05	-1.9234	-1.5833	**-13.640**	25	14.000	1.400	1	1.0002
06.1	-1.6539	-6.0344	-7.0080	25	13.899	1.400	1	1.0008
06.2	-1.6539	-6.0344	-7.0080	25	13.899	1.400	1	1.0008
07.1	-1.3552	+2.7154	-1.3819	25	14.000	1.400	1	1.0003
07.2	-1.3335	+2.7641	-1.4393	25	14.000	1.400	1	1.0002
08	-0.9115	+1.1833	+0.4784	25	14.000	1.400	1	1.0004
09	-0.8710	-0.0093	-2.3828	25	13.899	1.400	1	1.0005
10	-0.4326	+2.4290	-5.5333	25	13.899	1.400	1	1.0006
11	-0.2843	-5.3041	-1.1149	25	14.000	1.400	1	1.0002
12.1	-0.2815	-3.5040	-1.6299	25	13.899	1.400	1	1.0004
12.2	-0.2815	-3.5040	-1.6299	25	13.899	1.400	1	1.0004
12.3	-0.2712	-3.5089	-1.6371	25	13.899	1.400	1	1.0005
12.4	-0.2607	-3.5433	-1.6147	25	13.899	1.400	1	1.0005
13	-0.0542	-5.9970	-1.1573	25	13.899	1.400	1	1.0007
14	+0.7737	+0.4470	-1.7432	25	13.899	1.400	1	1.0005
15	+1.3410	-3.3231	+1.2399	25	13.899	1.400	1	1.0004
16.1	+1.9291	-2.7139	+0.5660	**25**	13.899	1.400	1	1.0005
16.2	+1.9291	-2.7139	+0.5660	**25**	13.899	1.400	1	1.0005
16.3	+1.9391	-2.7125	+0.5629	**25**	13.899	1.400	1	1.0005
16.4	+1.9391	-2.7125	+0.5629	**25**	13.899	1.400	1	1.0005
17	+1.9927	-6.5979	**+2.5568**	**23**	13.899	1.400	1	1.0008
18	+2.2094	**-8.3793**	+2.5161	25	13.899	1.400	1	1.0006
19	+2.9010	-1.2626	-6.3853	25	13.899	1.400	1	**1.0016**
20.1	+3.5985	+2.9764	-0.7565	25	14.000	1.400	1	1.0003
20.2	+3.6050	+2.9653	-0.7697	25	14.000	1.400	1	1.0003
21	+6.8376	-6.3311	-1.1998	25	14.000	1.400	1	1.0005
22	**+8.7578**	-5.1163	-8.1219	25	14.000	1.400	1	**1.0001**

Pictures with the same major numbers come from the same camera. Different values for the same camera suggests that the metadata contains best-fit approximate generated on the fly rather than values obtained during the calibration process.

Minimum and maximum values are highlighted. It is worth pointing out that the rotations of the microlens array relative to the sensor itself are spanning very large values and cannot be ignored. For example, a rotation of -0.006 radians of picture 01.1 on 3,280 px resolution causes shift of 20 px, twice the size of the microlens.

See the source code for `MicroLensCollection` for computation of the microlens centres used by the accompanying software.

Appendix B. List of files on Lytro camera

The following files were discovered on the camera's internal storage as of firmware version v1.2.2 (build v1.0a208). Not all files are present on all cameras (e.g. various log files) and the list might not be exhaustive as its source was firmware analysis rather than storage dump. Readers can download these files for further research with the help of supplied software over wireless connection.

The storage uses FAT file system.

`A:\` for firmware's internal use, USB descriptors and mounted media, media assets, camera model information, version compatibility

- assets
  - asset-atlas-deDE.img
  - asset-atlas-deDE.txt
  - asset-atlas-enUS.img
  - asset-atlas-enUS.txt
  - asset-atlas-frCA.img
  - asset-atlas-frCA.txt
  - assets.txt
  - menus.txt
- LUA
  - CHARGETO.LUA
  - MENU.LUA
- MCU
  - FIREFLY.TXT.BIN
- media
  - default.bin
- MODELS
  - MODEL.TXT
- ADC.BIN
- AE.BIN
- AF.BIN
- AVIMODELSTR.BIN
- AVISTRLSTR.BIN
- AVISTRNSTR.BIN
- AWBCFG.BIN
- AWBSETTINGS.BIN
- BASENLGF0.BIN
- BASENLGF2.BIN
- COPMASKING.BIN
- COPNMRLOOP1.BIN
- COPQTTABLE.BIN
- COPTRANSFORM.BIN
- CTLUT1.BIN
- DLUT.BIN
- EPS_GCP0.BIN
- Eps_XSCL.BIN
- FIRMWARE.TXT
- GAMMADDE1.BIN
- GAMMALUT0.BIN
- LCLUT0.BIN
- MEDIAFORMAT.BIN
- UsbDevDesc.BIN
- UsbModeDesc.BIN
- VCM.TXT
- WAVEEXIF.BIN
- YLUT.BIN

`B:\` operational storage, file system check logs, user settings, usage statistics

- T2CALIB
  - BIPOS.BIN
- wifi
  - settings.txt
- ASERIAL.TXT
- CALREFSHA1.TXT
- FSCK.LOG
- HWSERIAL.TXT
- LENSODOMETER.TXT
- SETTINGS.TXT
- STATE.TXT
- USER.TXT

`C:\` calibration data, flat field images, partition read-only

- B1CALIB
  - TIMESTAMP.TXT
- CALIB
  - ACC.TXT
  - BATTERY.TXT
  - CAPSLIDER.TXT
  - EMMC_INFO.TXT
  - HISTORY.TXT
  - HW_VERSION.TXT
  - ITE_H.TXT
  - ITE_V.TXT
  - SENSORID.TXT
  - THERMISTOR.TXT
  - THROUGHPUT.TXT
  - TOUCHPANEL.TXT
  - WIFI_MAC_ADDR.TXT
- L50CALIB
  - MLACALIBRATION.TXT
- media
  - default.bin
- MODELS
  - MODEL.TXT
- ADC.BIN
- AE.BIN
- AF.BIN
- AVIMODELSTR.BIN
- AVISTRLSTR.BIN
- DEVICES
  - ACCELEROMETER.TXT
  - BOARDS.TXT
  - CAPSLIDER.TXT
  - COLOR.TXT
  - EMMC.TXT
  - GASGAUGE.TXT
  - LENS.TXT
  - MLA.TXT
  - SENSOR.TXT
  - THERMISTORS.TXT
  - TOUCHPANEL.TXT
  - WIFI.TXT
- T1CALIB
  - GCFN_ZZZZ_FFFF.BIN[1]
  - MLACALIBRATION.TXT
  - MOD_0000.RAW[2]
  - MOD_0000.TXT[2]
- T2CALIB
  - BIPOS.TXT
  - HOTPIXEL.BIN
  - HOTPIXEL.RAW
  - HOTPIXEL.TXT
- DEFECTIVEPIXEL0.BIN
- FSP.BIN
- LSCLUT0.BIN
- LSCLUT1.BIN

[1] **ZZZZ** is for zoom step, FFFF for focus step; an example set of combinations:

0100	**0115**	**0155**	**0230**	**0360**	**0490**	**0620**	**0740**	**0860**	**0982**
0835	0913	1067	1215	1219	1099	0957	0829	0713	0603
1032	1080	1238	1311	1313	1253	1082	0904	0763	0653
1229	1247	1409	1407	1407	1407	1207	0979		
1426	1414								

[2] files range from 0000 through 0061

`I:\` user data (pictures), crash and error logs

- DCIM
  - 100PHOTO[1]
    - IMG_0000.???[1,2]
- CRASH000.LOG[1]
- err.log
- RTCERROR.LOG

[1] depending on the number of items, number might grow
[2] available extensions are 048, 128, 133, INF, JPG, RAW, STK, TXT

Appendix C. Accompanying media

A DVD with the following structure complements this thesis:

Camera		
	Calibration	Compression calibration data from two cameras.
	Firmware	Firmware version v1.0a208. This is a stream format including metadata, which can be directly uploaded to a camera.
	Storage	Files downloaded from internal storage of two cameras. One running firmware v1.0a208, includes full T1 calibration. Other one running firmware v1.0a204.
Literature		Bibliography sources when available.
Pictures		
	Processed	Sample pictures processed with the Lytro Desktop Software.
	Raw	Acquired raw light field pictures, see Appendix A for their metadata overview. Same numbers group pictures from the same camera. Files can be opened either using the supplied software or the Lytro Desktop software.
Software		Includes binaries of the library (chapter 3.3.3), the accompanying software (Lytro Compatible Viewer and Lytro Compatible Communicator, ref. chapter 3.5) and their user manuals.
	Source	Source codes for the library and accompanying software.
	Supplementary	2D light field raytracing software that generated illustrations in this thesis.
	Support	Microsoft .NET Framework 4.5.2 Offline Installer .NET Portable Class Library Profile1

You might want to:

Try the Lytro Compatible Viewer

- ▶ Ensure you have Microsoft .NET Framework 4.5 installed.
  If not, use the installer in `Software\Support` directory.
- ▶ Run the viewer from `Software` directory.
- ▶ Try opening some of the sample pictures in `Pictures` directory.
- ▶ See the user manual in `Software` directory for detailed description of available features and user interface.

Explore how the light field works

- Ensure you have Microsoft .NET Framework 4.5 installed.
  If not, use the installer in `Software\Support` directory.
- Run the `LightFieldGeometry` from `Software\Supplementary` directory.
- Play with the camera parameters and see how it affects the captured image.

Control your Lytro camera wirelessly

- Ensure you have Microsoft .NET Framework 4.5 installed.
  If not, use the installer in `Software\Support` directory.
- Run the communicator from `Software` directory.
- Follow the user manual in the same directory to connect to the camera and learn what the software can do.
- See Appendix B on what files can be downloaded from the camera.

Check your software works with real-world pictures

- Use the acquired set in `Pictures\Raw` directory.

Use the library to build your software

- If you need to use the .NET Portable Class Library, add reference to the `LytroCompatibleLibrary.dll` assembly in the `Software` directory.
- If you are building desktop software and want to take advantage of asynchronous methods and file system, add reference to the `LytroCompatibleLibrary.Desktop.dll` assembly instead.
- Check the chapter 3.4 for documentation and examples for the library. You can also visit http://lytro.miloush.net/ for even more examples and newer releases of the library.

Build the supplied software yourself

- Ensure you have Visual Studio 2013 or newer installed.
  If not, install it from www.visualstudio.com.
- *Note:* If you want to build the .NET Portable Class Library for the original set of platforms, you need to overwrite the `Profile1` directory in `%PROGRAMFILES(X86)%\Reference Assemblies\Microsoft\Framework\.NETPortable\v4.0\Profile` with the one in `Software\Support` directory. Otherwise, you would be asked to upgrade the portable library project and miss the ability to target Windows Phone 7, Silverlight 4 and Xbox 360.
- If you are copying the source code of the media, make sure you include the `InsertIcons.exe` from `Software\Support` directory which is used during the build process.

Reveal further secrets about the Lytro camera

- Study the enclosed firmware release under `Camera\Firmware` directory.
- Publish your findings!

Appendix D. Imprint

Printed version of the thesis contains inserted copies of the following press articles:

Inside the Lytro
published on February 29, 2012 in New York Times
http://www.nytimes.com/interactive/2012/03/01/business/inside-the-lytro.html

Forget the autofocus: how Lytro cameras work
published on June 15, 2012 in Wired magazine
http://www.wired.co.uk/magazine/archive/2012/07/start/photos-in-full-focus

New Thing in Photography
published on May 24, 1908 in New York Times
http://query.nytimes.com/mem/archive-free/pdf?res=9D07E5D8143EE233A25757C2A9639C946997D6CF